

SOFTWARE ENGINEERING

Week 2


Software Development Lifecycle and Process Models

Prof. Dr. Tolga OVATMAN Assoc. Prof. Dr. Ayşe TOSUN

Istanbul Technical University
Computer Engineering Department

Agenda

1. Software Processes
2. Plan Driven Software Process Models
3. Agile Software Process
 - a. Agility
 - b. Common Agile Practices
 - c. Extreme Programming
 - d. Scrum
 - e. Kanban - Scrumban

1. Software Processes 
2. Plan Driven Software Process Models

Software Processes

∞ 2.1 ∞

The Meaning of Process

- ∞ A **process**: a series of steps involving activities, constraints, and resources that produce intended output of some kind
- ∞ A process involves a set of tools and techniques



Reasons for Modeling a Process

- ∞ To form a common understanding
- ∞ To find inconsistencies, redundancies, omissions
- ∞ To define and evaluate appropriate activities for reaching process goals
- ∞ To tailor a general process for a particular situation in which it will be used



Software Life Cycle

∞ When a process involves building a software product, the process may be referred as software (development) life cycle

- Requirements analysis and definition
- System (architecture) design
- Program (detailed/procedural) design
- Writing programs (coding/implementation)
- Testing: unit, integration, system
- System delivery (deployment)

-
- Maintenance

Software Development

☞ Ideally, software is developed as described in

- Linear
- Starting from scratch

☞ In the real world, software development is totally different

- We make mistakes
- The client's requirements change while the software product is being developed

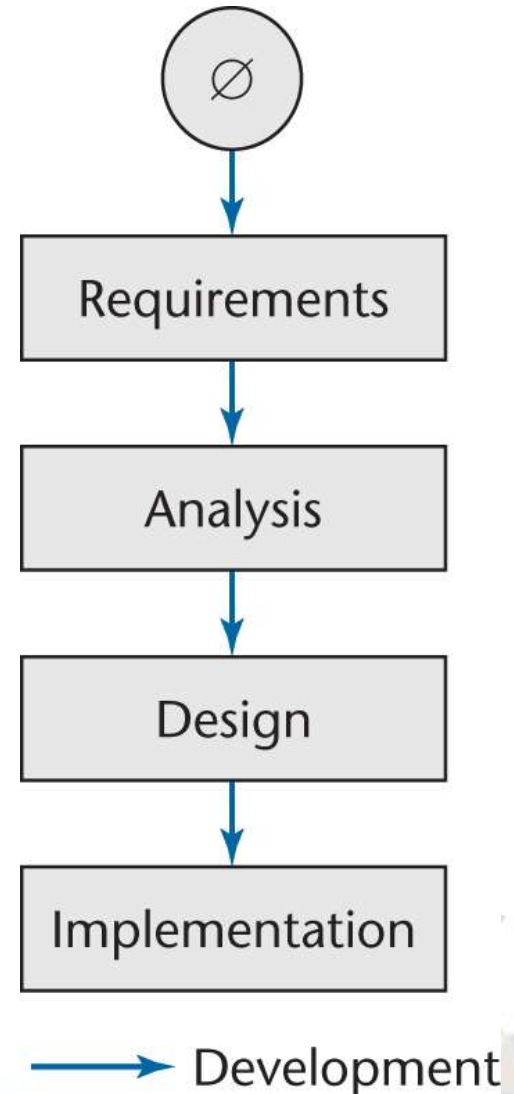


Figure 2.1

Moving Target Problem

- ∞ A change in the requirements while the software product is being developed
- ∞ Even if the reasons for the change are good, the software product can be adversely impacted
 - Dependencies will be induced
- ∞ Any change made to a software product can potentially cause a *regression fault*
 - A fault in an apparently unrelated part of the software
- ∞ If there are too many changes
 - The entire product may have to be redesigned and reimplemented
- ∞ Change is inevitable
 - Growing companies are always going to change
 - If the individual calling for changes has sufficient clout, nothing can be done about it
- ∞ There is no solution for the moving target problem

Iteration and Incrementation

- ✎ In real life, we cannot speak about “the analysis phase”
 - Instead, the operations of the analysis phase are spread out over the life cycle
- ✎ The basic software development process is *iterative*
 - Each successive version is intended to be closer to its target than its predecessor

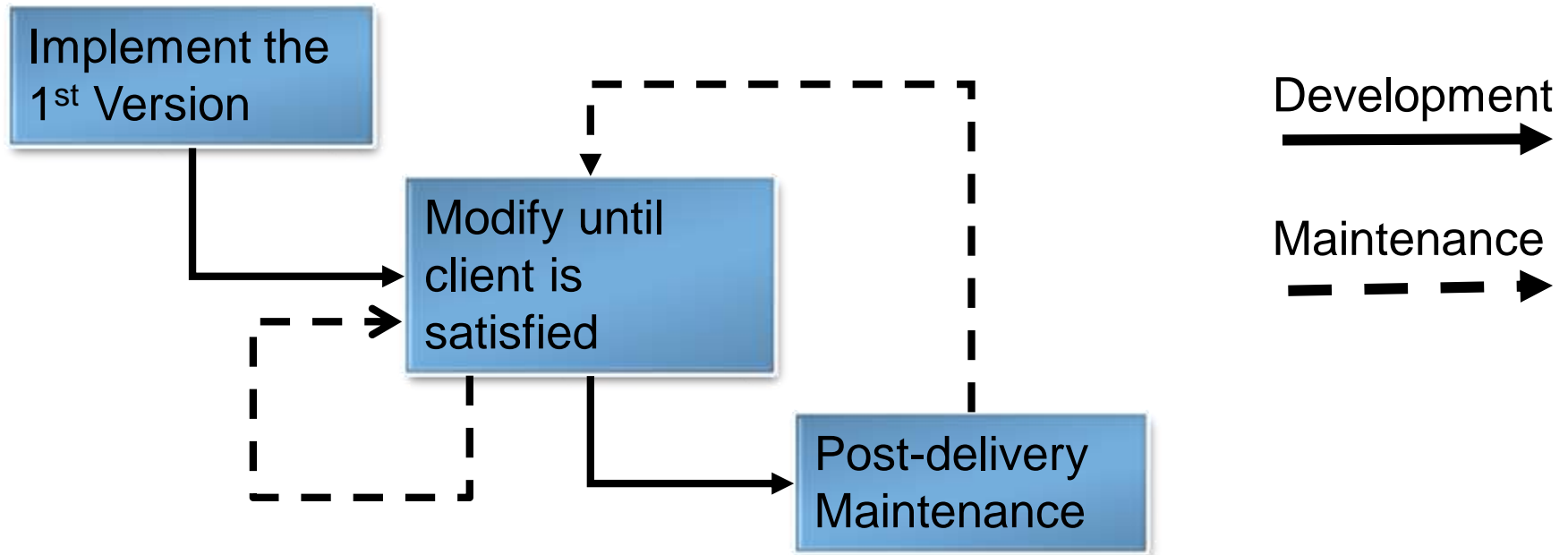
Miller's Law

- ✧ At any one time, we can concentrate on only approximately seven *chunks* (units of information)
- ✧ To handle larger amounts of information, use *stepwise refinement*
 - Concentrate on the aspects that are currently the most important
 - Postpone aspects that are currently less critical
 - Every aspect is eventually handled, but in order of current importance
- ✧ This is an *incremental* process

Software Process Models

- ✎ Software process models are general approaches for organizing a project into activities.
 - Help the project manager and his or her team to decide:
 - What work should be done;
 - In what sequence to perform the work.
 - The models should be seen as *aids to thinking*, not rigid prescriptions of the way to do things.
 - Each project ends up with its own unique plan.

