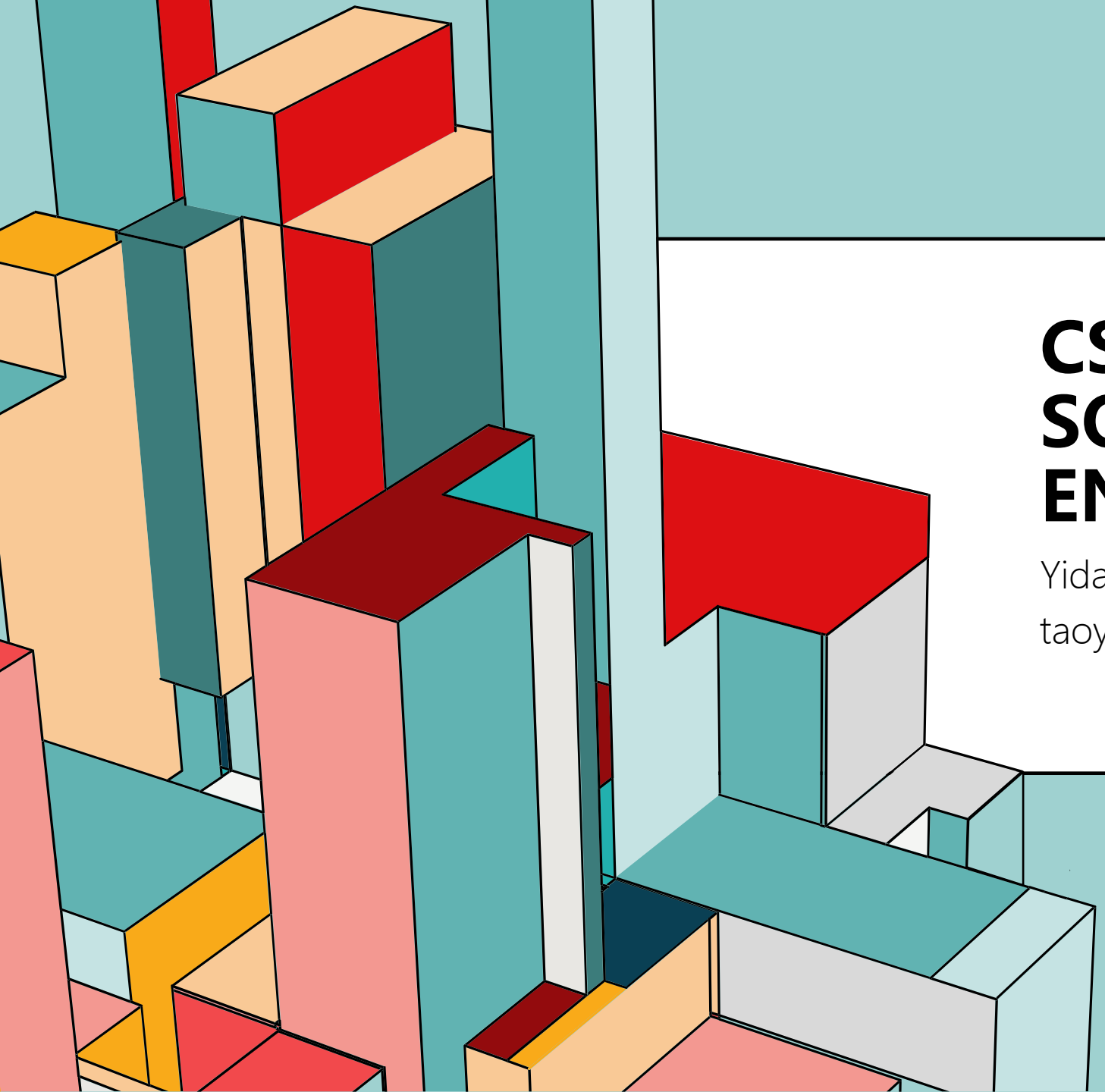


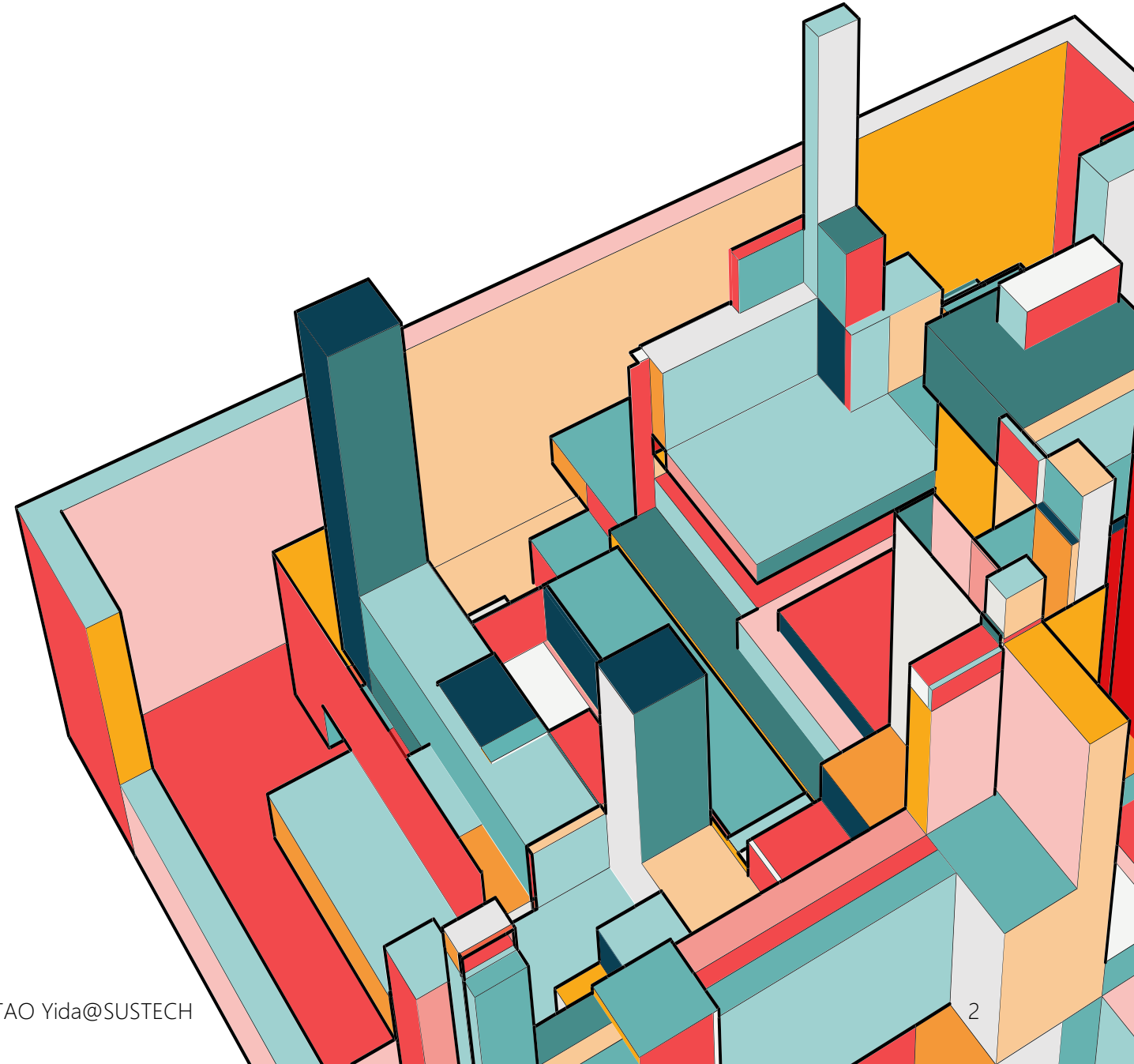
CS304 SOFTWARE ENGINEERING

Yida Tao

taoyd@sustech.edu.cn

LECTURE 2

- Software Process Overview
- Process Models
- Agile Development
- DevOps



WHY SOFTWARE PROCESS?

- Writing code is easy

- Engineering good software system is HARD
  - Different roles are involved (customers, developers, managers, etc.)
  - Many aspects of complexity
  - Trade-offs & making good decisions

WHY SOFTWARE PROCESS?

- Organizations want a well-defined, well-understood, repeatable software development process

- Benefits
  - New team members know what to do
  - Know what's been done
  - Estimate time to completion, costs, etc.
  - Follow and repeat good practices

WHAT IS SOFTWARE PROCESS?

Software process is a set of related activities that take place in sequence, and leads to the production of a software product. All processes have some form of the following activities:

Assignment Description

- Software specification: The functionality of the software and constraints on its operation must be defined.

Assignment code

- Software design and implementation: The software to meet the specification must be produced.

OJ Testing

- Software validation: The software must be validated to ensure that it does what the customer wants.

???

- Software evolution & maintenance: The software must evolve to meet changing customer needs.

The form/iteration/timing/execution of these activities defines different families and methodologies of software process

WHAT IS SOFTWARE PROCESS?

Broadly speaking, process models and thus methodologies fall on a continuum

- **Plan-driven processes:** processes where all of the process activities are planned in advance and progress is measured against this plan.
- **Agile processes:** planning is incremental and it is easier to change the process to reflect changing customer requirements.
- Generally, you need to find a balance between plan-driven and agile processes.

Agile — Plan-driven

There is no ideal process and most organizations have developed their own software development processes.

WHAT IS SOFTWARE PROCESS?

In addition to activities, process may also include products, roles, pre- and post-conditions

- **Products,** which are the outcomes of a process activity.
  - For example, the outcome of the activity of architectural design may be a model of the software architecture.

- **Roles,** which reflect the responsibilities of the people involved in the process.
  - Examples of roles are project manager, configuration manager, programmers, customers etc.

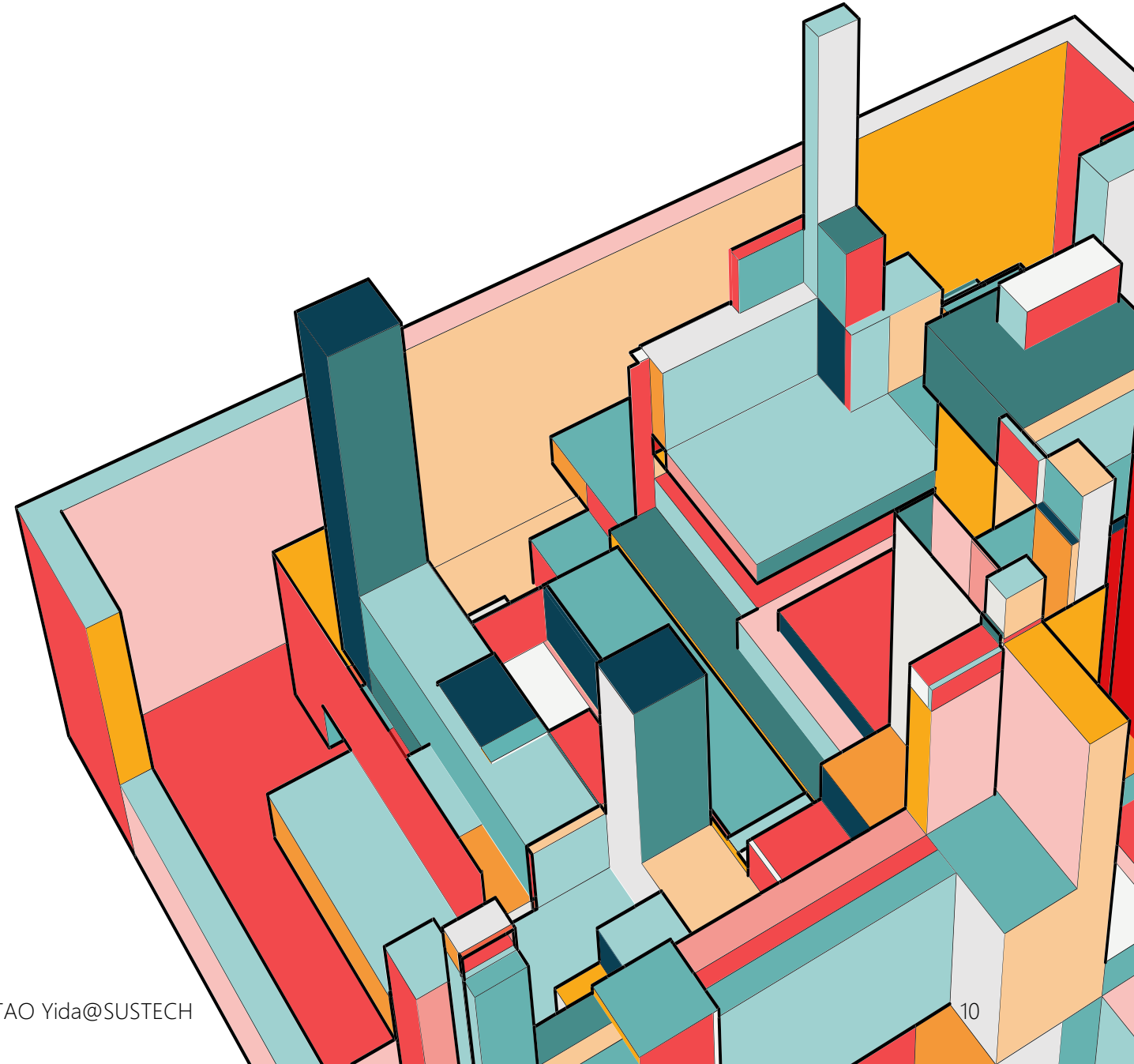
WHAT IS SOFTWARE PROCESS?

In addition to activities, process may also include products, roles, pre- and post-conditions

- **Pre- and post-conditions,** which are statements that are true before and after a process activity has been enacted or a product produced.
  - For example, before architectural design begins, a pre-condition may be that all requirements have been **approved by the customer**;
  - After this activity is finished, a post-condition might be that the UML models describing the architecture **have been reviewed**.

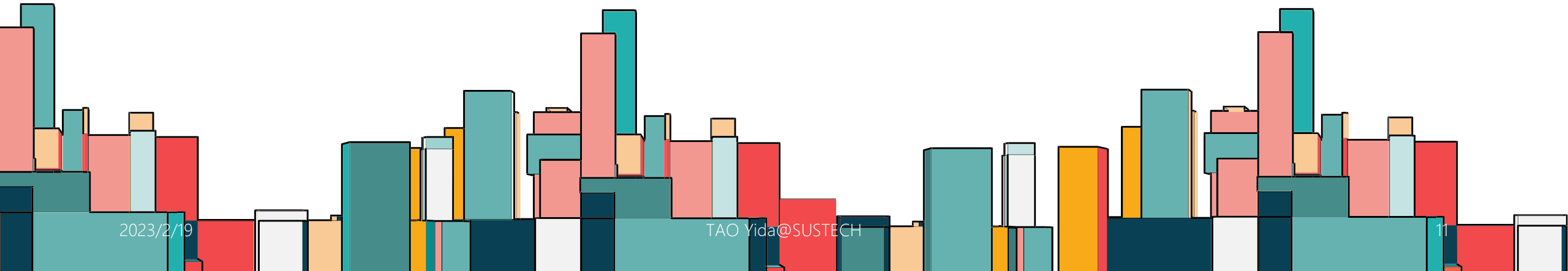
LECTURE 2

- Software Process Overview
- Process Models
- Agile Development
- DevOps



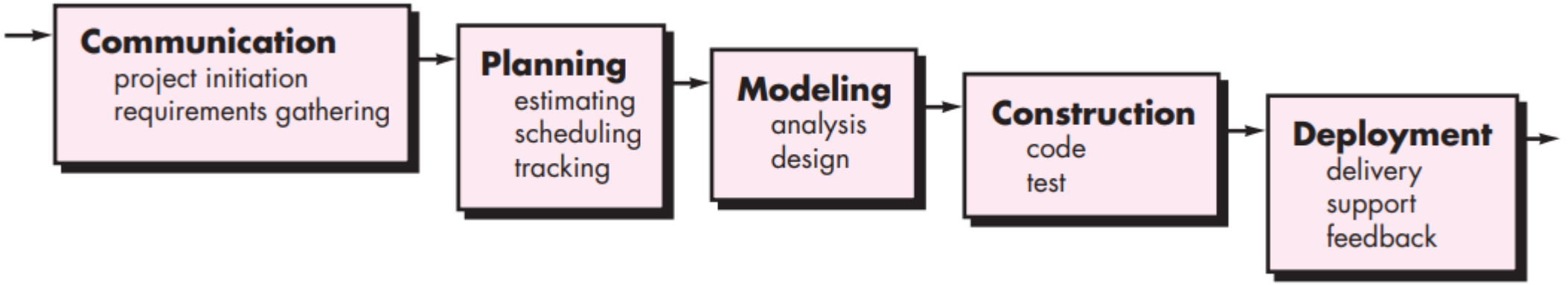
PROCESS MODELS

- Process models were originally proposed to bring order to the chaos of software development
- The general process models (or process paradigms) covered here could be extended or adapted in real practice
  - The Waterfall Model
  - Incremental Process Models
  - Evolutionary Process Models



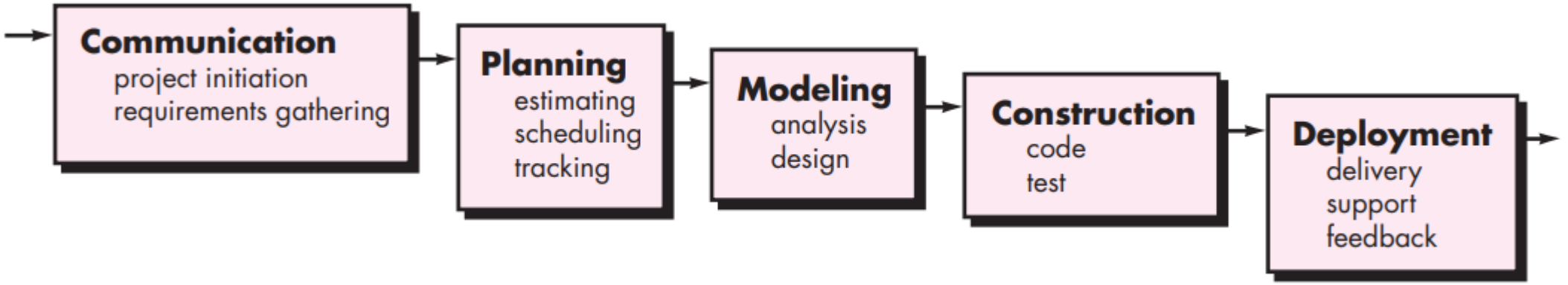
THE WATERFALL MODEL

The waterfall model (classic life cycle), suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through design, implementation, testing, deployment and maintenance



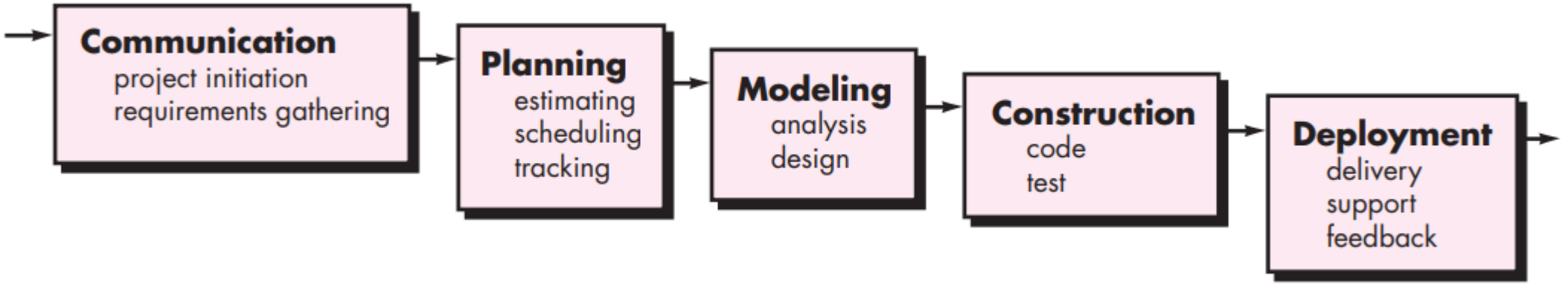
THE WATERFALL MODEL

- In principle, the result of each phase is one or more documents that are approved
- The following phase should not start until the previous phase has finished.