Code-and-Fix Model



- ∞ The easiest way to develop software
- ∞ No design, no specifications
- ∞ Maintenance extremely difficult
- ∞ The most expensive way
- ∞ Typically used by a start-up

Code-and-Fix is also called 'Opportunistic Approach'

- ✎ When an organization does not follow good engineering practices
- ✎ It does not acknowledge the importance of working out the requirements and the design before implementing a system.
- ✎ The design of software deteriorates faster if it is not well designed.
- ✎ Since there are no plans, there is nothing to aim towards.
- ✎ There is no explicit recognition of the need for systematic testing and other forms of quality assurance.
- ✎ The above problems make the cost of developing and maintaining software very high.

1. Software Processes
2. Plan Driven Software Process Models 

Plan Driven Software Process Models

∞ 2.2 ∞

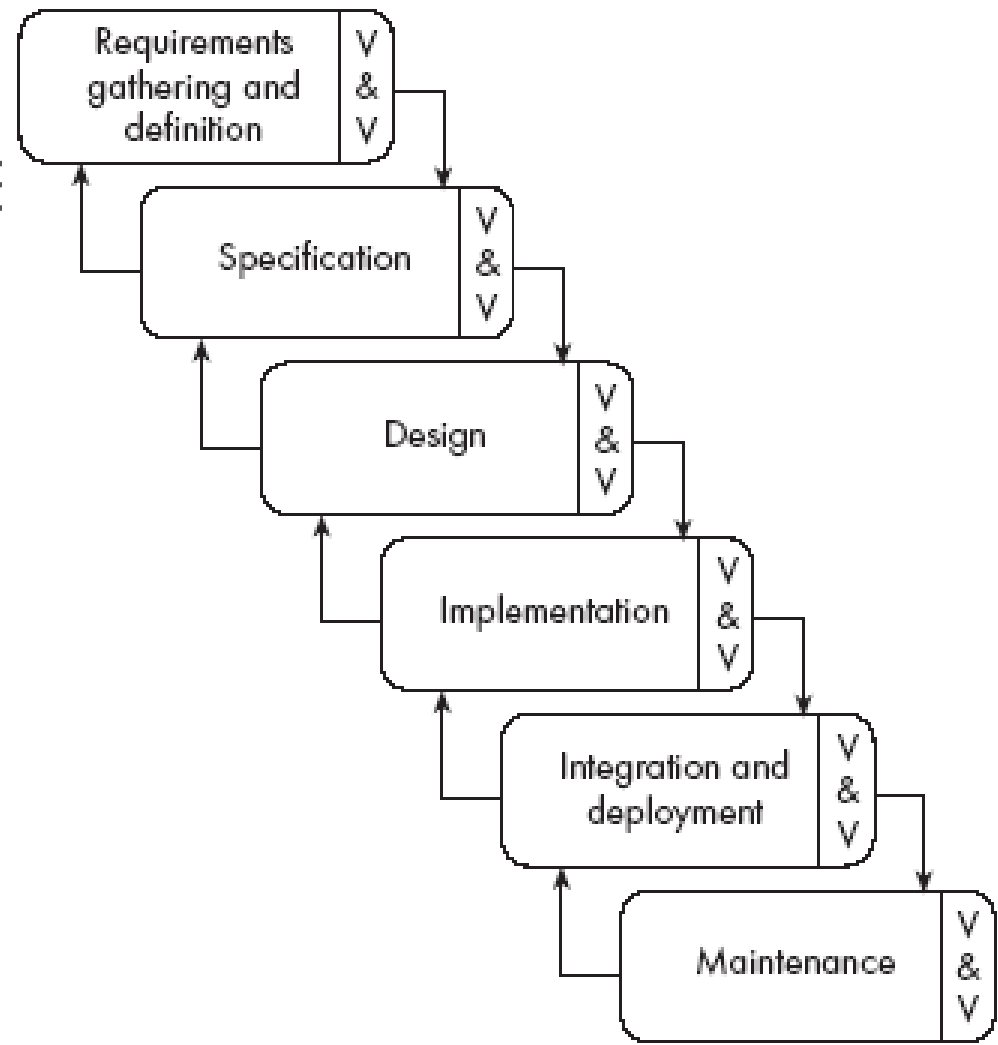
Plan-Driven Software Process Models

1. Waterfall model
2. Iterative/Incremental model
3. Component-Based model
4. Rapid prototyping model
5. Spiral model
6. Unified Process model

✎ In practice, most large systems are developed using a process that incorporates elements from all of these models.

1. Waterfall Model

- ∞ Classical approach of SE
- ∞ Separate and distinct phases
- ∞ Feedback loops (to a limited extent)
- ∞ Documentation-driven



1. Waterfall Model Pros and Cons

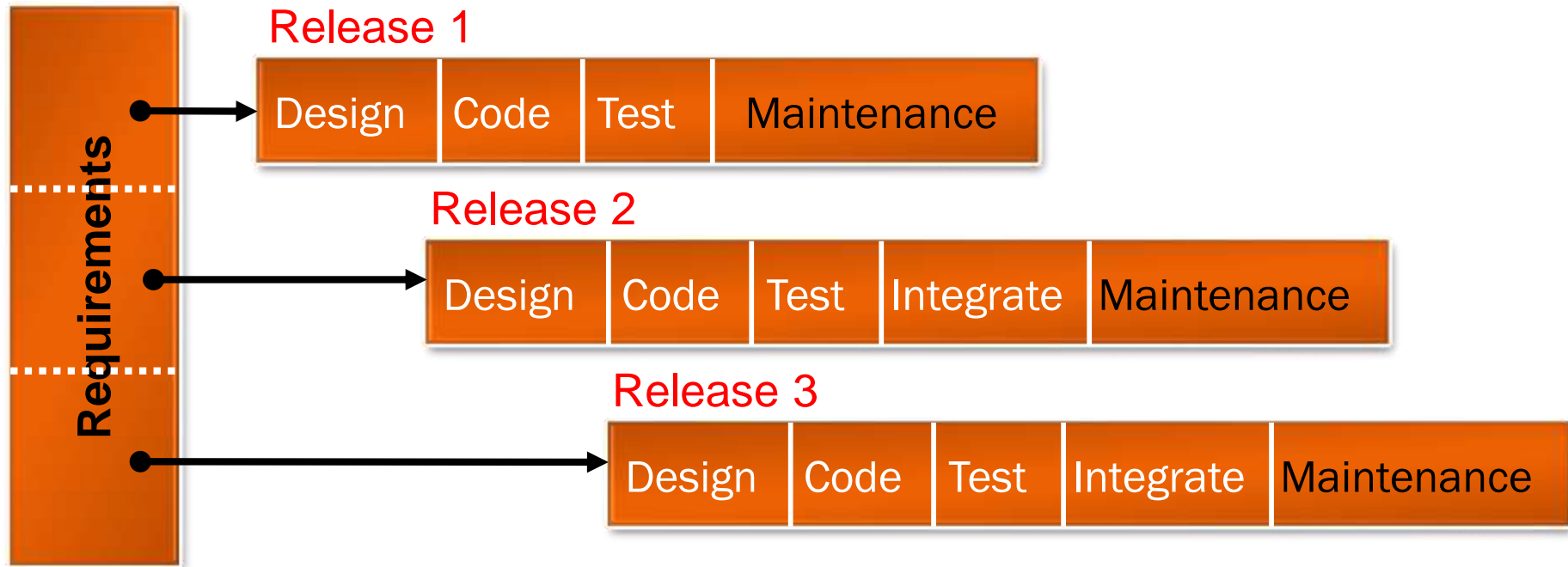
Pros

- ∞ Simple and disciplined, structured approach
 - ∞ Project Management is easy
 - ∞ Maintenance is easier
- This model is only appropriate when the requirements are well-understood and changes will be limited during the design process.
 - The waterfall model is mostly used for large system engineering projects where a system is developed at several sites.
 - In those circumstances, the plan-driven nature of the waterfall model helps to coordinate the work.