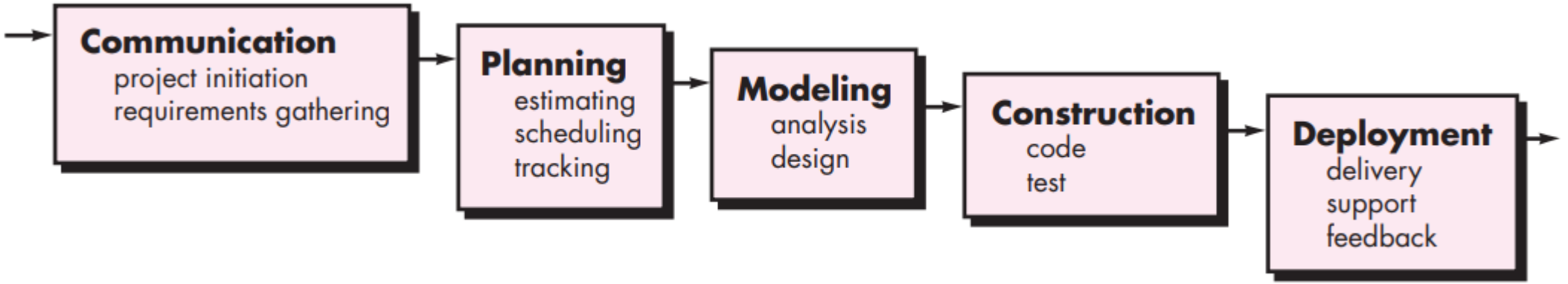
THE WATERFALL MODEL

- **Requirements analysis and definition**: The system's services, constraints, and goals are established by consultation with system users. They serve as a system specification.
- **System and software design**: The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture.



THE WATERFALL MODEL

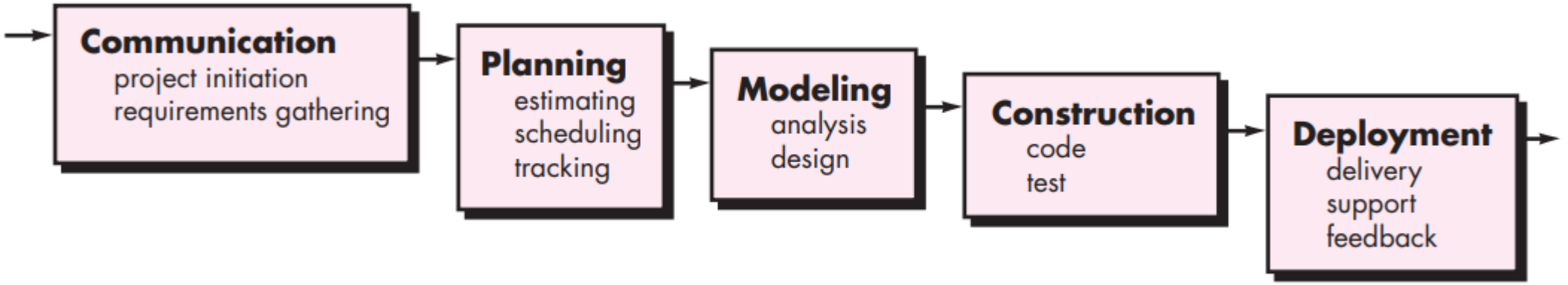
- **Implementation and unit testing**: The software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.
- **Integration and system testing**: The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.



THE WATERFALL MODEL

- **Operation and maintenance**: The system is installed and put into practical use. Maintenance fixes errors and adapt the system to new requirements.

The principal stages of the waterfall model directly reflect the fundamental dev activities



PROBLEMS W/ THE WATERFALL MODEL

Sequential Flow

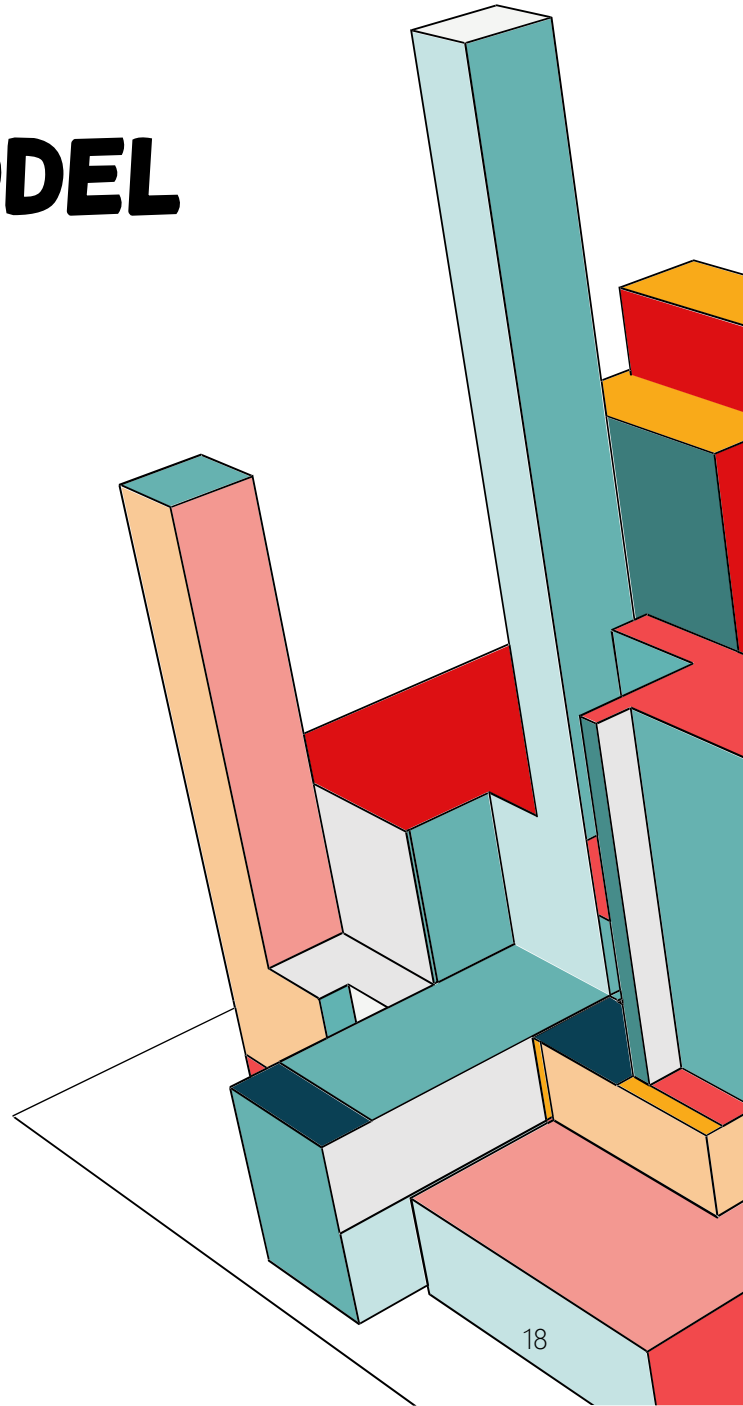
Real projects rarely follow the sequential flow that the model proposes. Although the linear model can accommodate iteration, it does so indirectly. As a result, changes can cause confusion as the project team proceeds.

Clear Requirements

It is often difficult for the customer to state all requirements explicitly. The waterfall model requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of many projects.

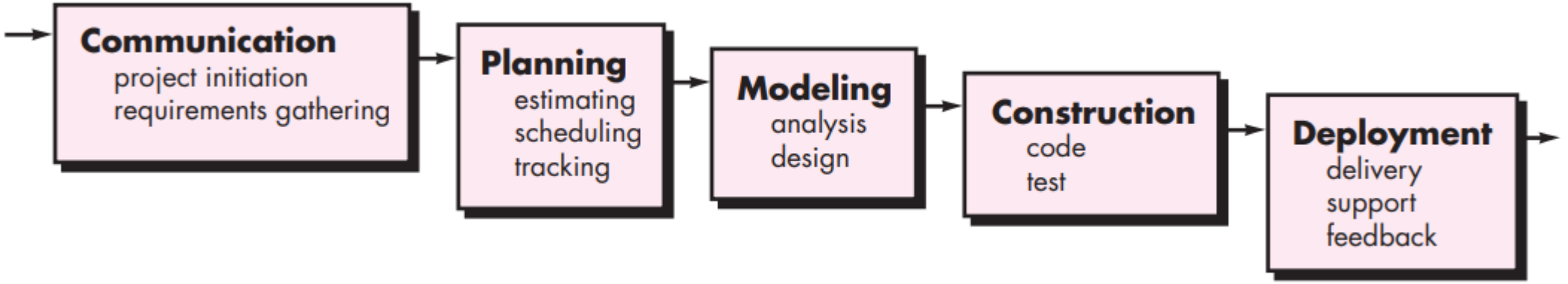
Customer Patience

A working version of the program(s) will not be available until late in the project time span. A major blunder, if undetected until the working program is reviewed, can be disastrous



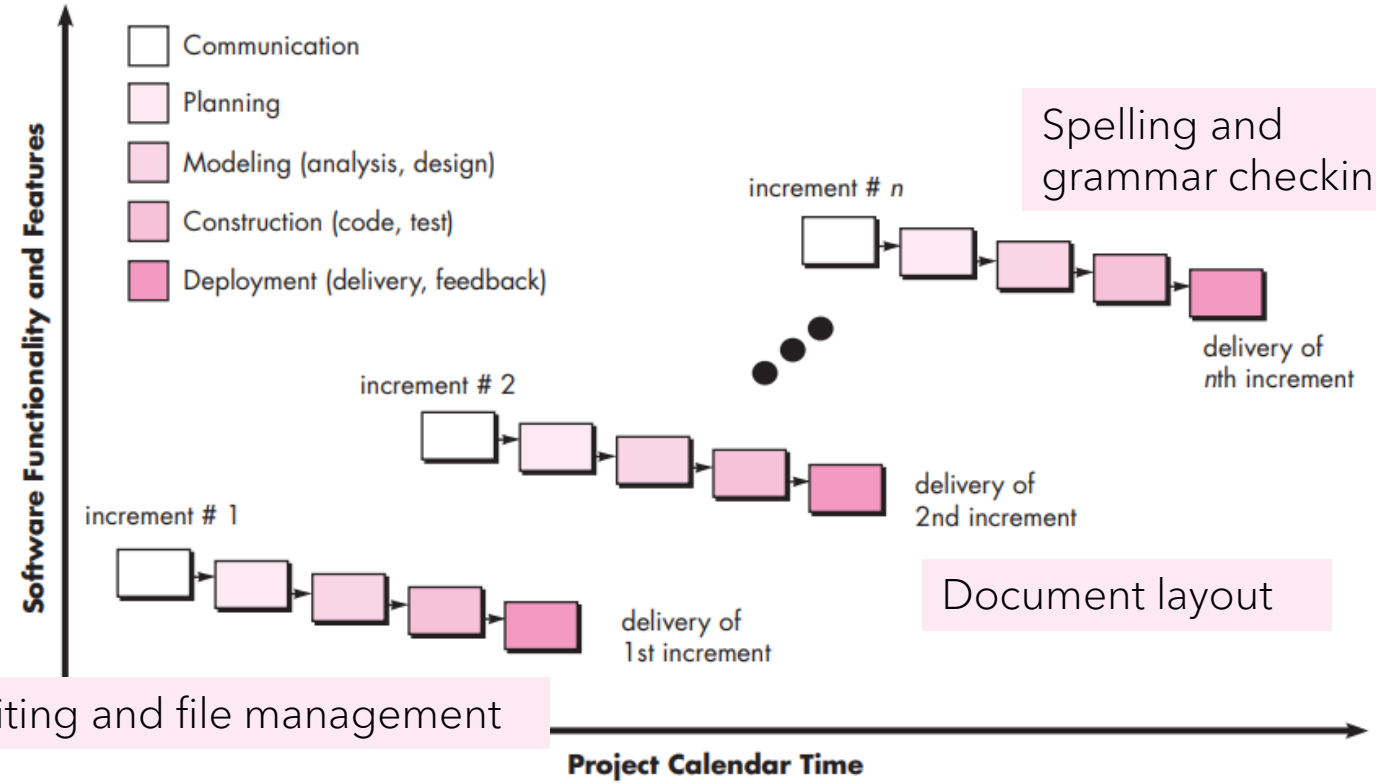
THE WATERFALL MODEL

The waterfall model might be adapted only when requirements are well defined and reasonably stable.



INCREMENTAL PROCESS MODELS

- The incremental model (增量模型) combines elements of linear and parallel process flows
- Each linear sequence produces deliverable "increments" of the software
- The first increment is often a core product with the basic requirements addressed. Supplementary features are developed in subsequent increments.



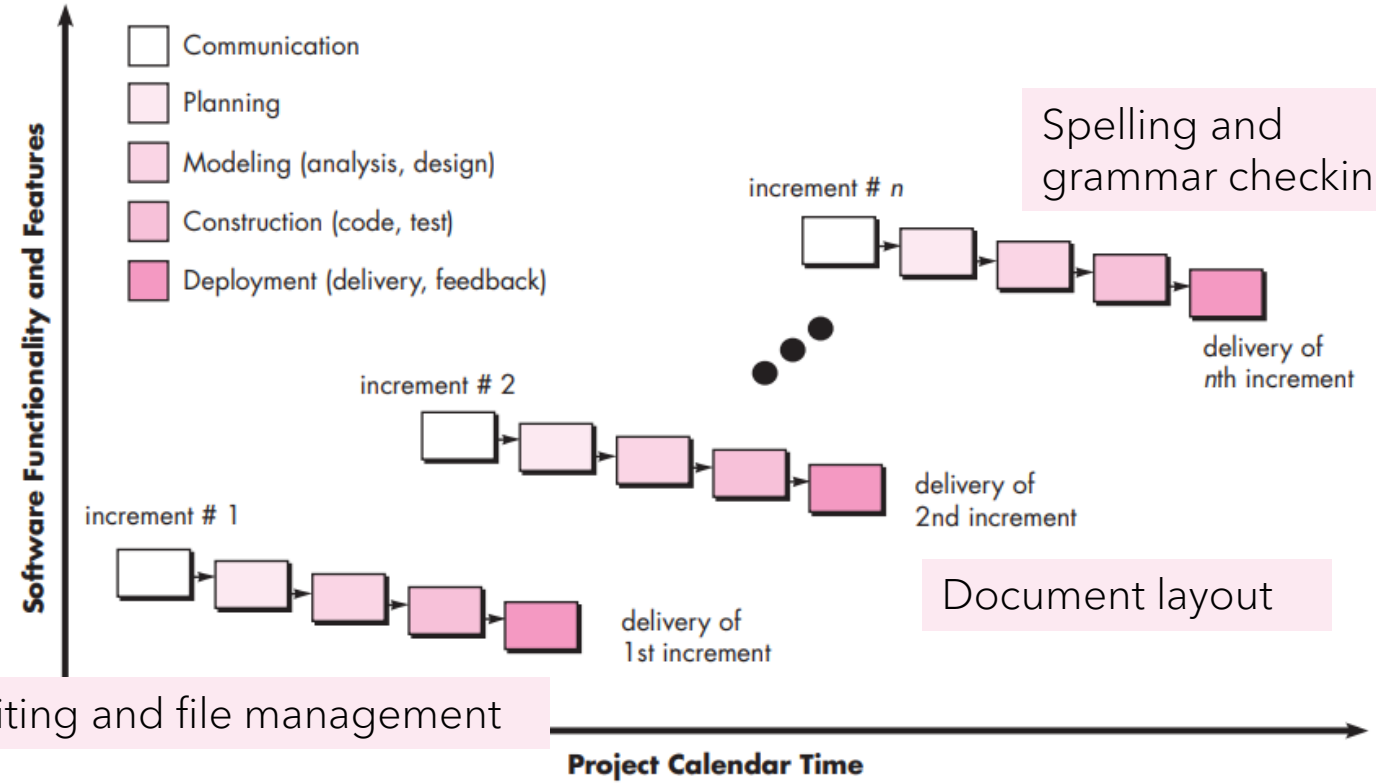
Spelling and grammar checking

Document layout

Basic editing and file management

INCREMENTAL PROCESS MODELS PROS

- Reducing cost due to requirement changes
- Easier to get customer feedback and adapt
- Easier to test and debug during a smaller iteration
- More rapid delivery and deployment of useful software
- Particularly useful when staffing is unavailable for a complete implementation



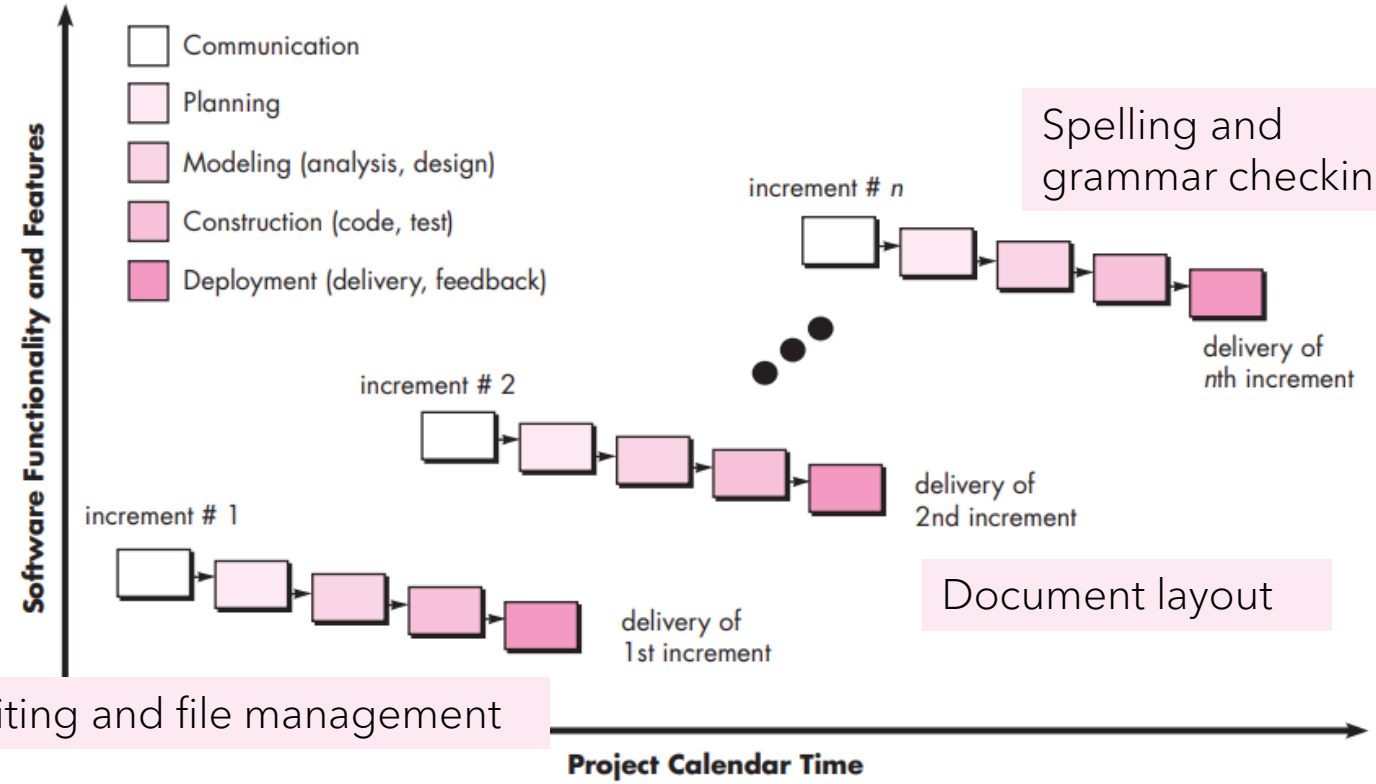
Spelling and grammar checking

Document layout

Basic editing and file management

INCREMENTAL PROCESS MODELS CONS

- Needs good planning and design of the whole system before breaking it down for incremental builds
- System structure tends to **degrade** as new increments are added. Unless effort is spent on **refactoring** to improve the software, regular change tends to corrupt its structure.
- Incorporating further software changes becomes increasingly difficult and costly.