Cons

- ∞ Difficulty of accommodating changes after the process is underway.
- ∞ Necessitates stable requirement
 - Few business systems have stable requirements.
 - Military, Government Projects
- ∞ Major design problems may not be detected till very late.
- ∞ Very late delivery
- ∞ Blocking phases
 - Coders must wait designers to prepare design document

2. Incremental Development



- Avoids “big bang” implementation
- Assumes all requirements are known up-front
- Each release adds more functionality
- Once the development of an increment is started, the requirements are **frozen** though requirements for later increments can continue to evolve.

Iteration and/or Incrementation

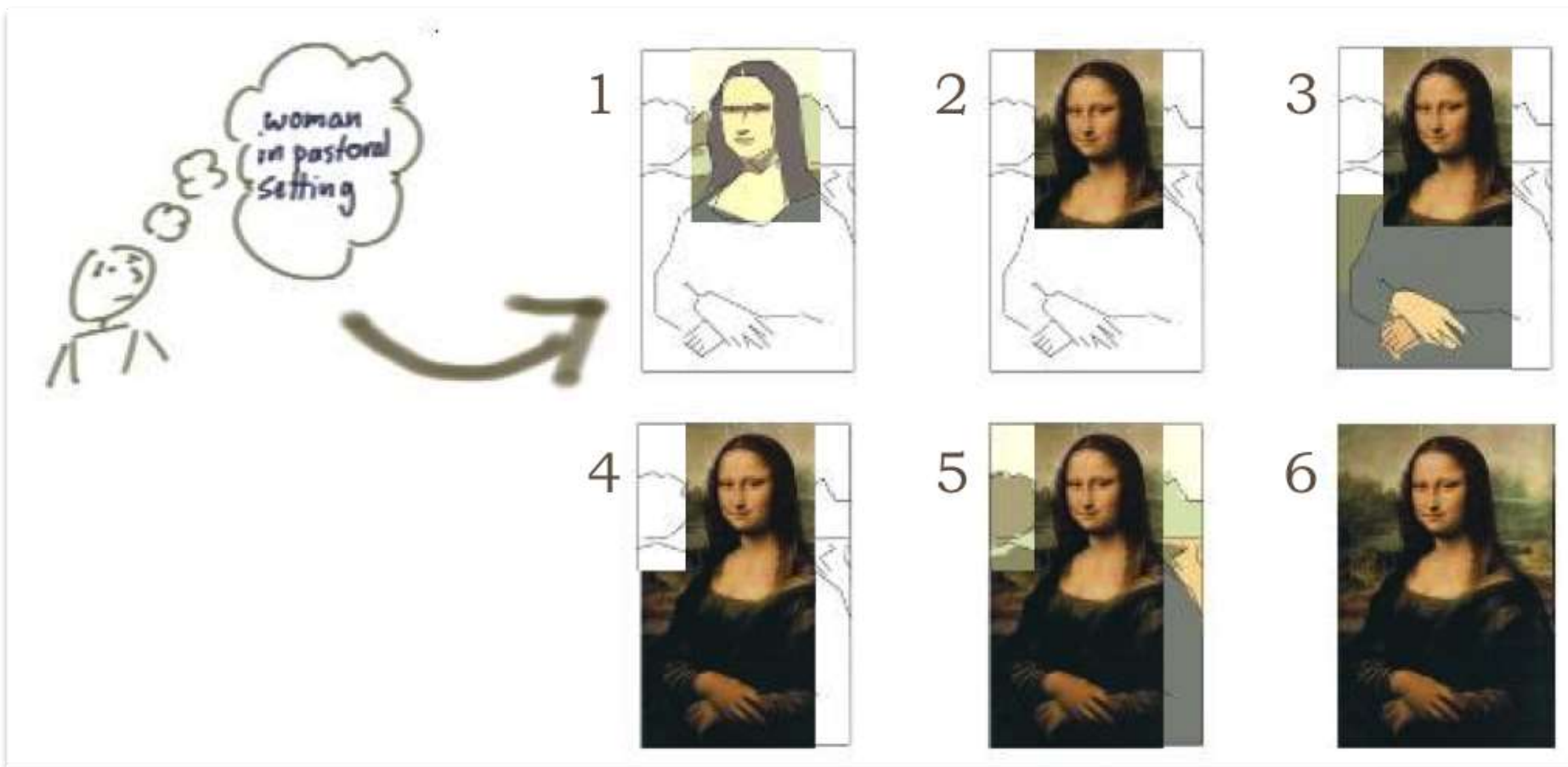
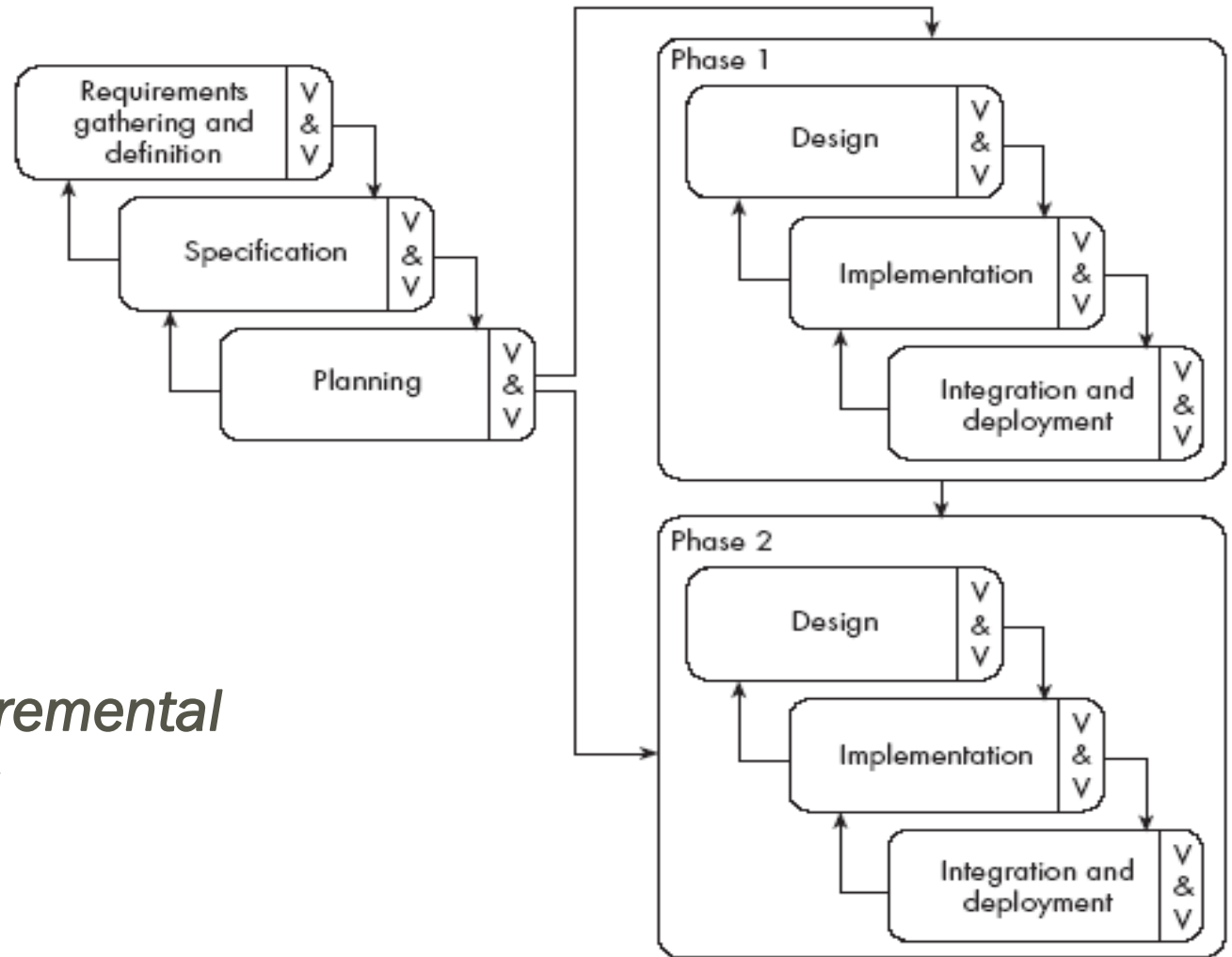