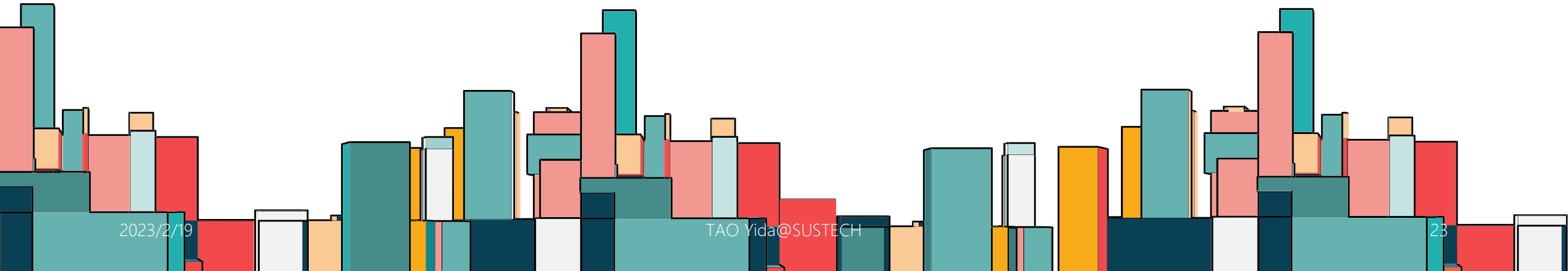


Spelling and grammar checking

Document layout

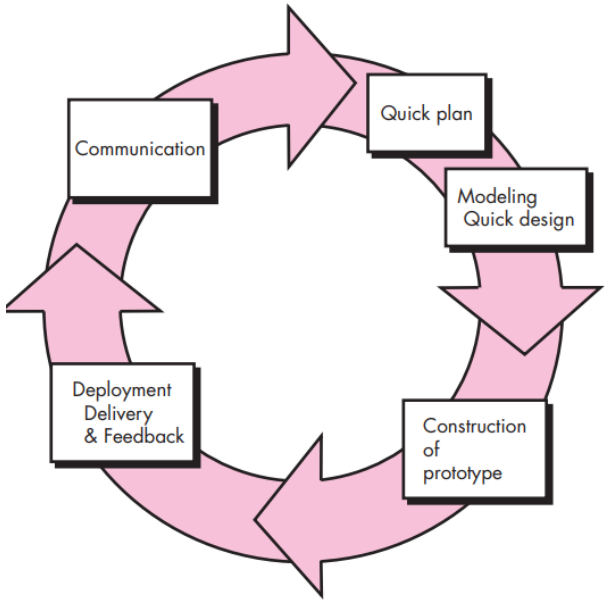
Basic editing and file management

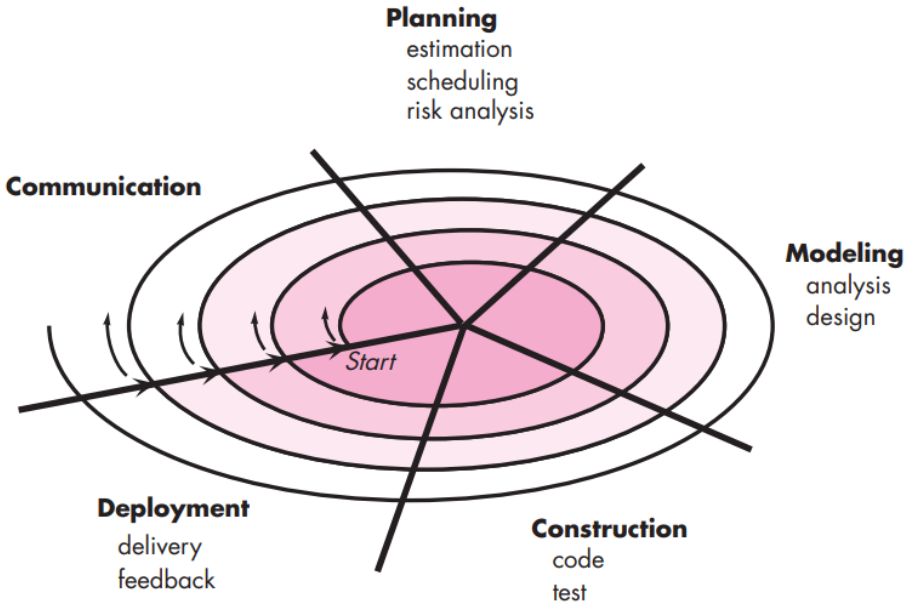
IS SOFTWARE DEVELOPMENT A LINEAR PROCESS?



EVOLUTIONARY PROCESS MODELS

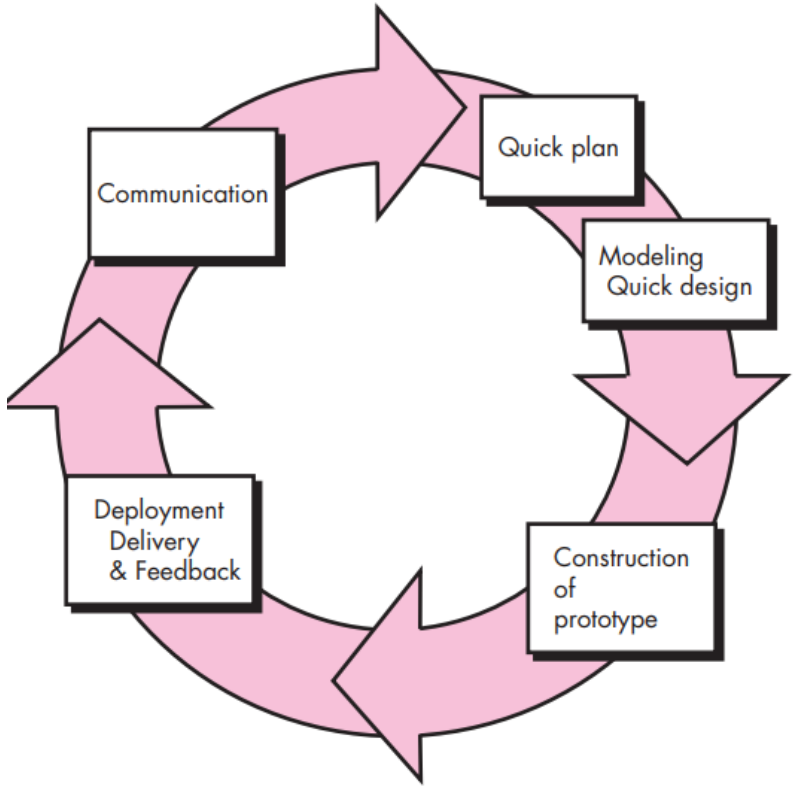
- Evolutionary models are iterative (迭代), which is explicitly designed to accommodate a product that evolves over time.
- Two common evolutionary process models: prototyping and the spiral model





PROTOTYPING

- Prototyping (原型开发) might be used when customers define a set of general objectives for software, but do not identify detailed requirements for functions and features
- The prototype iteration begins with communication, followed by quick planning and modeling, which lead to the construction of a prototype that can be delivered and evaluated
- Stakeholders could quickly see what appears to be a working version of the software and give feedback

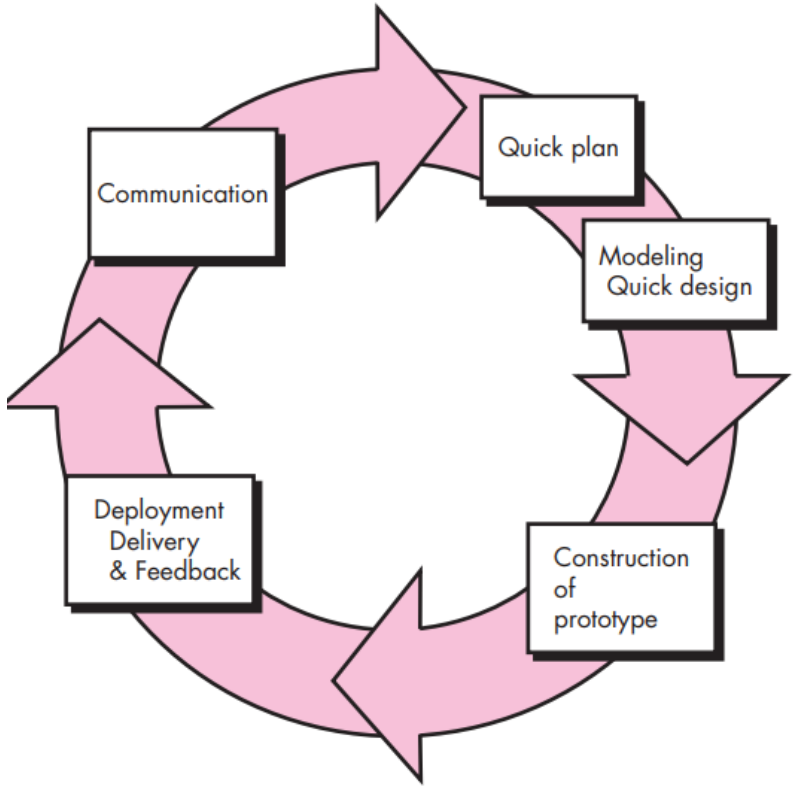


PROTOTYPING

- Developers often make implementation compromises, e.g., inappropriate OS, PL, and inefficient algorithms, in order to get a prototype working quickly

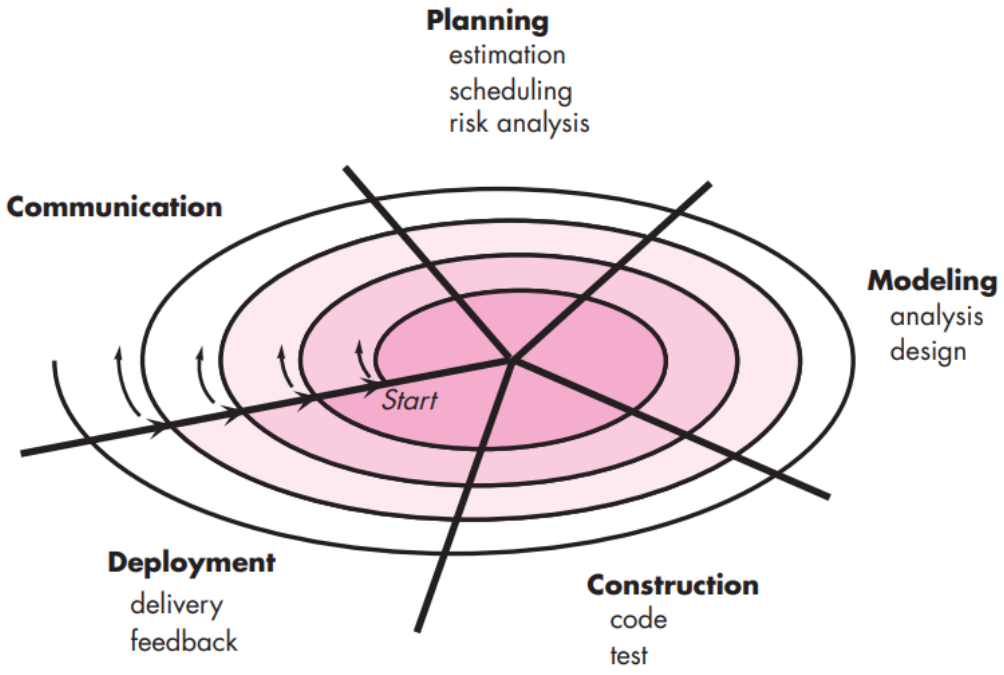
- The prototype is often discarded (at least in part), and the actual software is engineered with an eye toward quality.

✓ Prototyping should be used when the requirements are not clearly understood or are unstable
✓ Prototyping can also be used for developing UI or complex, high-tech systems in order to quickly demo the feasibility



THE SPIRAL MODEL

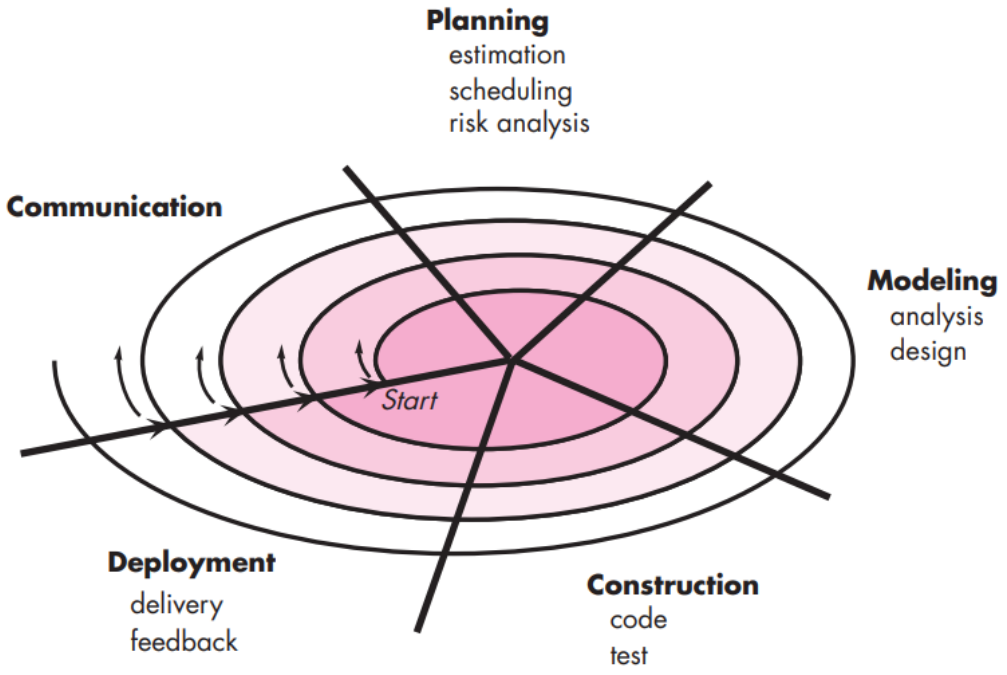
- The spiral model couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall mode
- Using the spiral model, software is developed in a series of evolutionary releases.
- During early iterations, the release might be a model or prototype.
- During later iterations, increasingly more complete versions of the engineered system are produced

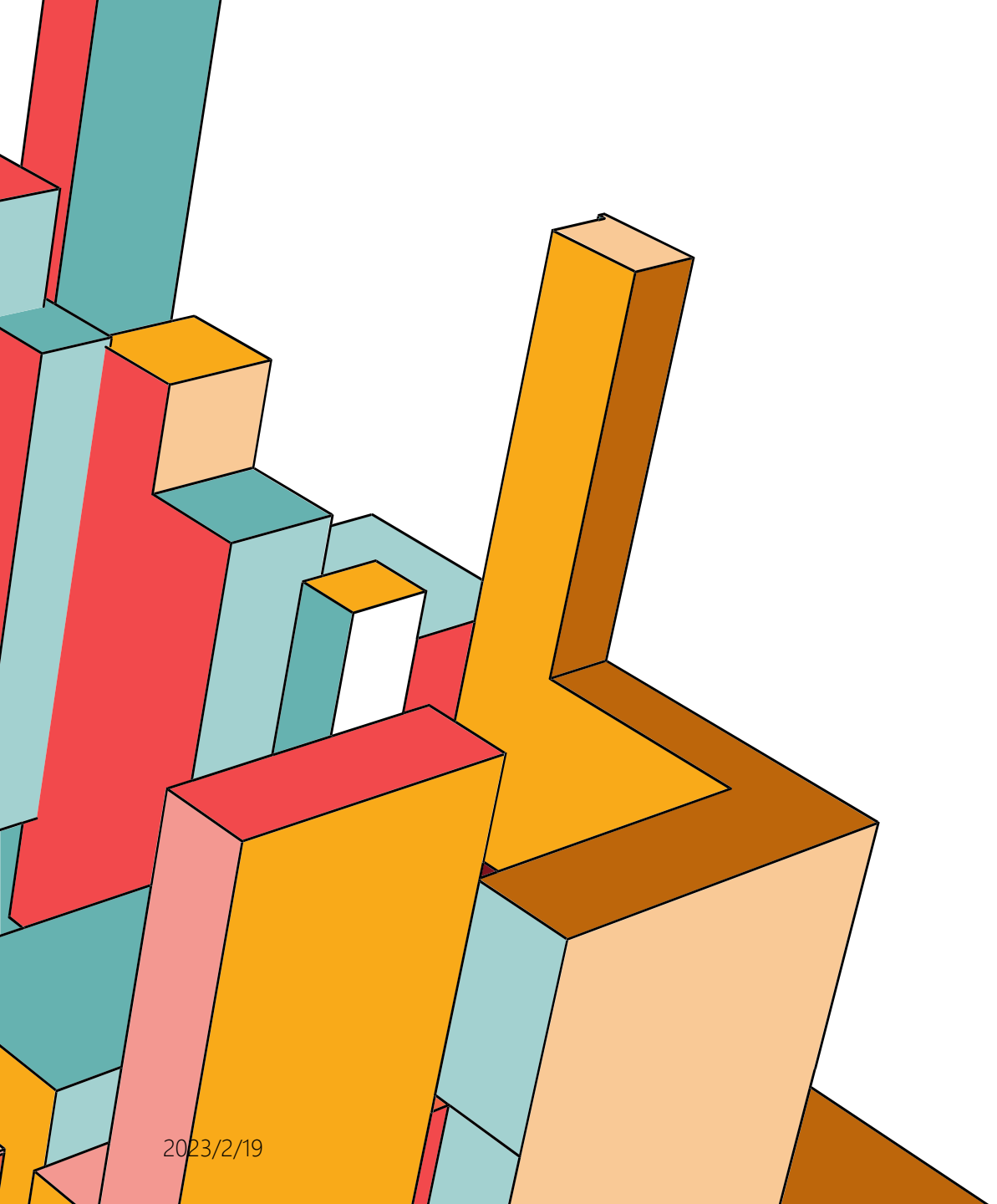


THE SPIRAL MODEL

- This model demands a direct consideration of technical risks at all stages of the project, effectively reducing potential risks

- Comparing the two:
  - Prototyping: requirements are not clear; focusing on continuously communicating with the users
  - Spiral: requirements are clear but adaptive; focusing on risk management





ARGUMENTS ON PROCESS MODELS

Process models forget the frailties of the people who build computer software

Software engineers are not robots. They exhibit great variation in working styles; significant differences in skill level, creativity, orderliness, consistency, and spontaneity. Some communicate well in written form, others do not

Process models often deal with people's common weaknesses with discipline

"Because consistency in action is a human weakness, high discipline methodologies are fragile." [Alistair Cockburn. 2002. Agile Software Development]

FRAGILE

AGILE ORIGIN

In 2001, Kent Beck and 16 other noted software developers, writers, and consultants signed the “Manifesto for Agile Software Development.” (敏捷软件开发宣言)

WHAT IS AGILE, EXACTLY?

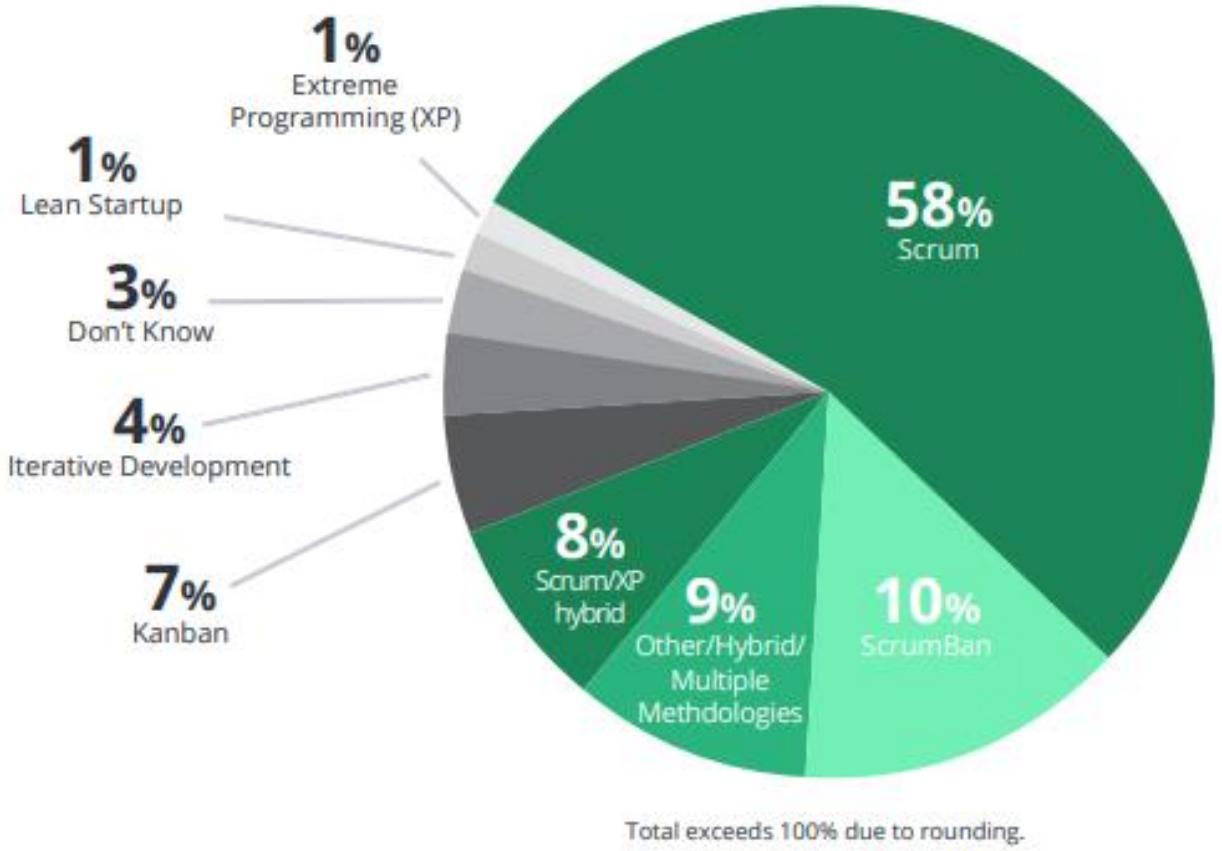
- Agile, in general, is the ability to create and respond to change.

- Agile, in the context of software engineering, refers to the methods and best practices for organizing projects based on the values and principles documented in the Agile Manifesto.

WHAT IS AGILE, EXACTLY?

- Agile is an iterative approach to project management and software development that helps teams deliver value to their customers faster and with fewer headaches
- An agile team delivers work in small, but consumable, increments. Requirements, plans, and results are evaluated continuously so teams have a natural mechanism for responding to change quickly.

AGILE METHODOLOGY



Source: 14th Annual State of Agile Report

- However, there is NO one right way to implement Agile
- There are many different types of agile methodologies

EXTREME PROGRAMMING

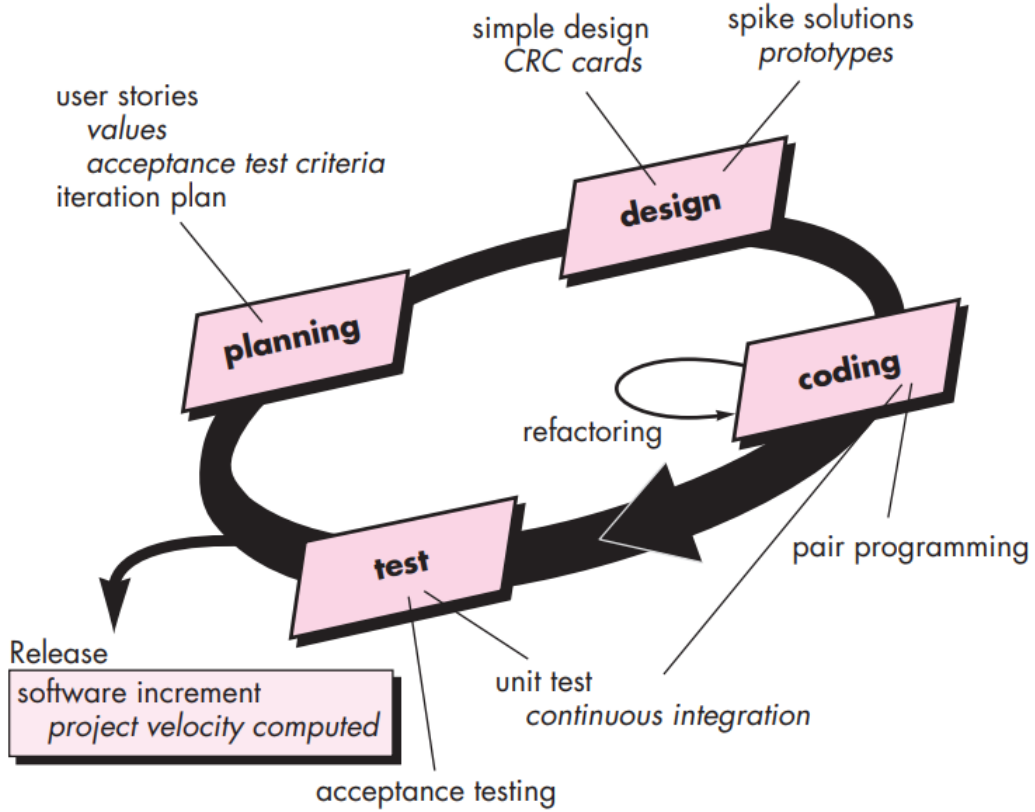
- Extreme programming (XP) is probably the first well-known agile process, created by Beck and Cunningham

- As a type of agile software development, XP advocates frequent releases in short development cycles, intended to improve productivity and introduce checkpoints at which new customer requirements can be adopt.

- XP uses an object-oriented approach as its preferred development paradigm

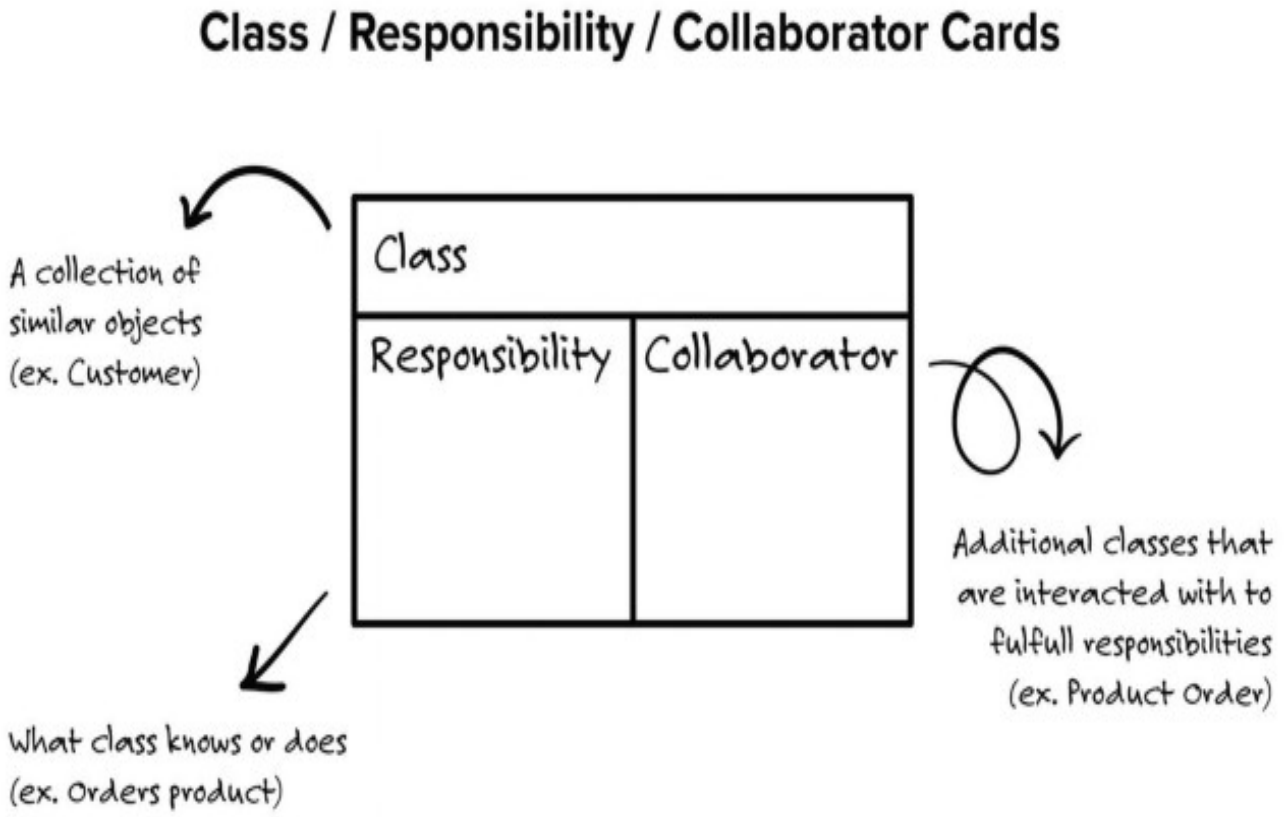
XP PROCESS - PLANNING

- User stories, written by customers, describe output, features, and functionality for software to be built
- Customers assign 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



XP PROCESS - DESIGN

- XP design following the KIS (keep it simple) principle
- XP encourages the use of CRC cards as an effective mechanism for thinking about the software in an object-oriented context.
- CRC (class-responsibility-collaborator) cards identify and organize the object-oriented classes that are relevant to the current software increment

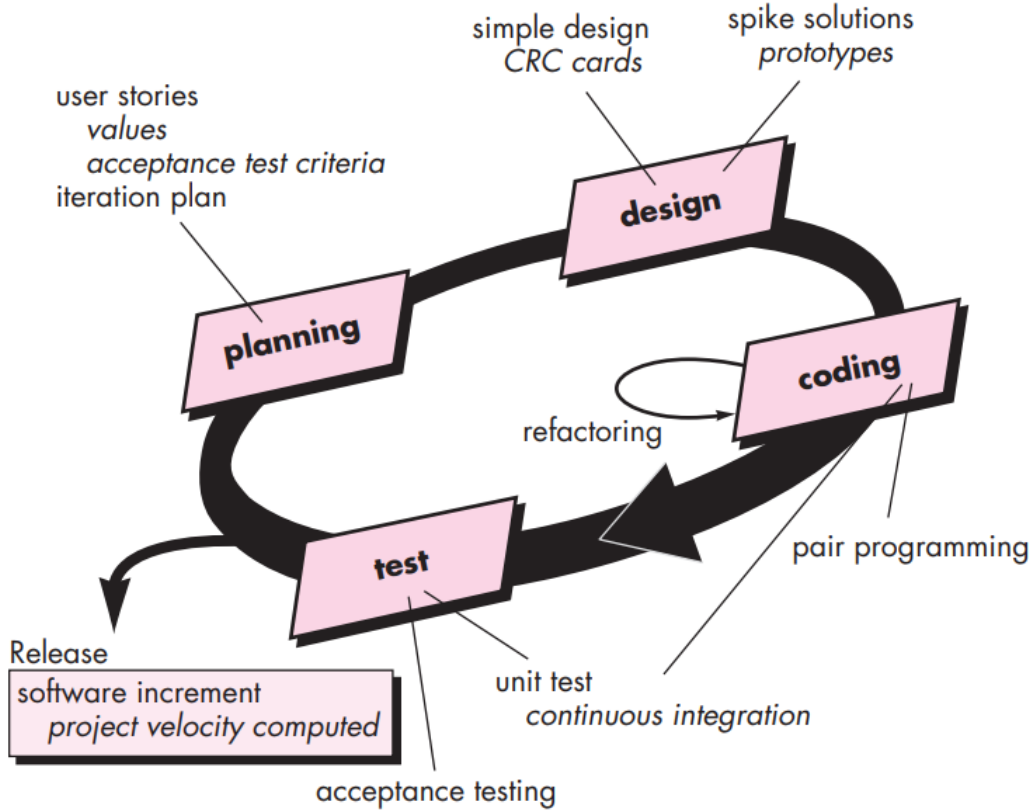


XP PROCESS - CODING

- A key concept during coding is pair programming
- Pair programming: two people work together at one computer workstation to create code for a story
- Code Review: Driver & navigator
- Refactoring: the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves the internal structure, making code more maintainable and less error-prone

XP PROCESS - TESTING

- Test-Driven Development (TDD)
  - Test first and make them automated
- Regression testing whenever code is modified
- After the first project release (also called a software increment) has been delivered, the XP team computes project velocity, which is the number of customer stories implemented during the release



THE ORIGIN OF SCRUM

- The term “scrum” was borrowed from the game of rugby, to stress the importance of teamwork to deal with complex problems
- Process of sport games = Process of software development

SCRUM ROLES

Product Owner

Define and adjust product features; Accept or reject work results

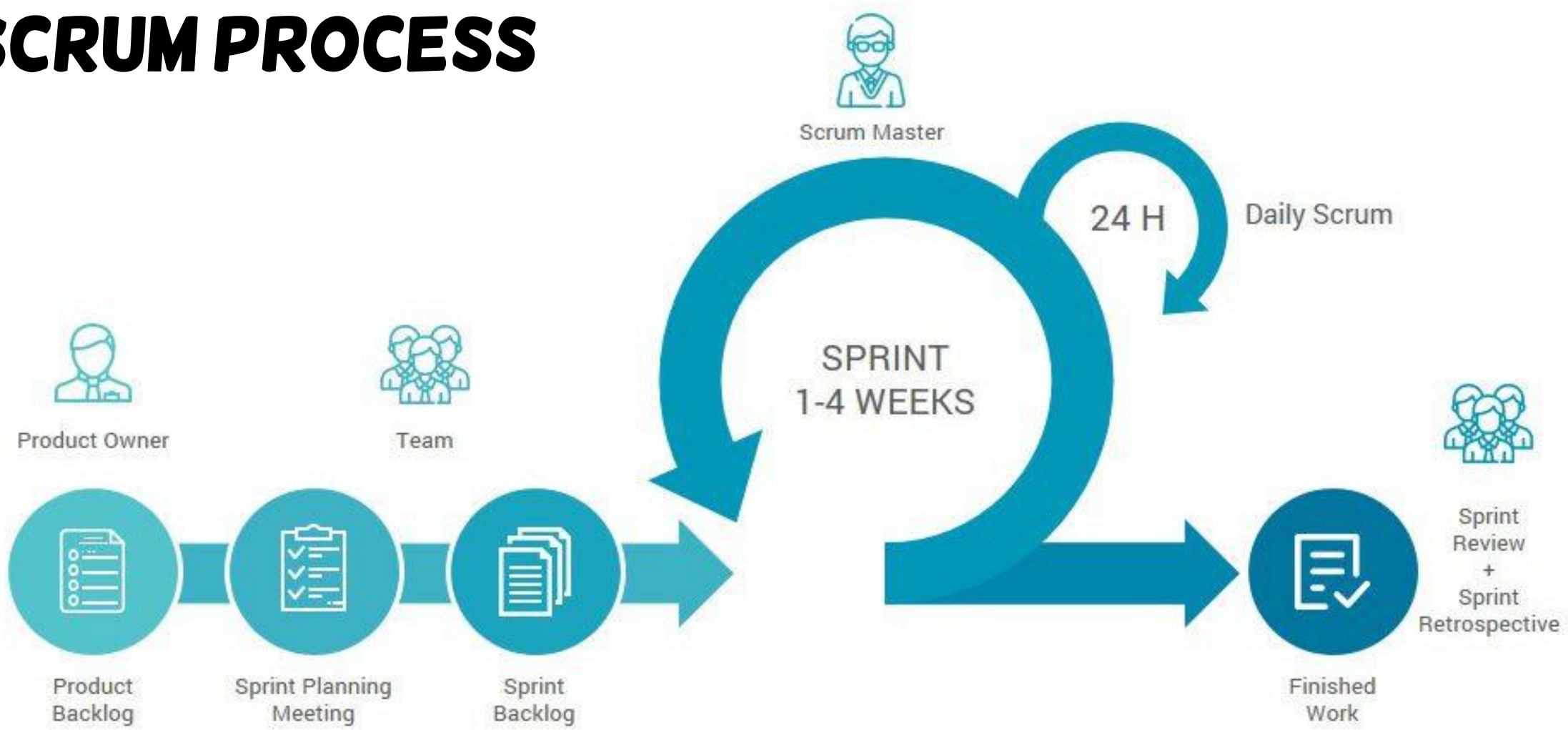
Scrum Master

Coach the team and enact Scrum values and principles

Scrum Team

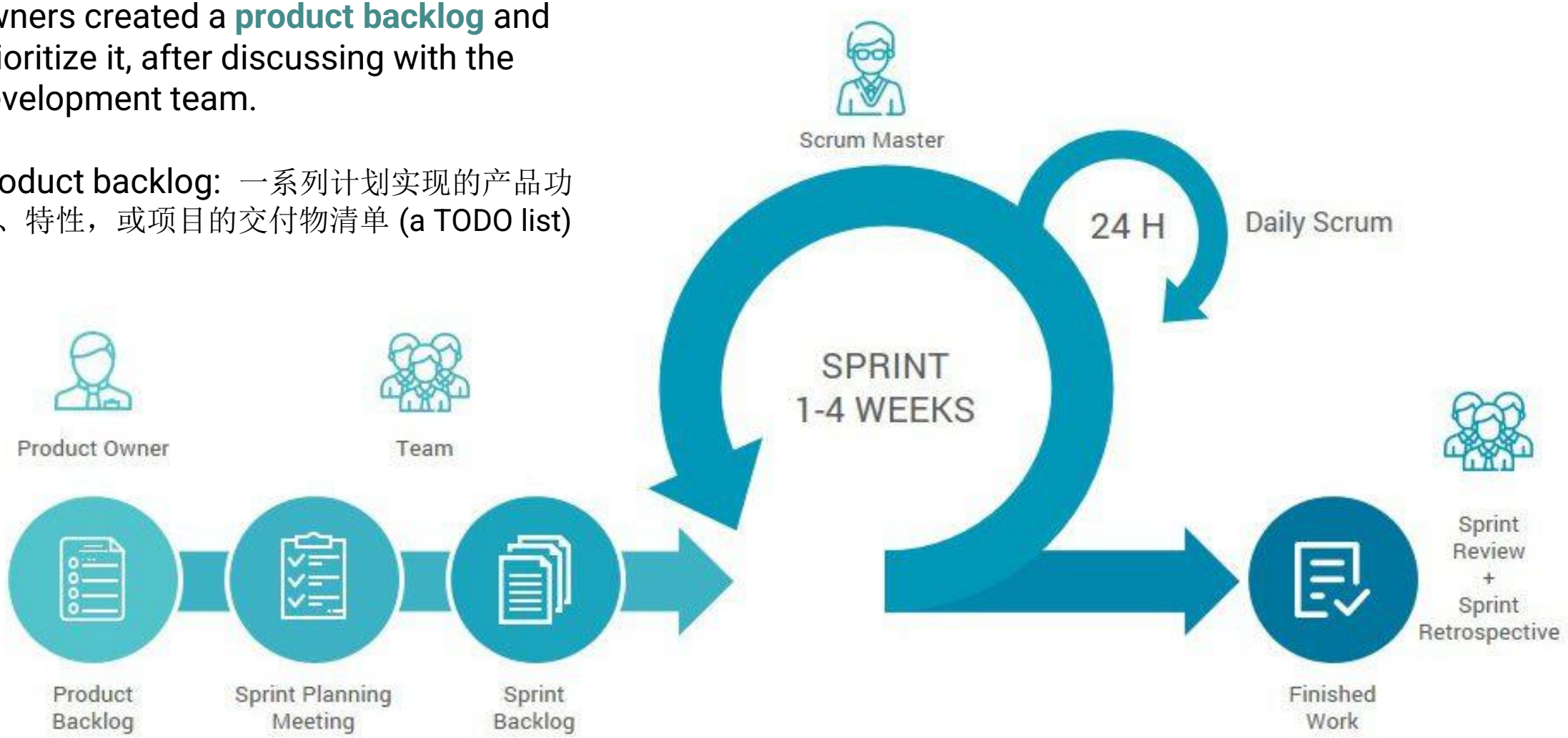
Cross-functional members like developers, testers, designers (Typically 5-9 size)

SCRUM PROCESS



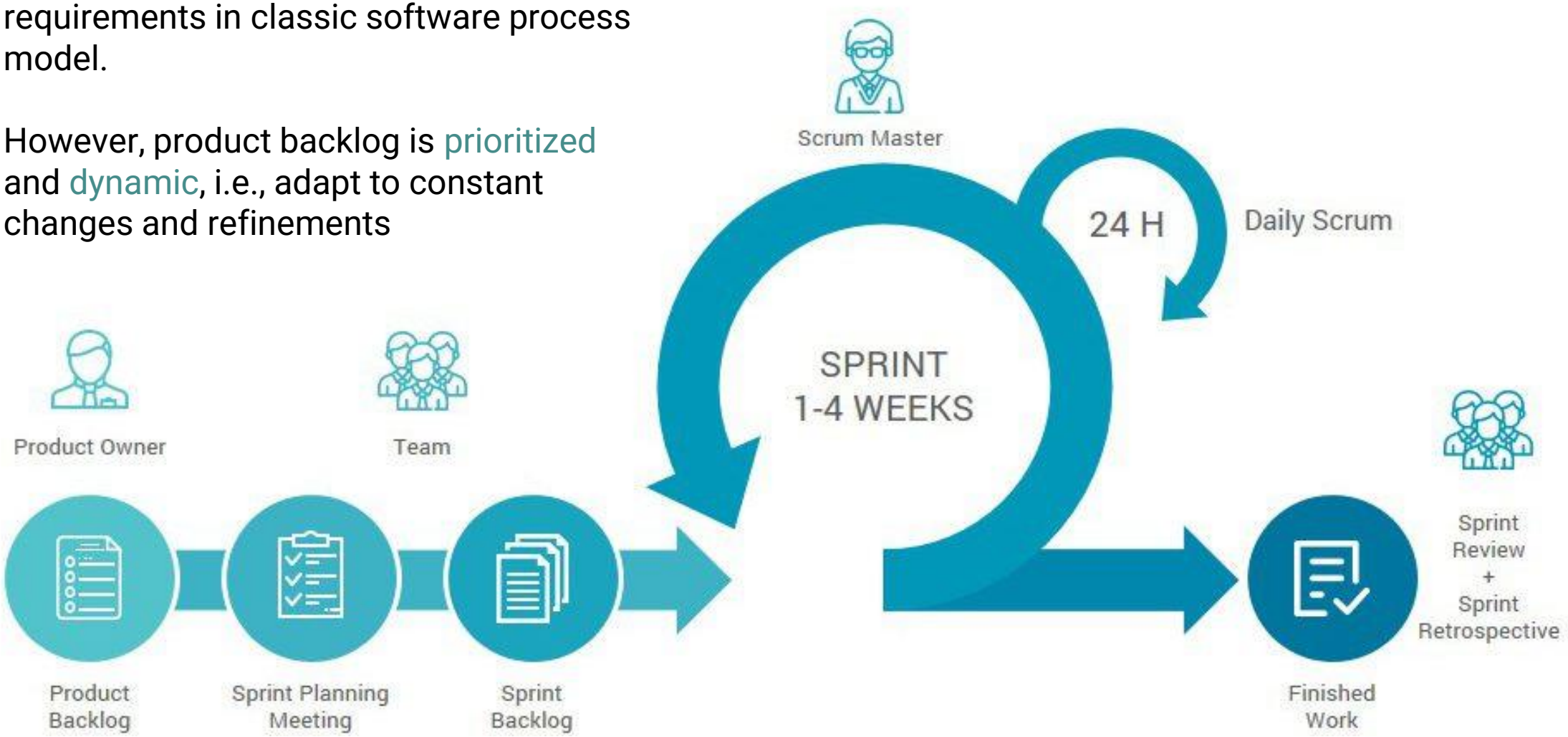
The scrum process starts with product owners created a **product backlog** and prioritize it, after discussing with the development team.