Figure 2.5

Another representation for a phased-release model



∞ Notion of *incremental development*

∞ *Not* iterative

2. Incremental Development Pros and Cons

Pros

- ∞ The cost of accommodating changing customer requirements is reduced.
 - The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- ∞ It is easier to get customer feedback on the development work that has been done.
 - Customers can comment on demonstrations of the software and see how much has been implemented.
- ∞ Rapid delivery and deployment of useful software to the customer is possible.
 - Customers are able to use and gain value from the software earlier than it is possible with a waterfall process.

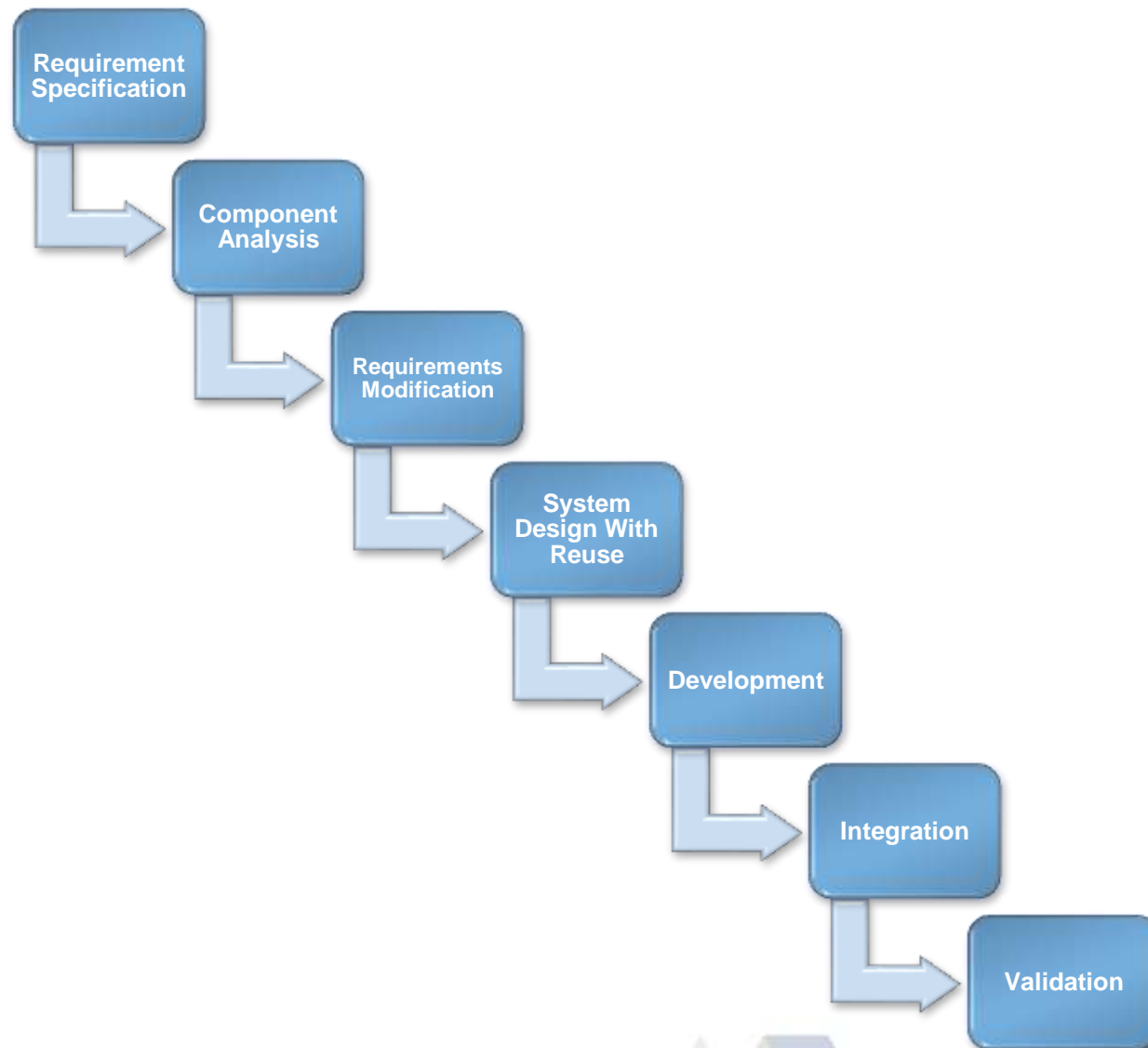
Cons

- ∞ The process is not visible.
 - Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
- ∞ System structure tends to degrade as new increments are added.
 - Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

3. Component Based Development

- ✎ Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.
- ✎ Process stages
 - Component analysis
 - Requirements modification
 - System design with reuse
 - Development and integration
- ✎ Reuse is now the standard approach for building many types of business system