Product backlog: 一系列计划实现的产品功能、特性，或项目的交付物清单 (a TODO list)



Product backlog is similar to requirements in classic software process model.

However, product backlog is prioritized and dynamic, i.e., adapt to constant changes and refinements



ToDo List

ID	Story	Estimation	Priority
7	As an unauthorized User I want to create a new account	3	1
1	As an unauthorized User I want to login	1	2
10	As an authorized User I want to logout	1	3
9	Create script to purge database	1	4
2	As an authorized User I want to see the list of items so that I can select one	2	5
4	As an authorized User I want to add a new item so that it appears in the list	5	6
3	As an authorized User I want to delete the selected item	2	7
5	As an authorized User I want to edit the selected item	5	8
6	As an authorized User I want to set a reminder for a selected item so that I am reminded when item is due	8	9
8	As an administrator I want to see the list of accounts on login	2	10
Total		**30**	

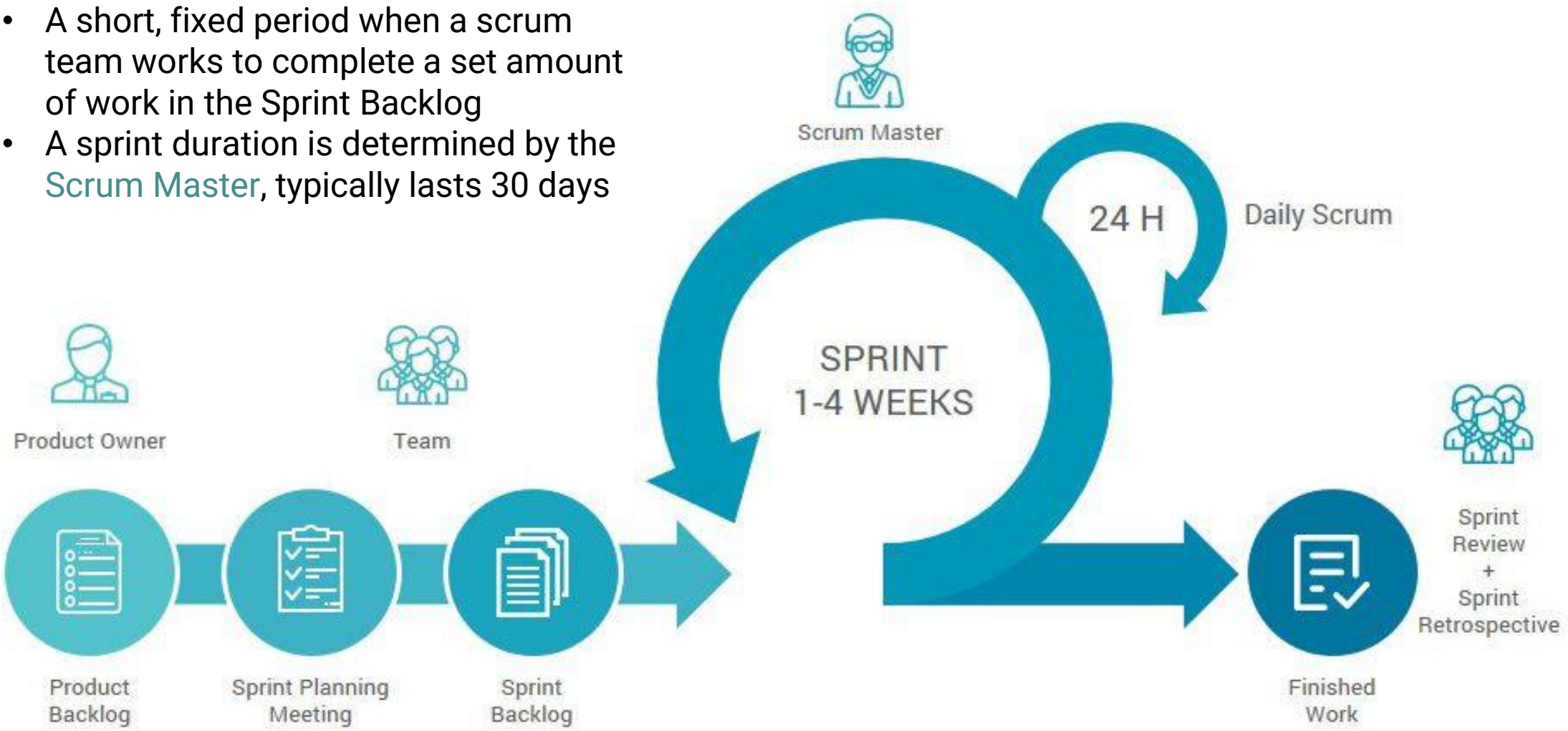
Next, the product owner and the development team attend the Sprint Planning Meeting to select the items to deliver in the next sprint.

Sprint (冲刺):Scrum项目管理方法中的一个常规、可重复的较短工作周期。在这个周期里，项目团队需快速完成预定的工作量（i.e., sprint backlog）

https://www.youtube.com/watch?v=pxxmSLJj8FQ&t=325s

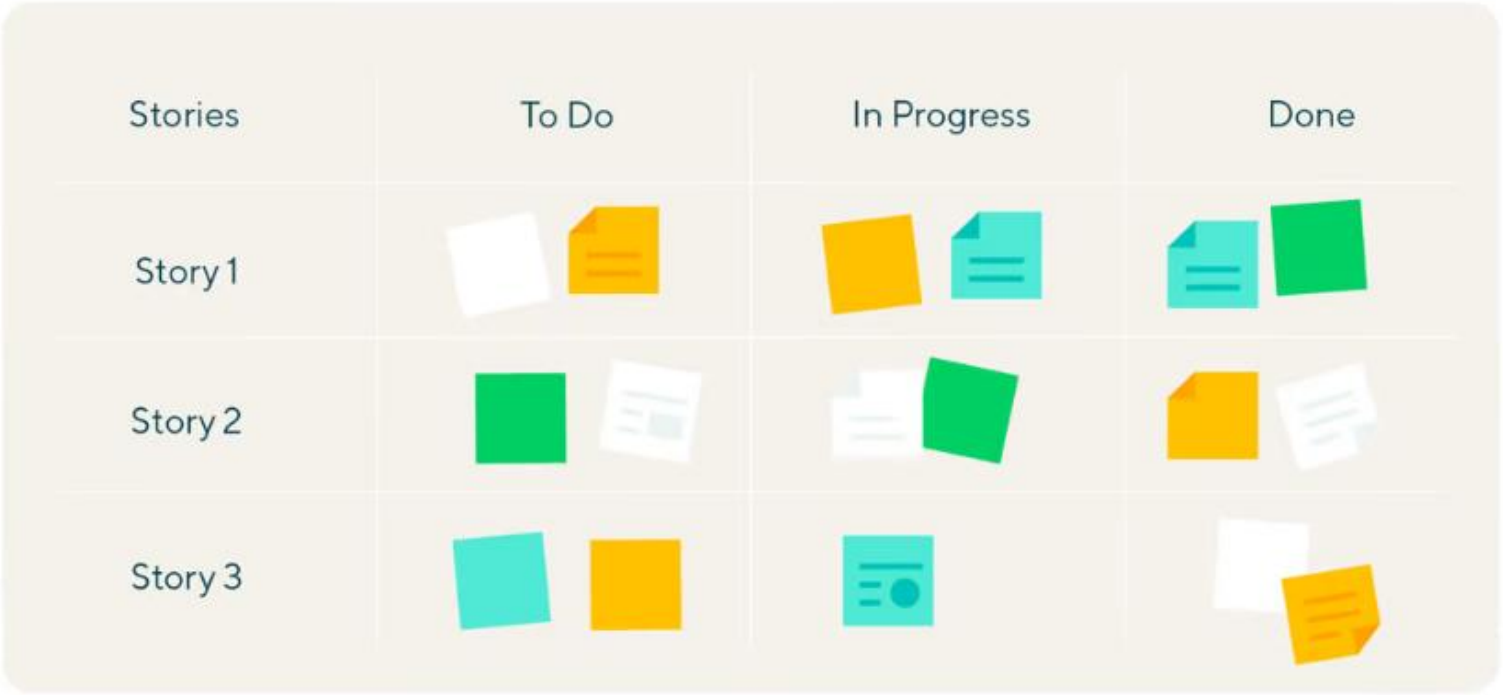
Sprint

- A short, fixed period when a scrum team works to complete a set amount of work in the Sprint Backlog
- A sprint duration is determined by the Scrum Master, typically lasts 30 days



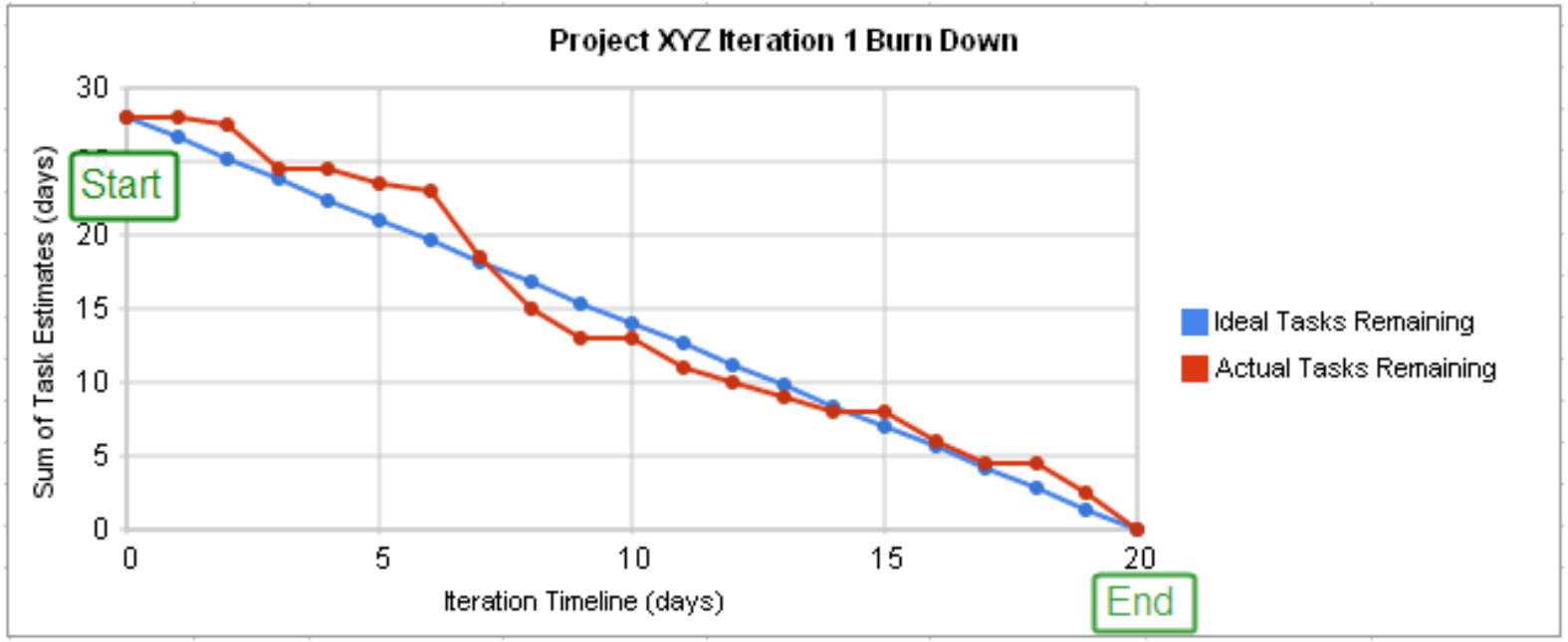
SCRUM BOARDS

- A Scrum Board is a tool that helps Scrum Teams make Sprint Backlog items visible.
- The board is traditionally divided up into three categories: To Do, Work in Progress and Done.



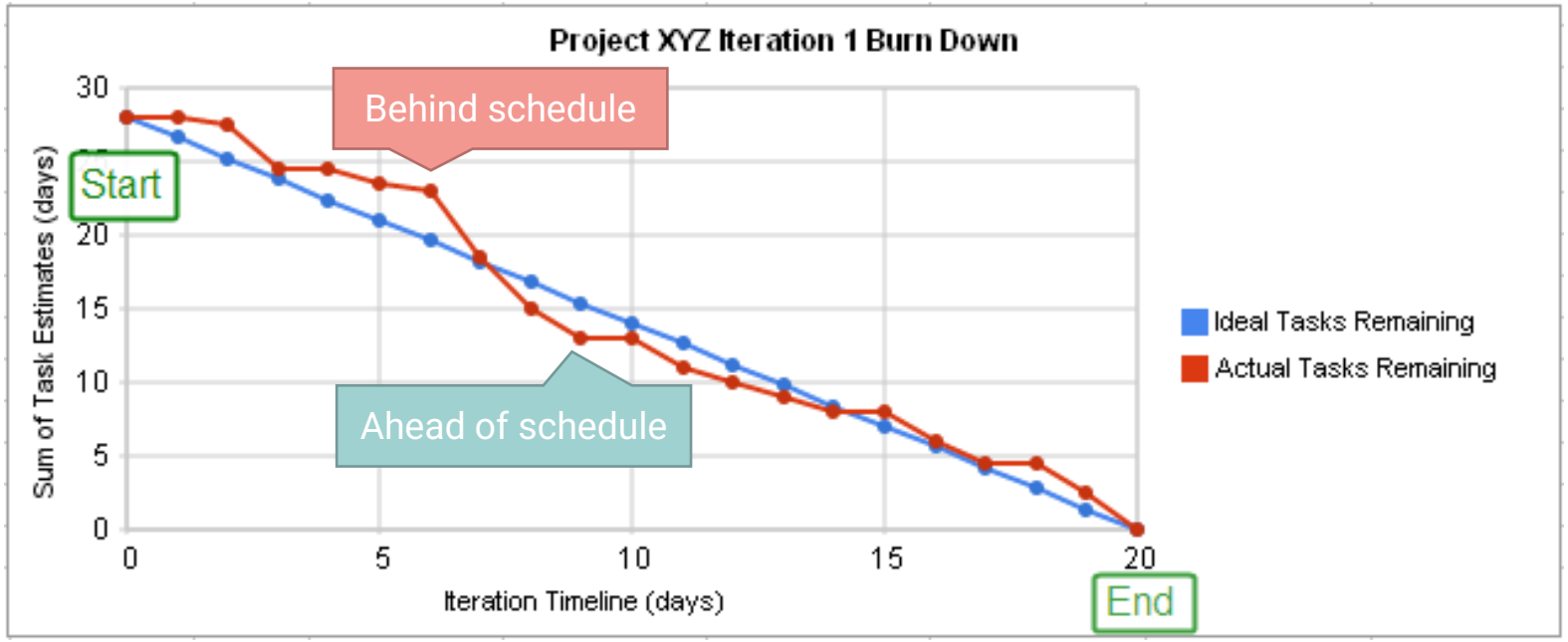
BURNDOWN CHARTS

- A graphical representation of work left to do versus time.
- The backlog or effort remaining is often on the Y-axis, with time in the sprint along the X-axis
- Useful for predicting when all of the work will be completed



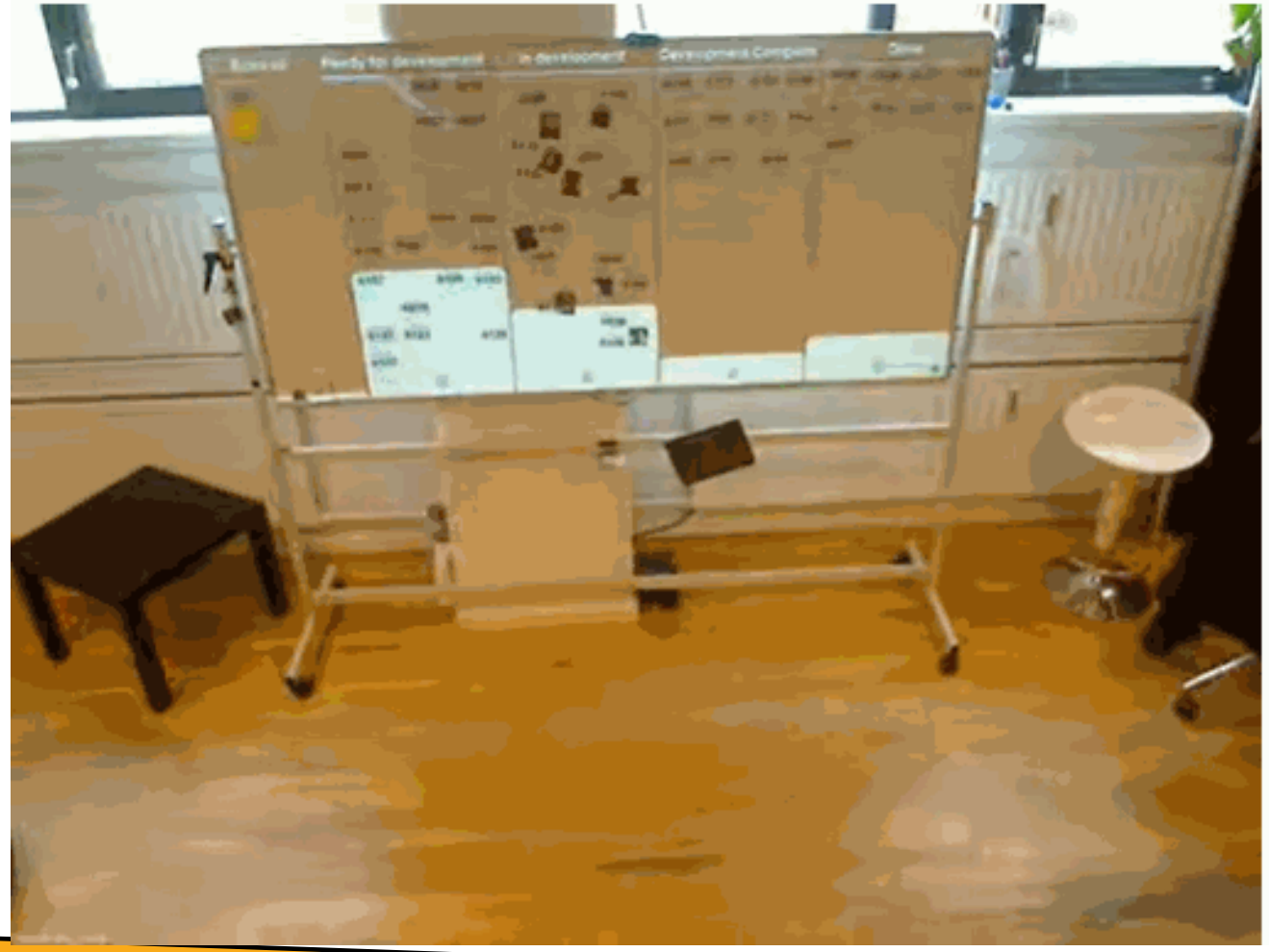
BURNDOWN CHARTS

- The top left corner is your starting point, and the successful project end is the bottom right.
- A straight line from start to finish represents your ideal work remaining
- A second line represents the actual work remaining