3. Component Based Development



3. Component Based Development Pros & Cons

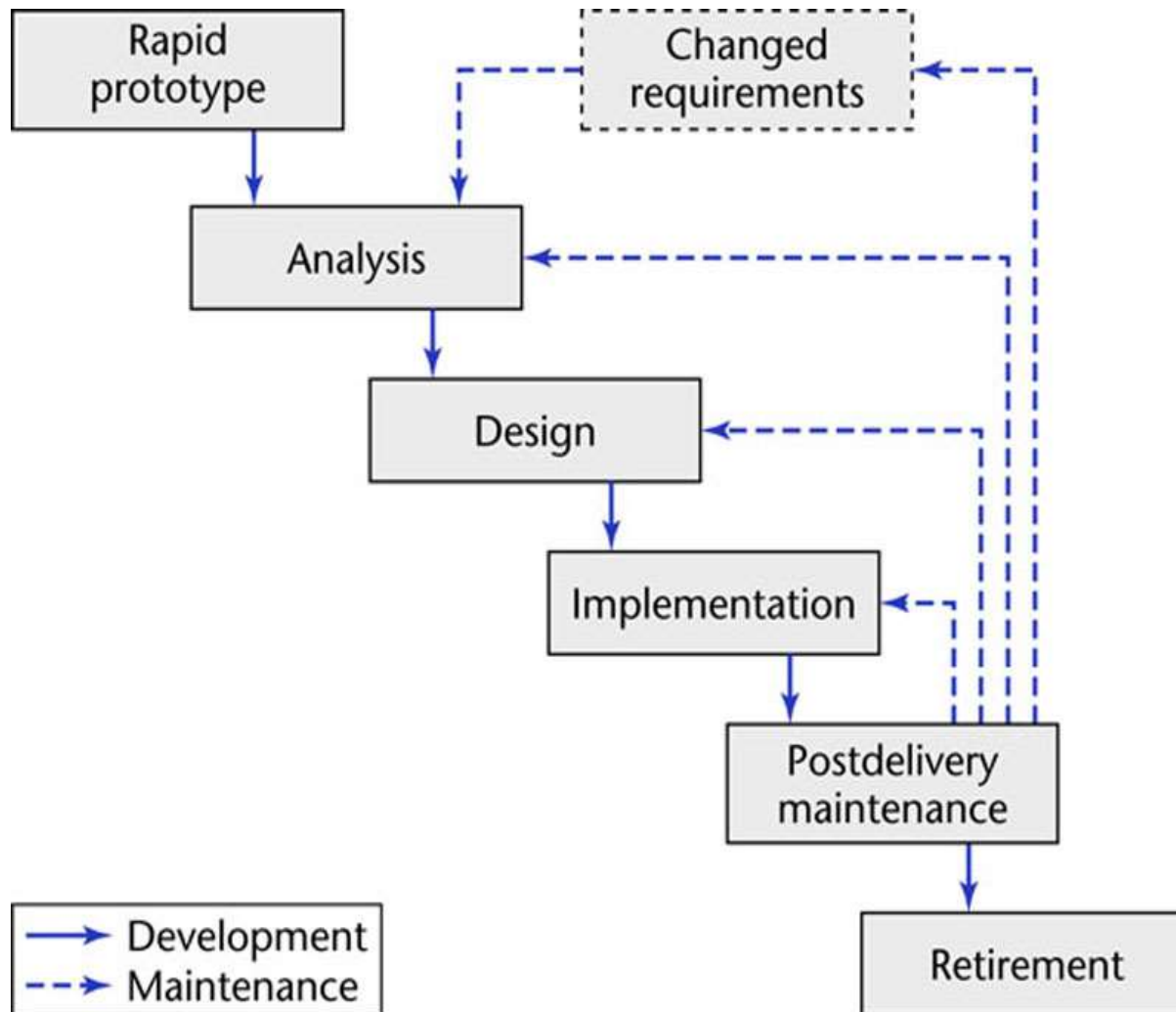
Pros

- ∞ system reliability is increased (standard reusable components should be well tested and perhaps formally verified)
- ∞ development time is reduced
 - design and coding time is reduced
 - testing time is reduced
- ∞ standards can be implemented as reusable components
 - standards for fault-tolerance or correctness
 - standards for user interfaces
 - a company's "look and feel" could come from reuse of standard user interface components

Cons

- ∞ reusing components leads to changes in design and requirements
- ∞ developers always believe they could develop better components anyway
- ∞ incorporating component libraries often leads to larger and less efficient implementations
- ∞ organizations are reluctant to expand resources (\$\$) to develop reusable components
- ∞ no standard way to catalog and search for reusable components
- ∞ no guarantee that reusing components leads to faster development or more reliable systems
- ∞ new versions of purchased components are not controlled by the development organization, which may affect system evolution

4. Rapid Prototyping Model



4. Rapid Prototyping Model

- ∞ Prototyping is used for:
 - understanding the requirements for the user interface
 - can start with initial requirements to clarify what is really needed
 - examining feasibility of a proposed design approach
 - exploring system performance issues
- ∞ Preferred for new technology projects.
- ∞ A prototype has only a limited capability.
- ∞ Prototyping often takes 3-4 months.

4. Rapid Prototyping Model Pros and Cons

Pros

- ✎ Improved system usability.
- ✎ A closer match to users' real needs.
- ✎ Improved design quality.
- ✎ Improved maintainability.
- ✎ Reduced development effort.

Cons

- ✎ Usually the customer insists on «small modifications» to prototype system after seeing sth appears to be a working version of the software.
- ✎ The developer may use inappropriate components for building prototype quickly. By the time, they get comfortable with the choices and forget all reasons why there were inappropriate. Less-than-ideal choices become a part of the system.

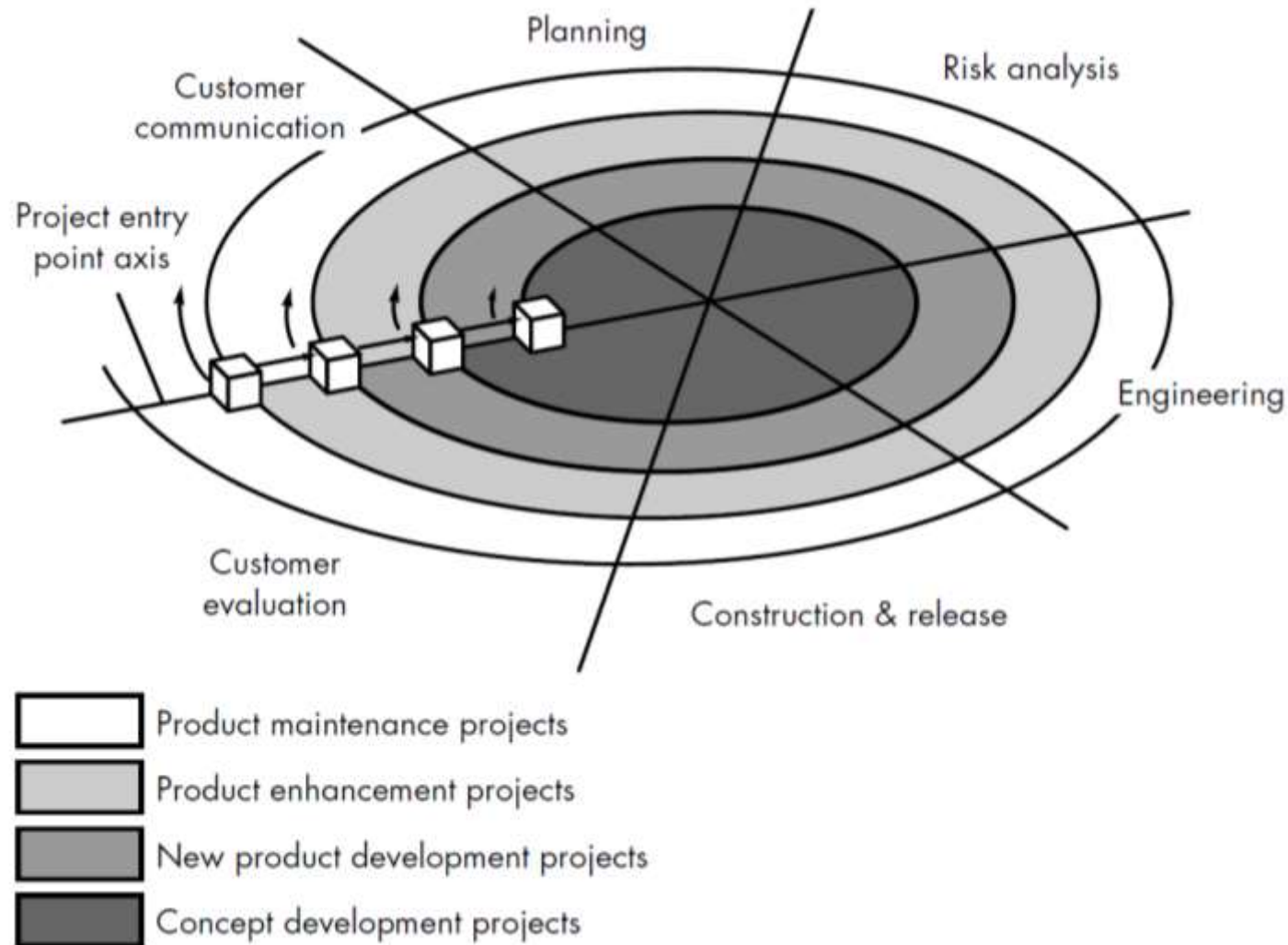
4. Prototype Development and Retirement

- ✎ May be based on rapid prototyping languages or tools
- ✎ May involve leaving out functionality
 - Prototype should focus on areas of the product that are not well-understood;
 - Error checking and recovery may not be included in the prototype;
 - Focus on functional rather than non-functional requirements such as reliability and security
- ✎ Prototypes should be discarded after development as they are not a good basis for a production system:
 - It may be impossible to tune the system to meet non-functional requirements;
 - Prototypes are normally undocumented;
 - The prototype structure is usually degraded through rapid change;
 - The prototype probably will not meet normal organizational quality standards.