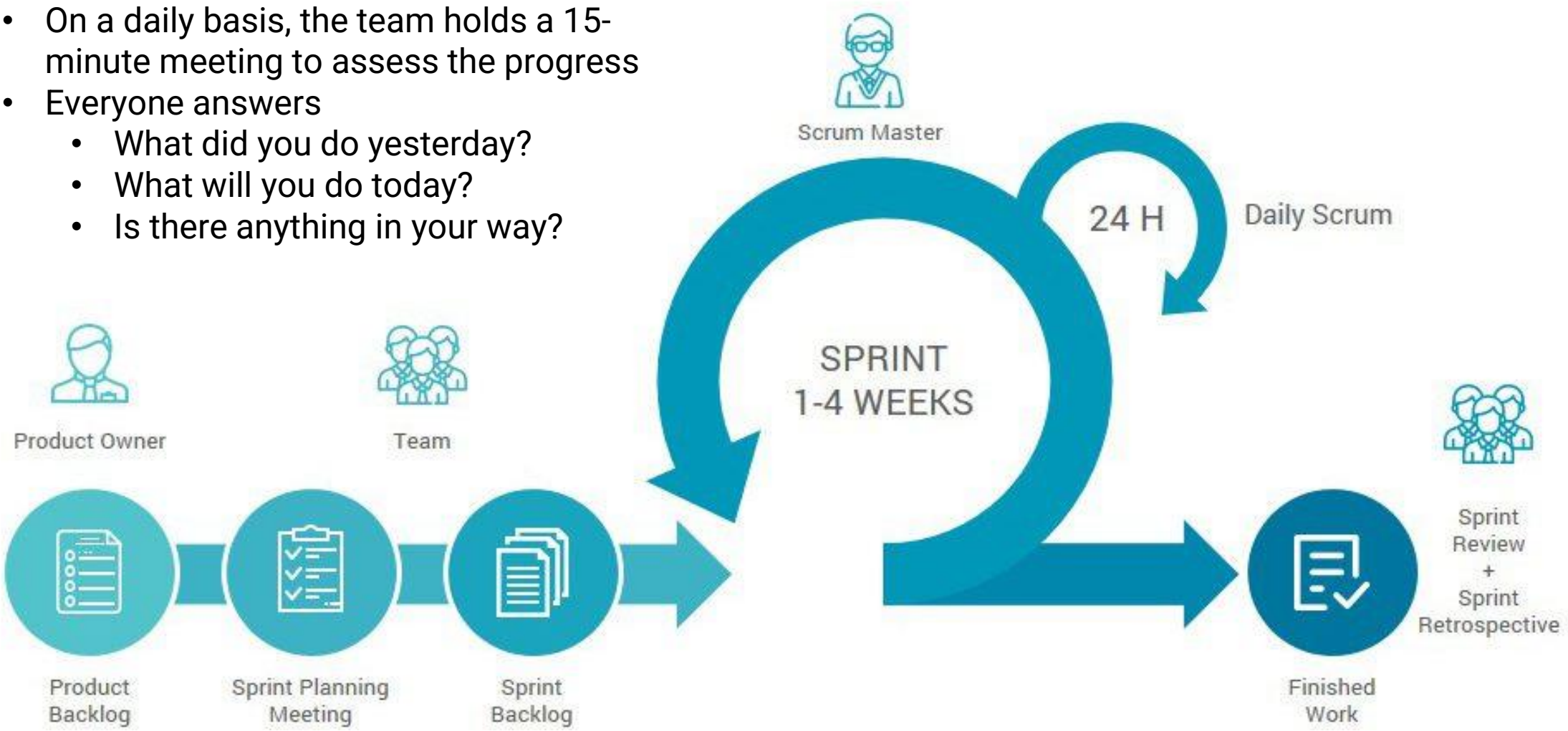
SCRUM BOARD AND BURNDOWN CHARTS IN ACTION

https://www.scruminc.com/sprint-burndown-chart/



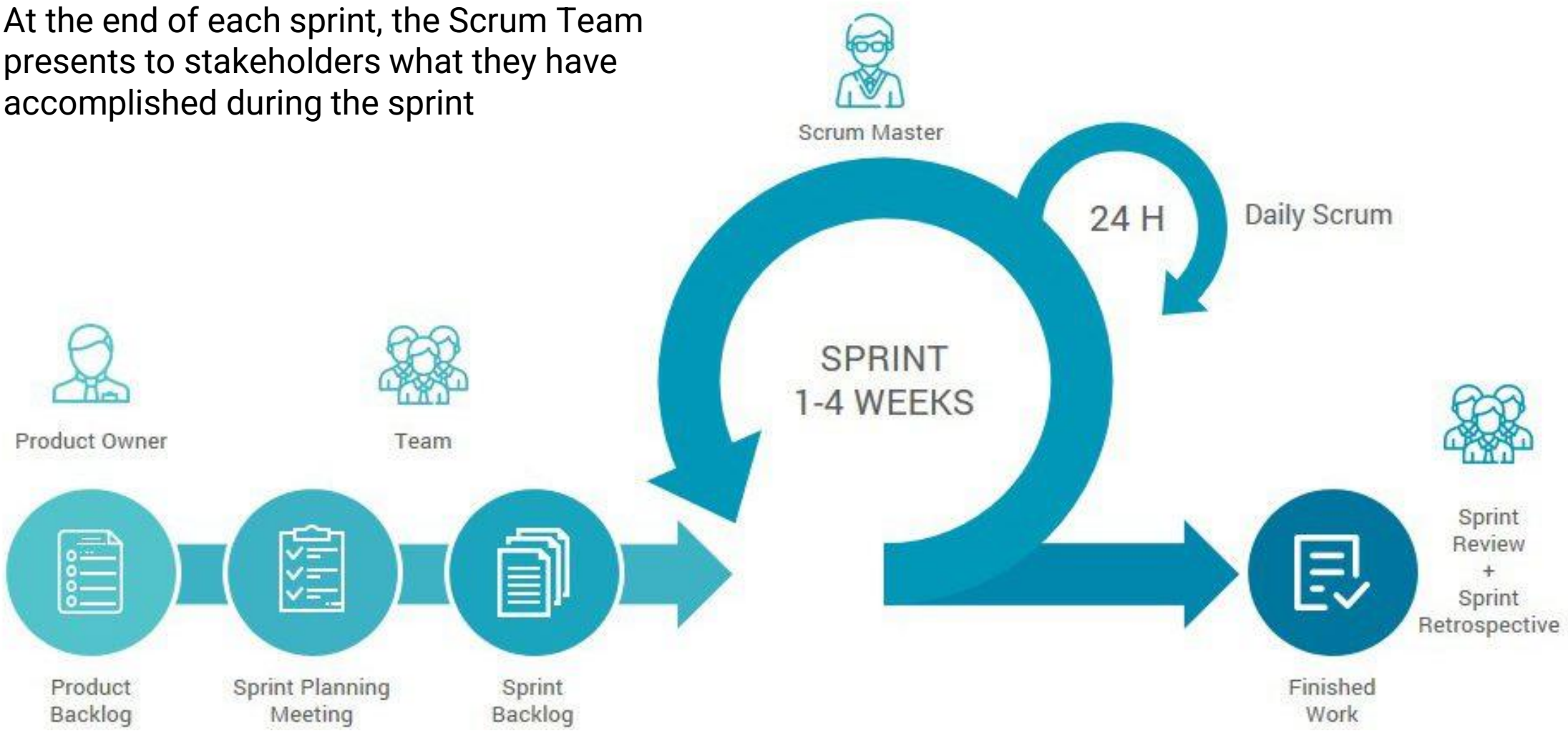
Daily Scrum (例会)

- On a daily basis, the team holds a 15-minute meeting to assess the progress
- Everyone answers
  - What did you do yesterday?
  - What will you do today?
  - Is there anything in your way?



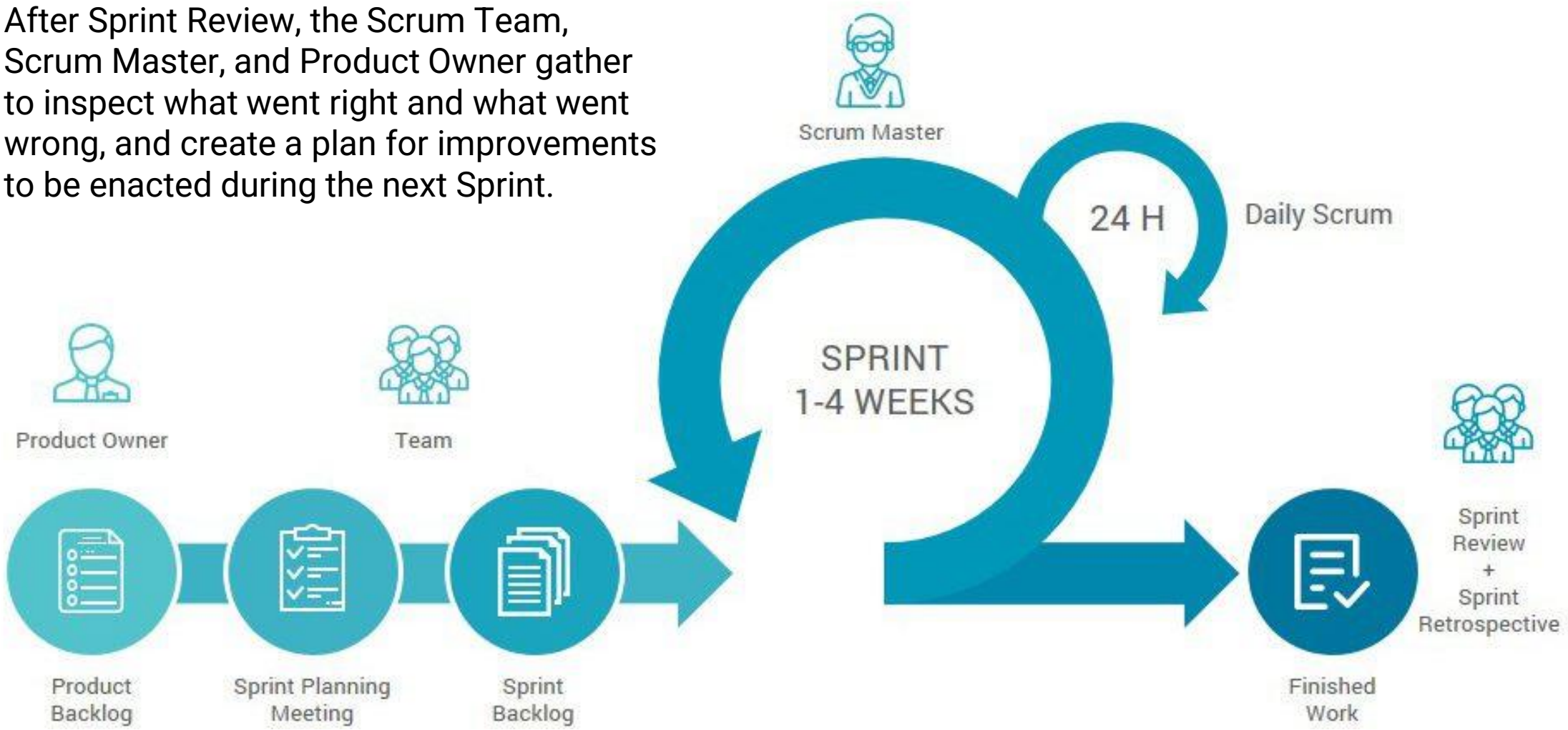
Sprint Review（评审）

At the end of each sprint, the Scrum Team presents to stakeholders what they have accomplished during the sprint



Sprint Retrospective（回顾）

After Sprint Review, the Scrum Team, Scrum Master, and Product Owner gather to inspect what went right and what went wrong, and create a plan for improvements to be enacted during the next Sprint.



SCRUM SUMMARY

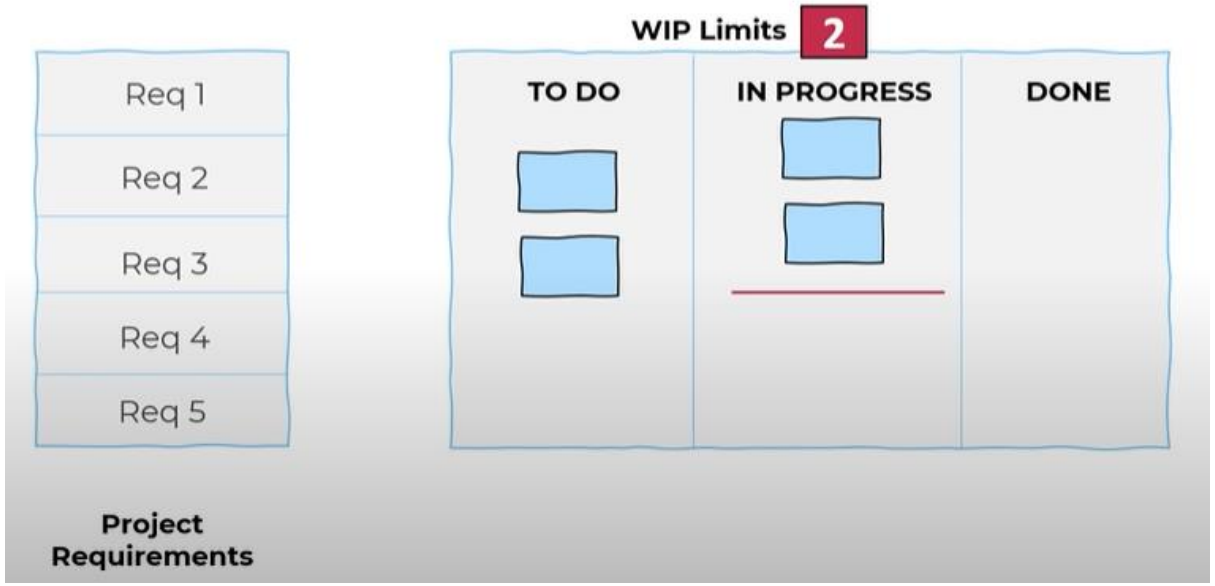
Roles	Ceremonies	Artifacts
Product Owner	Sprint Planning	Product Backlog
Scrum Master	Daily Scrum	Sprint Backlog
Scrum Team	Sprint Review	Burndown Charts
	Sprint Retrospective	

THE ORIGIN OF KANBAN

- Kanban originated in lean manufacturing, which was inspired by the Toyota Production System.
- It has its origin in the late 1940s when the Toyota automotive company implemented a production system called just-in-time; **which had the objective of producing according to customer demand and identifying possible material shortages within the production line.**
- Microsoft engineer David J. Anderson realized how this method devised by Toyota could become a process applicable to IT and software development process [Source: Wikipedia]

PRINCIPLES OF KANBAN

- Kanban focuses on maintaining a continuous task flow and continuous delivery
- Two primary principles of Kanban
  - **Visualization**: the work of the development team (features and user stories) are visualized
  - **Limited Work-in-progress** (WIP): the team is never given more work than it can handle



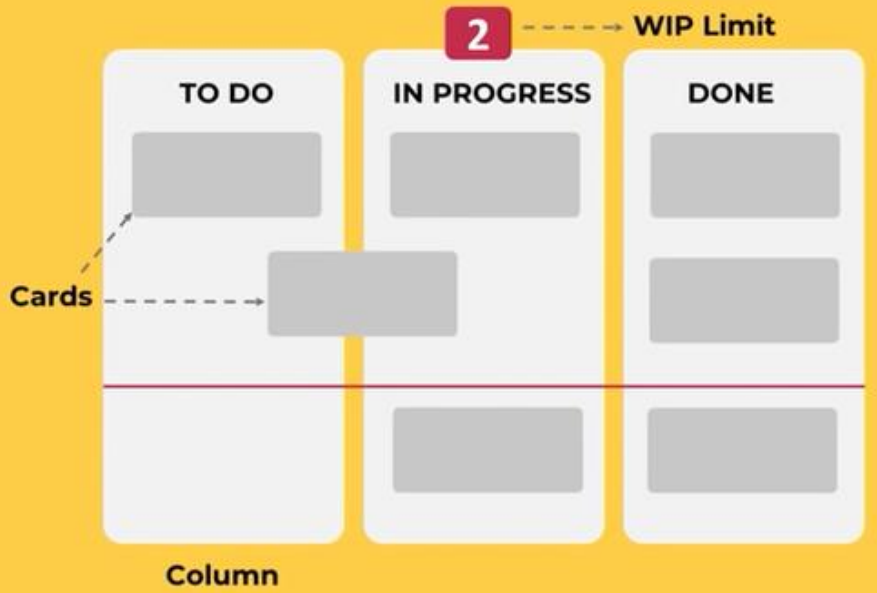
SCRUM VS. KANBAN

FLOW

Scrum

Scrum has ***sprints***. The Sprint is a timebox of one month or less during which the team produces a potentially shippable product increment.

Kanban

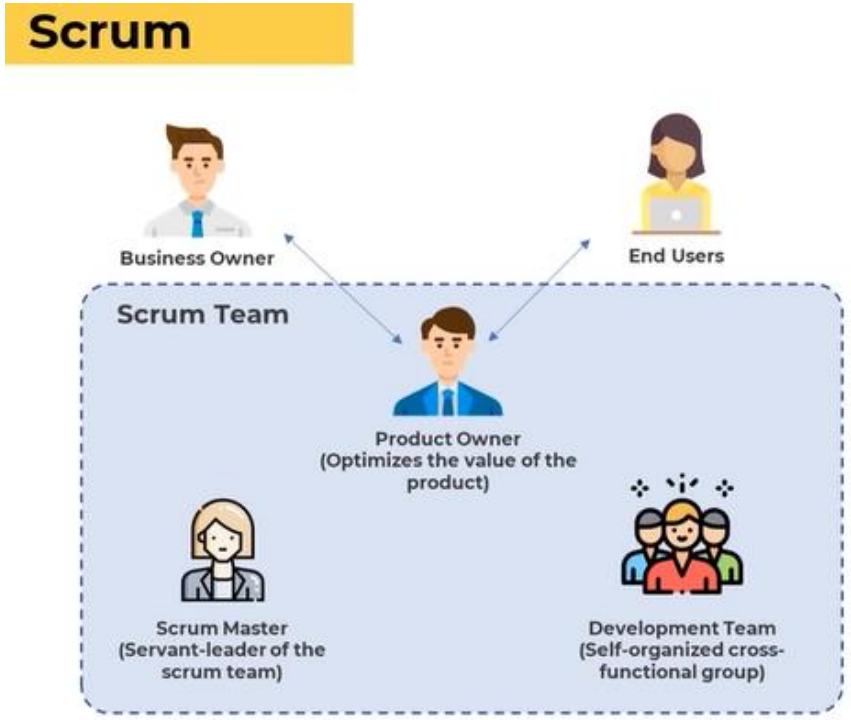


Kanban, there are ***no required time boxes or iterations***. It focuses on maintaining a continuous task flow and continuous delivery.

Source: Invensis Learning: https://youtu.be/pxxmSLJj8FQ

SCRUM VS. KANBAN

ROLES



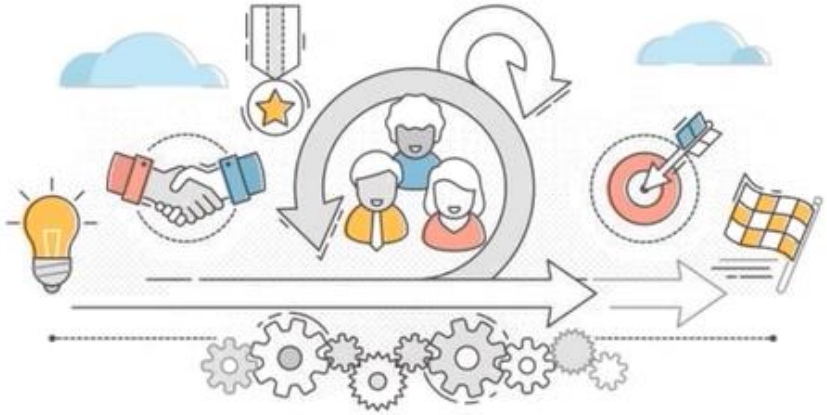
Scrum has a ***set of mandatory roles*** that you must implement. The Product Owner, Scrum Master, and Team Members.

Source: Invensis Learning: https://youtu.be/pxxmSLJj8FQ

SCRUM VS. KANBAN

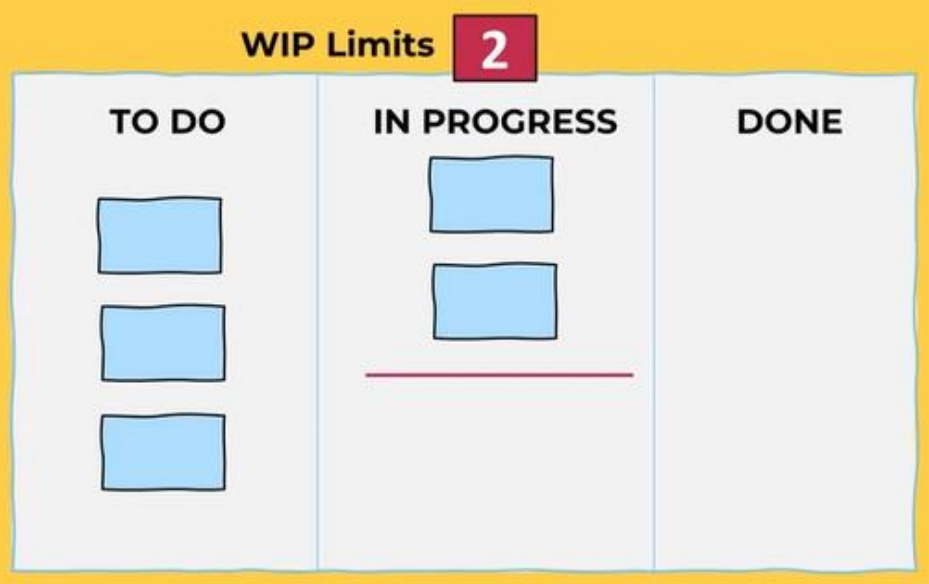
COMMITMENT

Scrum



The team ***commits to a specific amount of work for each iteration***. If the capacity is not measured accurately, sprint might fail.

Kanban



Commitment is based on ***capacity***. ***Work in progress (WIP) limits*** prevent team members from working on multiple tasks.

Source: Invensis Learning: https://youtu.be/pxxmSLJj8FQ

SCRUM VS. KANBAN

PLANNING

Scrum

Planning happens iteratively at the ***beginning of each Sprint.***

Kanban

JIT

Just-in-time Delivery

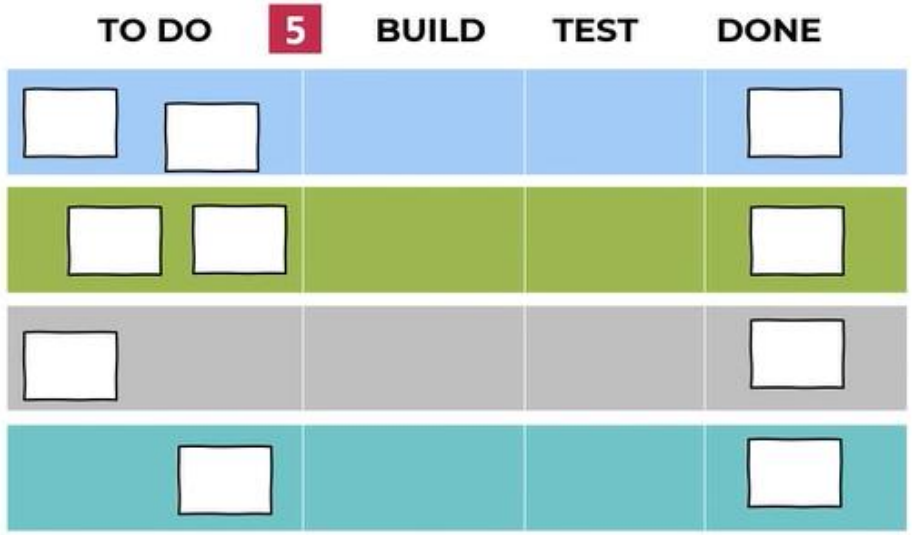
With Kanban, there's close to zero planning involved. It includes ***Just-in-time planning***, instead of planning for a bigger time period

Source: Invensis Learning: https://youtu.be/pxxmSLJj8FQ

SCRUM VS. KANBAN

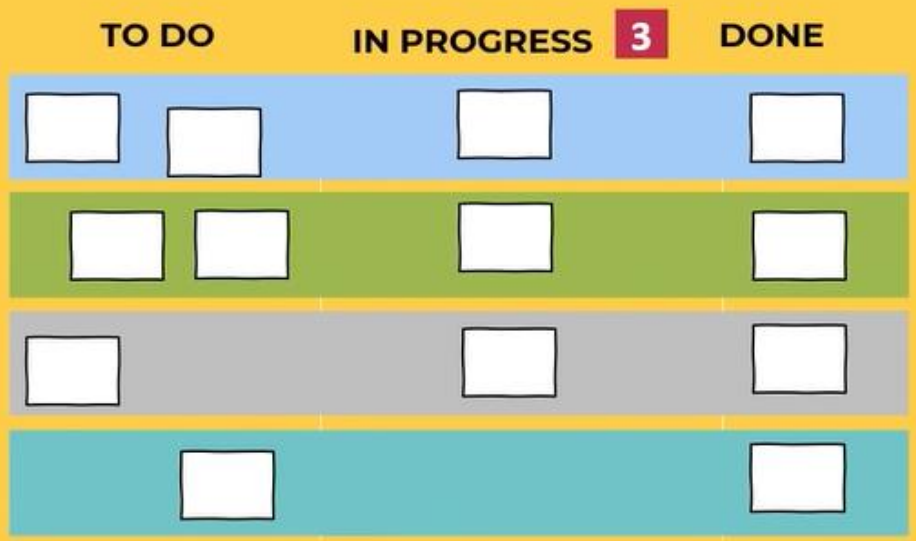
WORKLOAD

Scrum



Scrum ***limits work in progress per iteration***. The development team has to commit to the number of tasks that they are ready to accomplish during the Sprint.

Kanban



Limits work in progress by assigning a ***limit to the number of cards in any active-work columns***. When the limit is met, no new work can enter the column until a task is completed.

Source: Invensis Learning: https://youtu.be/pxxmSLJj8FQ

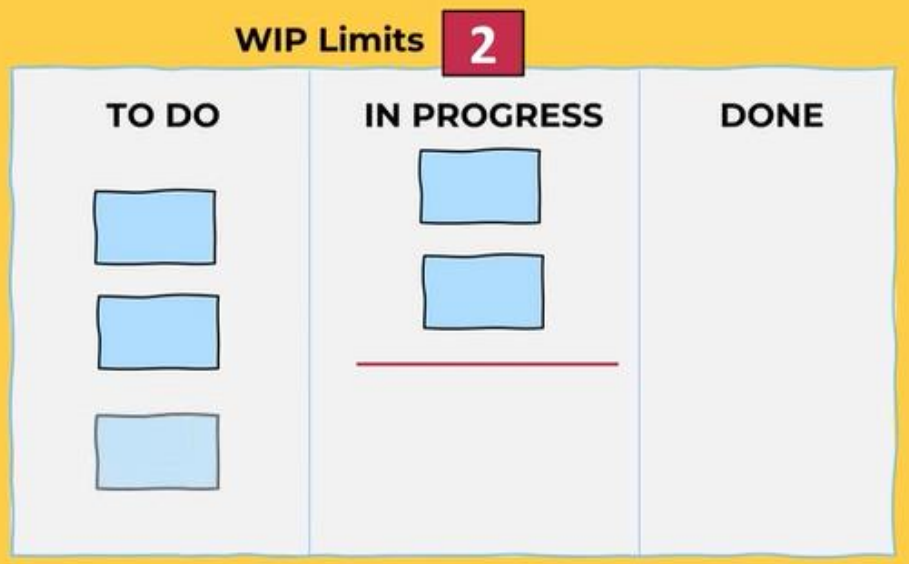
SCRUM VS. KANBAN

CHANGES

Scrum

Once the ***sprint begins, you aren't allowed to add any new requirements***. When more tasks than planned are added, ***scope creep*** occurs.

Kanban



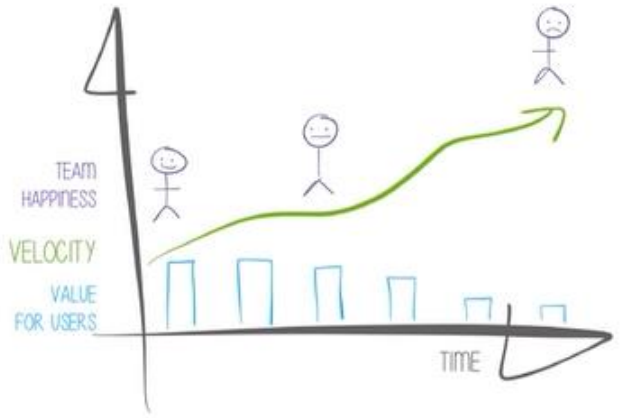
You can ***insert or add tasks constantly to the backlog*** and existing cards can get blocked or removed all together based on prioritization.

Source: Invensis Learning: https://youtu.be/pxxmSLJj8FQ

SCRUM VS. KANBAN

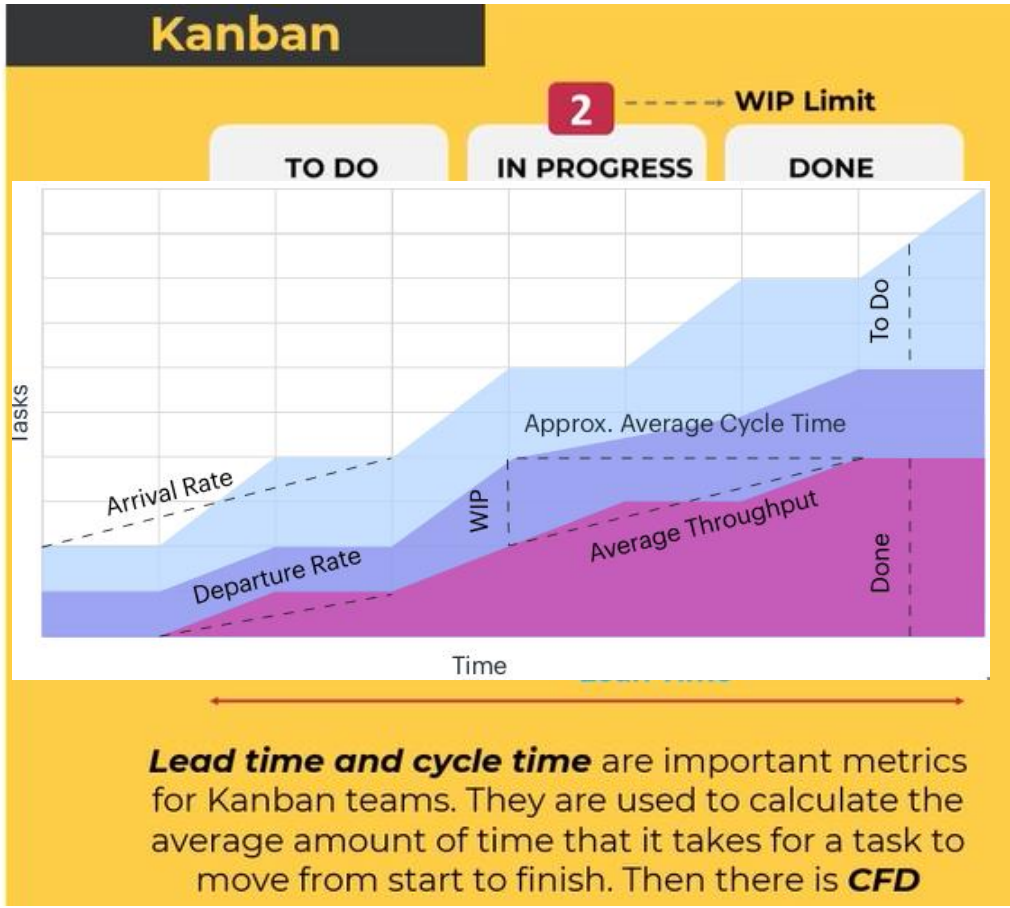
KPI

Scrum



Scrum measures productivity using a metric called ***velocity***, which is the number of story points completed in a sprint. Uses Burndown chart and Velocity chart.

Kanban

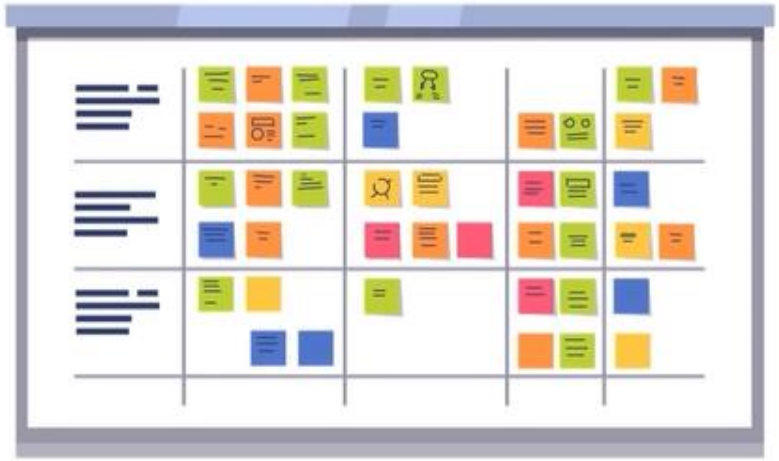


Lead time and cycle time are important metrics for Kanban teams. They are used to calculate the average amount of time that it takes for a task to move from start to finish. Then there is ***CFD***

Source: Invensis Learning: https://youtu.be/pxxmSLJj8FQ

SCRUM VS. KANBAN BOARD

Scrum



On a Scrum Board, the ***columns are labeled to show the workflow states.*** At the beginning of the sprint all the stories are added to the board

Kanban

Kanban board also has the columns labeled to show the workflow states. The difference is that Kanban ***board has WIP limits visualized on it***.

Source: Invensis Learning: https://youtu.be/pxxmSLJj8FQ

SCRUM VS. KANBAN

APPLICATION

Scrum

- Ideal for projects where you want to move fast but you need some degree of planning and coordination
- Goal-driven projects with fixed deliverables
- Suitable for non-mature teams

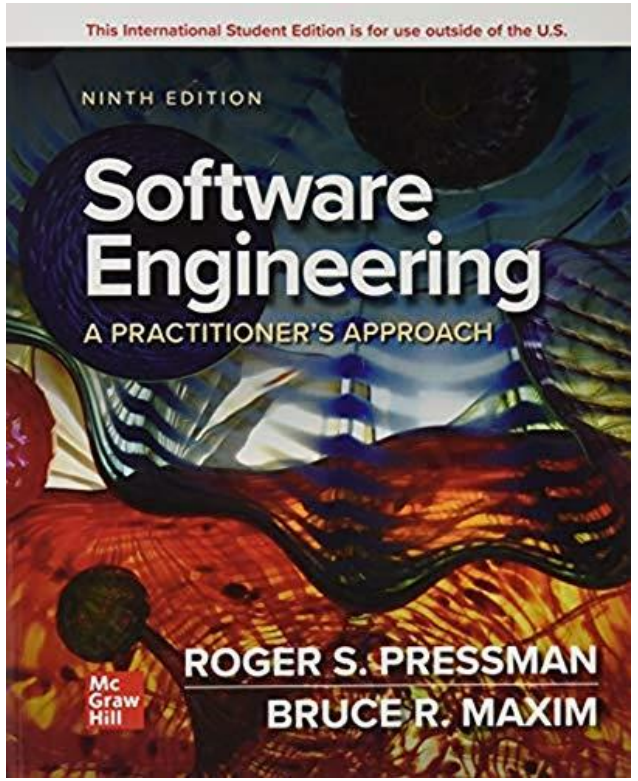
Kanban

- Kanban is the best choice when you have a lot of incoming tasks with changing priorities
- It is better suitable for mature teams
- Teams involved in industries such as marketing, software development, or content creation can benefit from Kanban

Source: Invensis Learning: https://youtu.be/pxxmSLJj8FQ

READING

- Chapter 2: Process Models
- Chapter 3: Agile Development

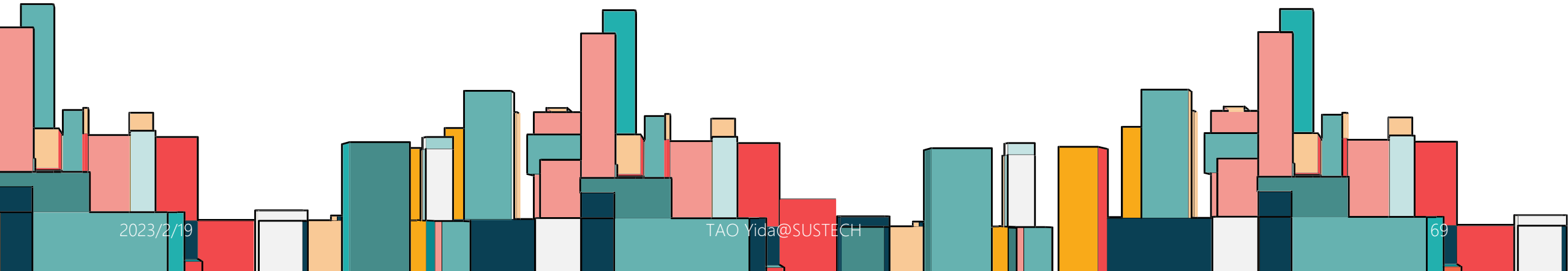


CHOOSE A PROCESS SUITABLE FOR YOUR TEAM!

Agile

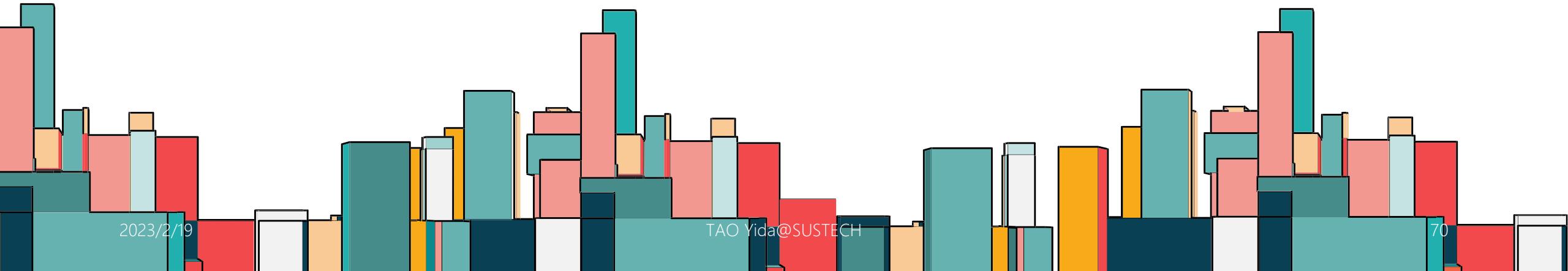
Plan-driven

Describe the type of process you choose and the corresponding plans & schedules in milestone1 team report!



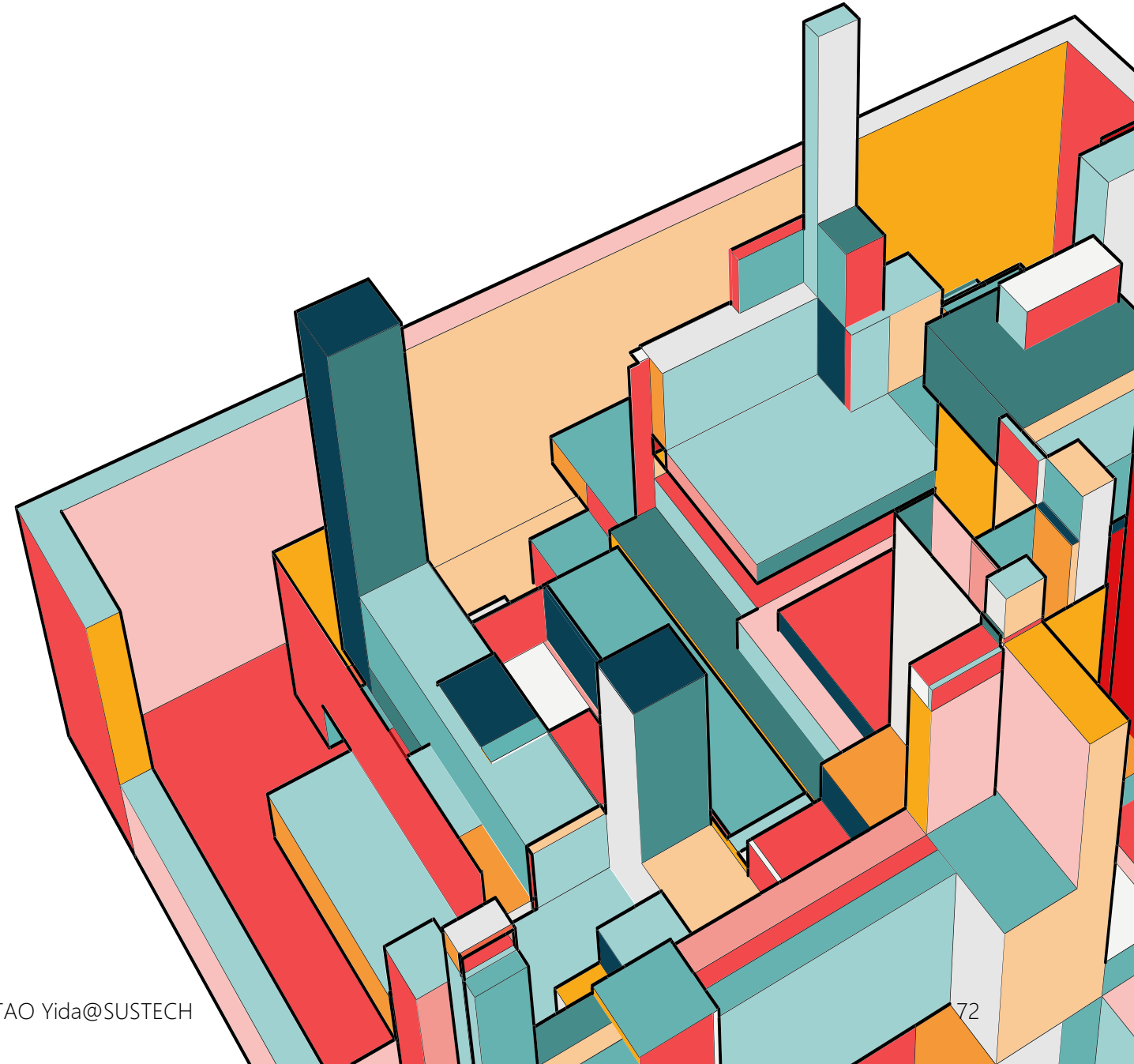
WHAT ABOUT THE OPERATIONS TEAM?