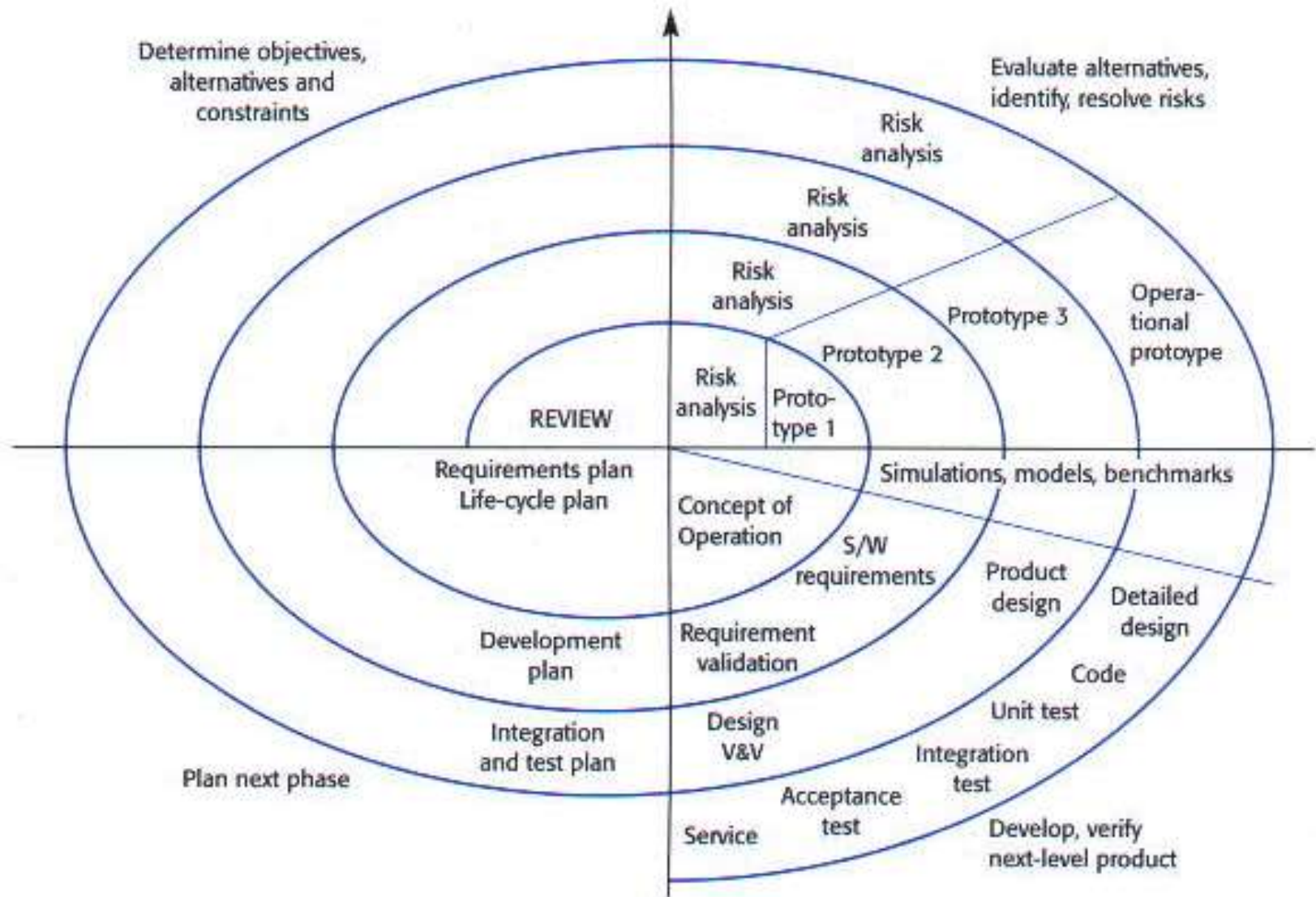
5. Spiral Model

- ✎ The spiral model is a software development process combining the elements of both design and prototyping-in-stages.
- ✎ This model of development combines the features of the **prototyping** model and the **waterfall** model. It is also **iterative**.
- ✎ The spiral model is intended for large, expensive and complicated projects.
- ✎ Process is represented as a spiral rather than as a sequence of activities with backtracking.
- ✎ Each loop in the spiral represents a phase/release in the process.
- ✎ No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required.
- ✎ Risks are explicitly assessed and resolved throughout the process.

5. Spiral Model



5. Spiral Model



5. Spiral Model Sectors

- ✎ Objective setting
 - Specific objectives for the phase are identified.
- ✎ Risk assessment and reduction
 - Risks are assessed and activities put in place to reduce the key risks.
- ✎ Development and validation
 - A development model for the system is chosen which can be any of the generic models.
- ✎ Planning
 - The project is reviewed and the next phase of the spiral is planned.

An example project built with spiral model:

<https://xbsoftware.com/blog/software-development-life-cycle-spiral-model/>

5.Spiral Model Pros and Cons

Pros

- ✎ Risk monitoring makes the project transparent
- ✎ Customer can see the Product at early stages
- ✎ Changes can be added
- ✎ Project can be divided into several parts depending on the risks associated with them.
- ✎ Estimates of cost and effort may be more realistic
- ✎ Strong documentation

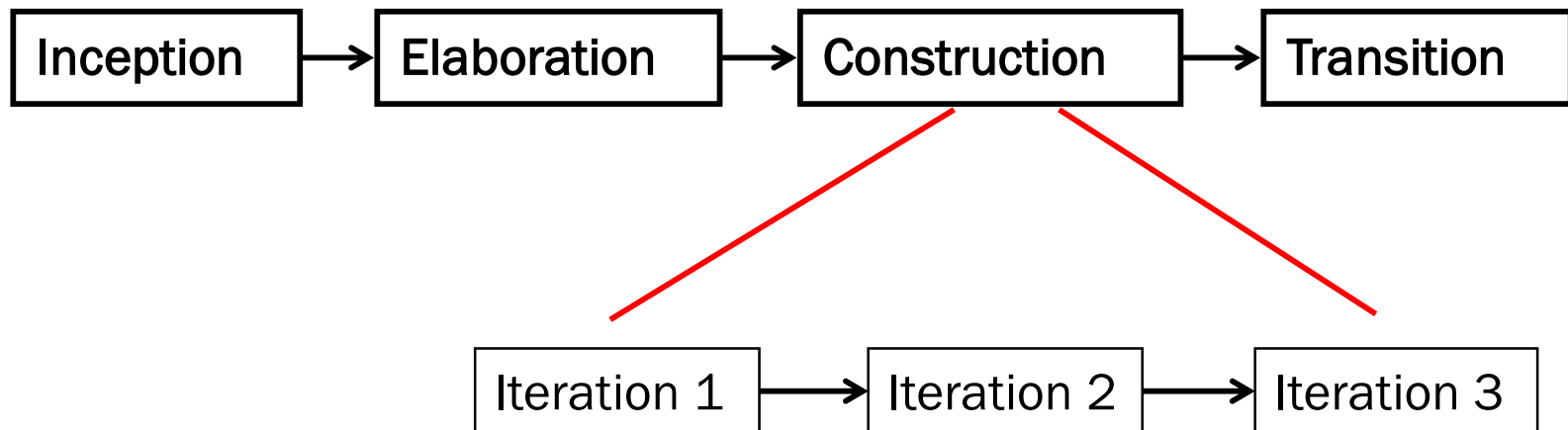
Cons

- ✎ Risk monitoring requires additional resources, this model can be costly to use.
- ✎ Each spiral requires specific expertise.
- ✎ Many intermediate stages
- ✎ Vast amount of documentation
- ✎ Time management may be difficult

6. The Unified Process

- ✎ The Unified Process (UP) is a “use-case driven, iterative and incremental” software process model closely aligned with Object-Oriented Analysis and Design.
- ✎ A modern generic process derived from the work on the UML and associated process.
- ✎ Brings together aspects of the 3 generic process models discussed previously.
- ✎ Normally described from 3 perspectives
 - A dynamic perspective that shows phases over time;
 - A static perspective that shows process activities;
 - A practical perspective that suggests good practice.
- ✎ Each phase ends at a major milestone and contains one or more iterations.
- ✎ An iteration is a distinct sequence of activities with an established plan and evaluation criteria, resulting in an executable release.

6. The Unified Process - II



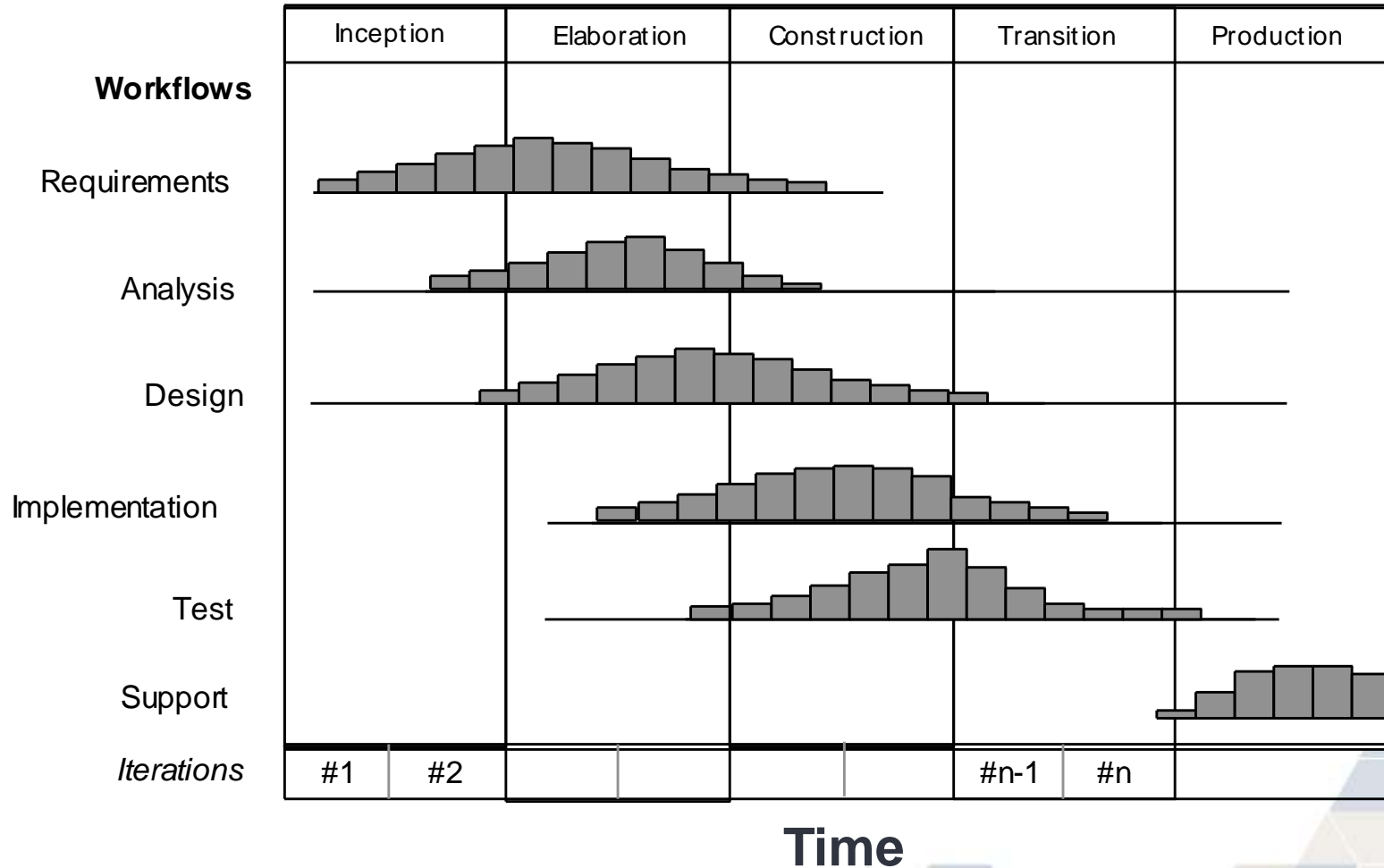
- Each iteration is defined in terms of the scenarios it implements.

6. The Unified Process—Workflows&Phases

- Workflow is Technical context of a step.
- Phase is Business context of a step.

Person-hours

UP Phases



6. The Unified Process Phases-I

1. Inception

- ✎ Establish business rationale for project
- ✎ Decide project scope
- ✎ Identify actors and use cases
- ✎ Work Products (artifacts):
 - Vision document
 - Initial use-case model
 - Initial risk assessment
 - Project plan
 - Prototype

2. Elaboration

- ✎ Collect more detailed requirements
- ✎ Do high-level analysis and design
- ✎ Establish baseline architecture
- ✎ Create construction plan
- ✎ Work Products:
 - Use-case model
 - Non-functional requirements
 - Analysis model
 - Software architecture description
 - Preliminary design model
 - Preliminary user manual


6. The Unified Process Phases-II

3. Construction

- ✧ Build, test and validate the project
- ✧ Work Products:
 - Design model
 - Software components
 - Test plan and test cases
 - Support documentation
 - User manuals
 - Installation manuals
 - Description of current increment

4. Transition

- ✧ Beta-test
- ✧ Tune performance
- ✧ Train users
- ✧ Work Products:
 - Delivered software increment
 - Beta test results
 - General user feedback

1. Agility 
2. Common Agile Practices
3. Extreme Programming
4. Scrum
5. Kanban

Agility

∞ 3.1 ∞

Rapid Software Development

- ✎ Rapid development and delivery is now often the most important requirement for software systems
 - Specification, design and implementation are interleaved
 - System is developed as a series of versions with stakeholders involved in version evaluation
 - User interfaces are often developed using an IDE and graphical toolset.

Agile Manifesto

- ∞ “We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
- **Individuals** and **interactions** over **processes and tools**
 - **Working software** over **comprehensive documentation**
 - **Customer collaboration** over **contract negotiation**
 - **Responding to change** over **following a plan**
- ∞ That is, while there is value in the items on the right, we value the items on the left more.”

Kent Beck et al.

www.agilemanifesto.org

Agile vs Plan Driven (in short)

∞ Plan Driven


- It has a separate planning stage, also done by separate group of people.
 - This is called **predictive** planning.
- It is also process centric

∞ Agile

- It does **adaptive** planning and execution together by the same team.
- It is also group (people) centric
- Agile team decides what to do, how to do, and they are aware of the processes, and if necessary they are capable of changing the process to make things better.