- Agile is typically used within development teams.
- What about the operations team (运营团队) that is responsible for deploying the software and managing the stability of the service and the infrastructure?



LECTURE 2

- Software Process Overview
- Process Models
- Agile Development
- DevOps



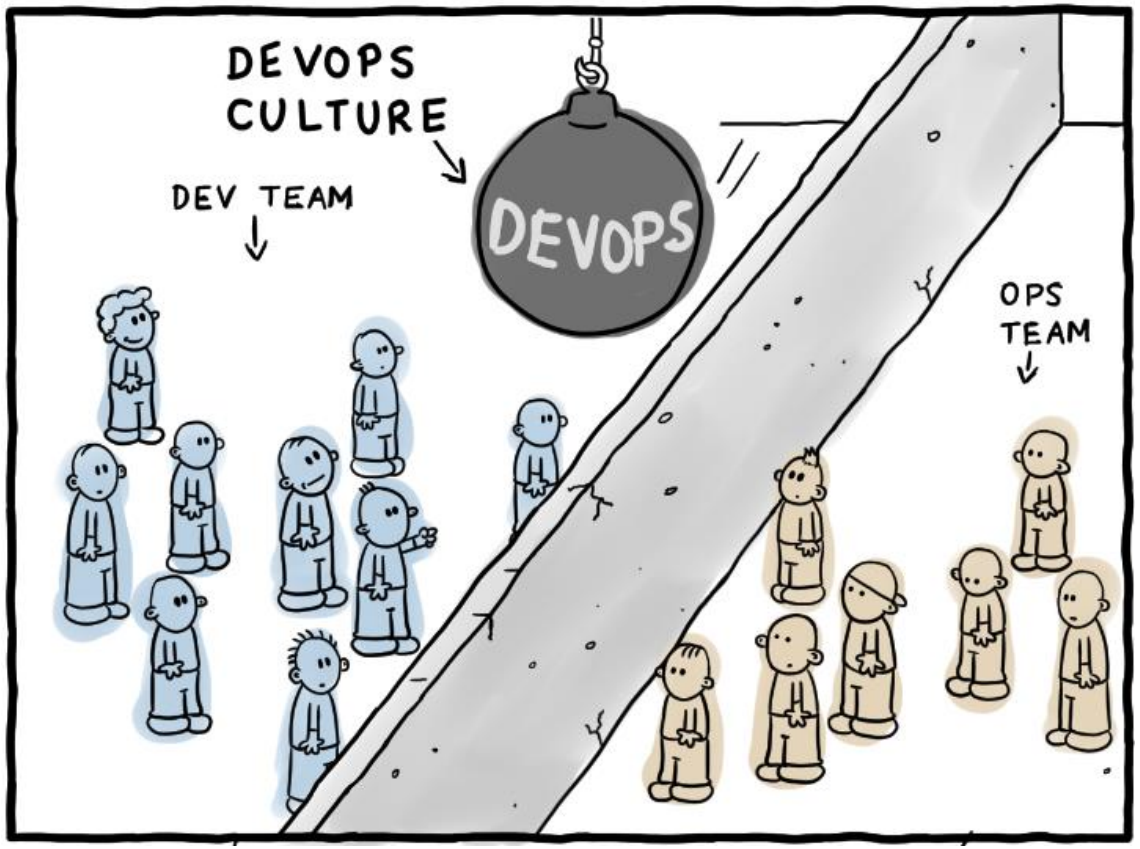
DEVOPS

- The word “DevOps” is a mashup of “development” and “operations”

- Before DevOps, in traditional IT operations, development teams and operations teams typically work sequentially and individually
  - **Development team** (开发团队): design, develop, document and test the software (sometimes with a QA team) in the development environment, then hand off the product to operations teams
  - **Operations teams** (运营团队): responsible for deploying the software in the production environment, managing the service’s stability and uptime, as well as the infrastructure that hosts the code, with little-to-no direct interaction with the development teams.

https://github.blog/2020-12-03-the-evolving-role-of-operations-in-devops/

BEFORE DEVOPS

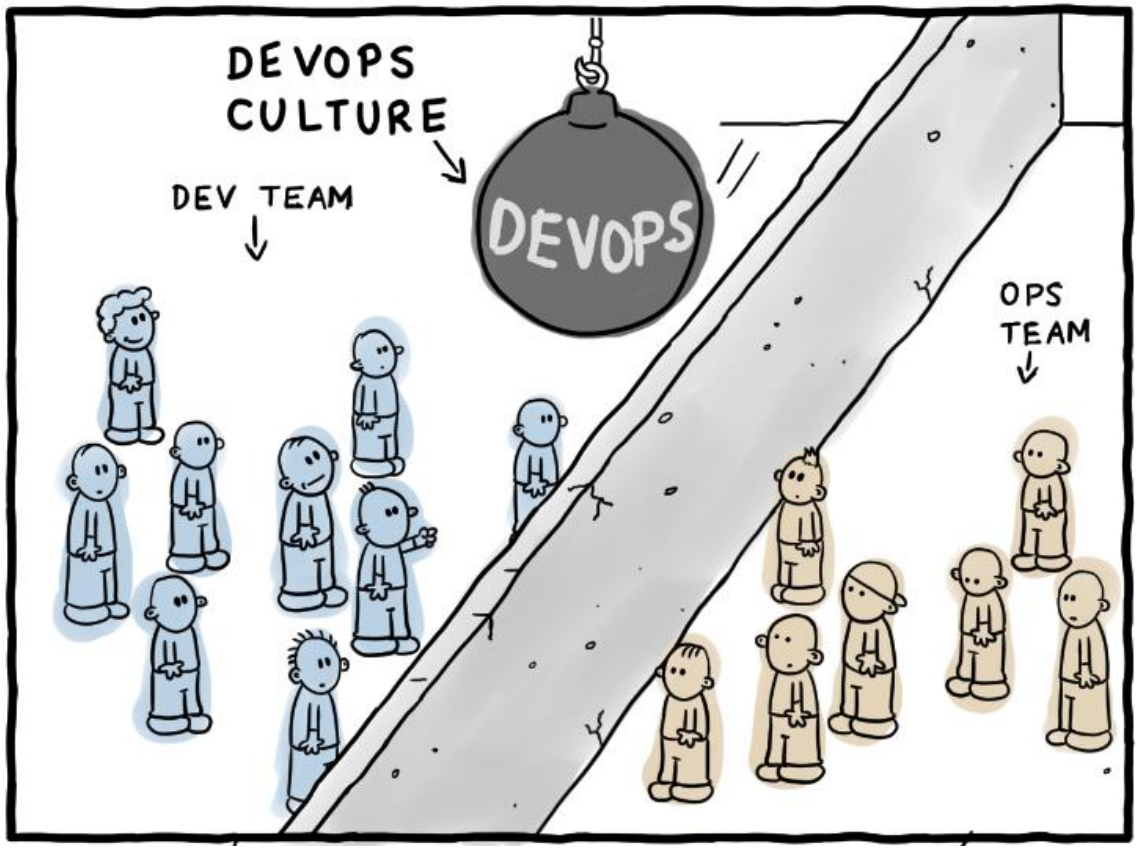


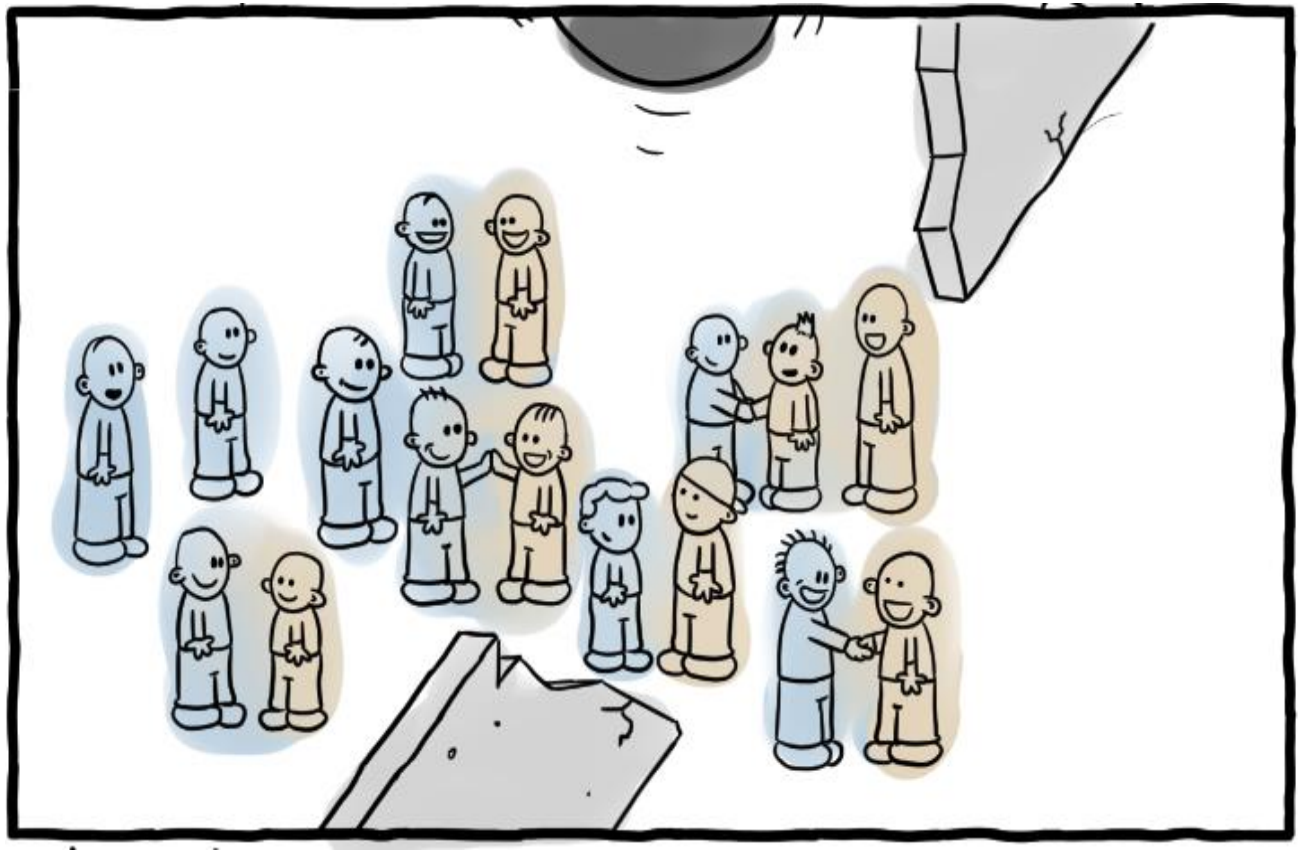
https://turnoff.us/geek/devops-explained/

- In traditional software organization, dev-to-ops handoffs are typically one-way, often limited to a few scheduled times in an application's release cycle
- Once in production, the operations team take charge
- If there are bugs in the code, the virtual assembly line of Devs-to-QAs-to-OPs is revisited with a patch, with each team waiting on the other for next steps, which might take weeks
- Such a model comes with significant overhead and extends the timeline

https://github.blog/2020-12-03-the-evolving-role-of-operations-in-devops/

WITH DEVOPS

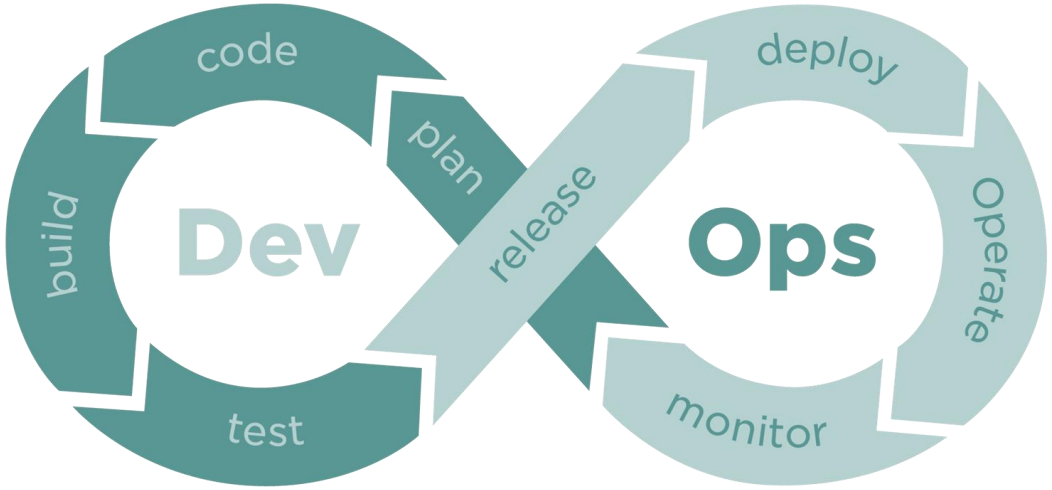




https://turnoff.us/geek/devops-explained/

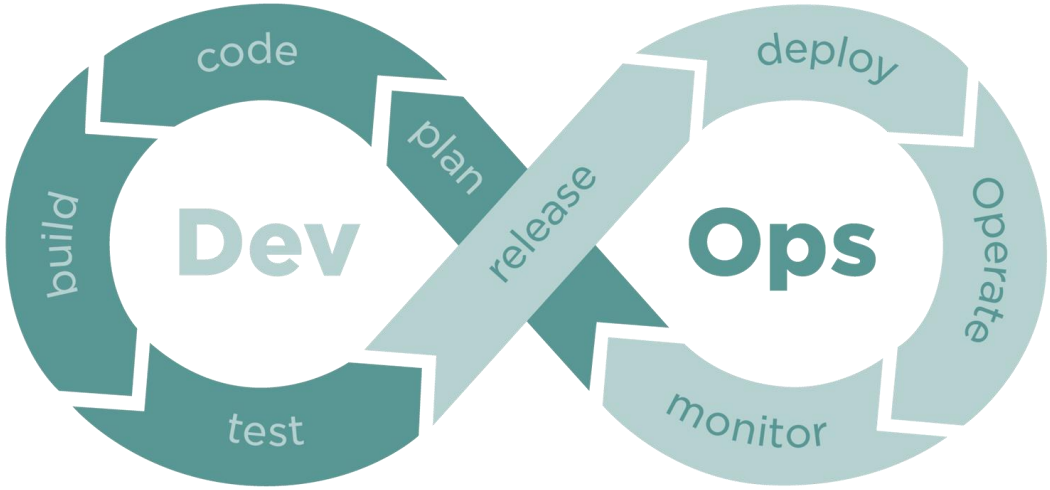
DEVOPS

- DevOps integrates developers and operations teams to improve collaboration and productivity by
  - Automating infrastructure
  - Automating workflows
  - Continuously measuring application performance

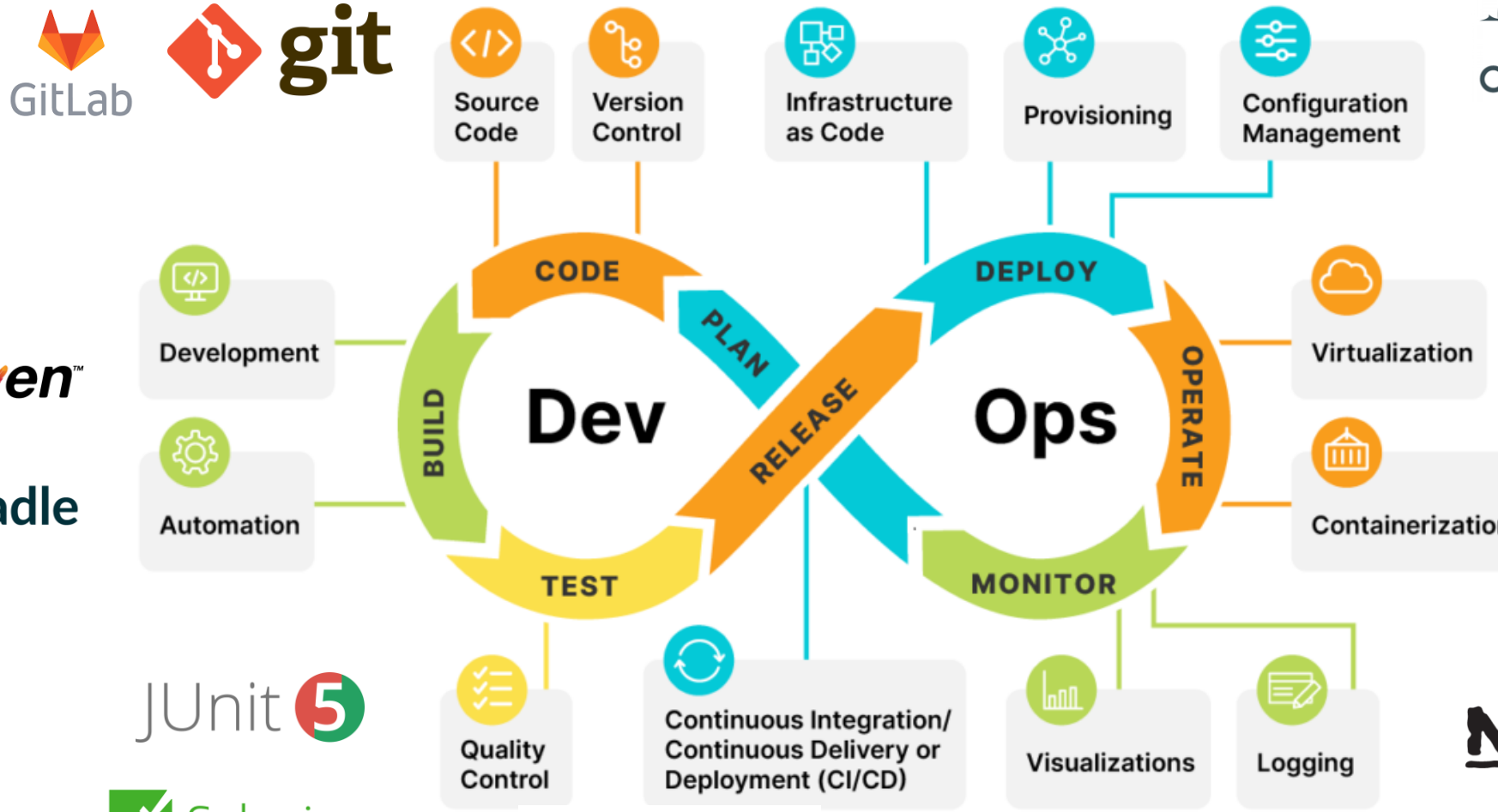


DEVOPS

- DevOps requires that development teams and operations teams **communicate frequently** and approach their work with empathy for their teammates.
  - Developers work closely with IT operations to speed software builds, tests, and releases, without sacrificing reliability.
  - Whoever needs power the most, get it — through self service and automation.



DEVOPS PHASES & TOOLS



GitLab
git
docker
aws
Maven
Gradle
ANSIBLE
kubernetes
JUnit 5
Selenium
Jenkins
CODESHIP
Nagios®
splunk>

NEXT

- Version Control
- Build Systems