Principles of Agile Methods

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

1. Agility
2. Common Agile Practices 
3. Extreme Programming
4. Scrum
5. Kanban

Common Agile Practices

∞ 3.2 ∞

User Stories

- ✧ In most of the agile methods, customer or user is part of the team and is responsible for making decisions on requirements.
- ✧ User requirements are expressed as scenarios or user stories.
- ✧ These are written on cards and the development team break them down into implementation tasks. These tasks are the basis of schedule and cost estimates.
- ✧ The customer chooses the stories for inclusion in the next release based on their priorities and the schedule estimates.

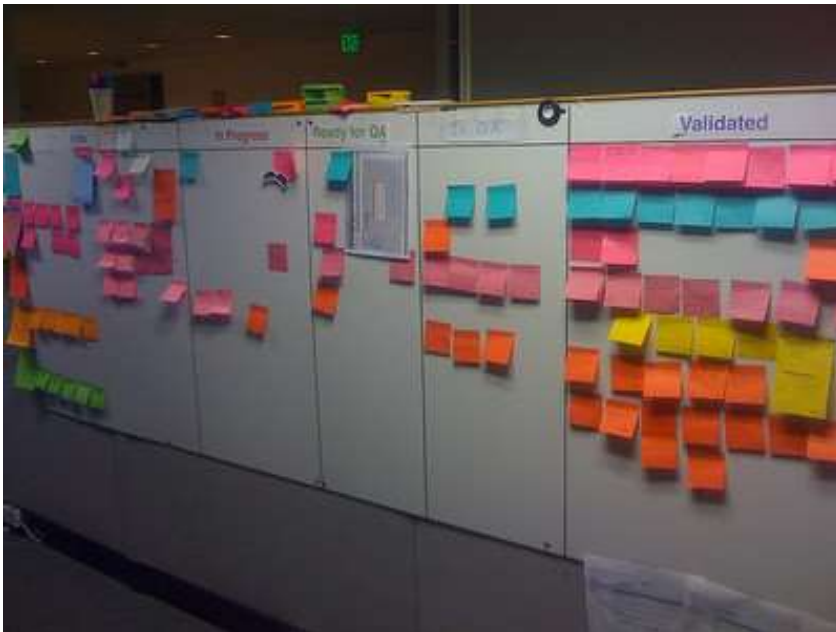


Copyright © 2003 United Feature Syndicate, Inc.

Story Boards

Display Scanned Item

When an item is scanned, the point of sale terminal displays a short description of the item and its price.



Story Points

- ∞ Story point is an arbitrary measure used to measure the effort required to implement a story.
- ∞ It is a number that tells the team how hard the story is. Hard could be related to complexity, unknowns and effort but not time

1, 2, 4, 8, 16, 32, 64

XS, S, M, L, XL, XXL

1, 2, 3, 5, 8, 13, 21, 100

1, 5, 10, 25, 50, 100

- ∞ For every team, story size could mean different things depending on what baseline they chose

Planning Poker

- ✧ Planning poker is a game where the development team estimates stories individually first and then compares the results collectively together after
- ✧ If everyone's estimate is more or less the same, the estimate is kept. If there are differences, however, the team discusses them and estimates again until consensus is reached.



Refactoring

- ✎ Refactoring is the process of changing the design of your code without changing its behavior—what it does stays the same, but how it does changes.
- ✎ You can think of it like improving a function-method-class without changing its interface.
- ✎ However, some changes requires architecture refactoring and this is much more expensive.
- ✎ Examples of refactoring:
 - 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.

[Martin Fowler is a great source for refactoring as well as agile and architecture.](#)

Pair Programming Environment



Advantages of Pair Programming

- ✧ It supports the idea of collective ownership and responsibility for the system.
 - Individuals are not held responsible for problems with the code. Instead, the team has collective responsibility for resolving these problems.
- ✧ It acts as an informal review process because each line of code is looked at by at least two people.
- ✧ It helps support refactoring, which is a process of software improvement.
 - Where pair programming and collective ownership are used, others benefit immediately from the refactoring so they are likely to support the process.

Collective Ownership

- ❧ Collective code ownership spreads responsibility for maintaining the code to all the programmers. Collective code ownership is exactly what it sounds like: everyone shares responsibility for the quality of the code.
- ❧ No single person claims ownership over any part of the system, and anyone can make any necessary changes anywhere.
- ❧ Collective code ownership requires letting go of a little bit of ego. Rather than taking pride in your code, take pride in your team's code.

Always leave the code a little better than you found it.

Collective Ownership

- ✎ How can you take ownership of code that you don't understand?
 - take advantage of pair programming
 - Rely on the unit tests for further documentation and as your safety net
 - As you work, look for opportunities to refactor the code.
- ✎ Collective code ownership shares knowledge and improves skills, but it won't make everyone an expert at everything. Take advantage of all skills and specialties.
- ✎ Rather than turning your junior programmers loose on the code, make sure they pair with experienced members of the team.
- ✎ Collective ownership increases the possibility of merge conflicts, and so it also requires continuous integration. Continuous integration decreases the chances of merge conflicts.


Continuous Integration

- ✎ The ultimate goal of continuous integration is to be able to deploy all but the last few hours of work at any time.
- ✎ Most software development efforts have a hidden delay between when the team says “we’re done” and when the software is actually ready to ship.
- ✎ The point is to be technologically ready to release even if you’re not functionally ready to release.
 - Integrate your code every few hours.
 - Keep your build, tests, and other release infrastructure up-to-date.
- ✎ To guarantee an always-working build
 - make sure what works on your computer will work on anybody’s computer.
 - nobody gets code that hasn’t been proven to build successfully.

Continuous Integration

- ✎ There's a lively community of open-source continuous integration servers
- CruiseControl
 - Jenkins
 - TeamCity
 - GitLab CI
 - Travis CI

Continuous Integration

 **Projects** | **My Changes** | **Agents (73)** | **Build Queue (11)**

Welcome, Pavel Sher | Administration

[Collapse All](#) | [Expand All](#) Hide Successful [Configure Visible Projects](#)

▼ **Eluru** TeamCity trunk 2 hidden ×

▼ **BuildDist**

Pending (7) Run

#17111 Tests passed: 321 Artifacts Changes (17) 05 Feb 11 05:02 (1h:0m)

▼ **Compile**

Run

#26374 Success Artifacts pavel.sher (1) 05 Feb 11 16:28 (11m:28s)

▼ **IntegrationBuild (JDK16 + IDEA X)**

Run

#870 Tests passed: 2349, ignored: 5 No artifacts Changes (6) 05 Feb 11 16:53 (38m:18s)

▼ **IntegrationBuild (Java Based Runners)**

Run

#1039 Tests passed: 258, ignored: 2 No artifacts Changes (6) 42m:48s left Stop

#1038 Tests passed: 571, ignored: 2 No artifacts eugene.petrenko (1) 05 Feb 11 14:49 (1h:08m)

▼ **IntegrationBuild (.NET Runners)**

Run

#1941 Tests passed: 362 No artifacts Changes (6) 1h:21m left Stop

#1940 Tests passed: 1210, ignored: 3 No artifacts eugene.petrenko (1) 05 Feb 11 14:49 (1h:56m)

▼ **IntegrationBuild (HSQLDB)**

Run

#15499 Tests passed: 3710, ignored: 5 No artifacts Changes (4) 35m:05s left Stop

#15498 Tests passed: 5673, ignored: 12 No artifacts victory.bedroso... (1) 05 Feb 11 15:42 (1h:13m)

▼ **IntegrationBuild (MySQL)**

Run

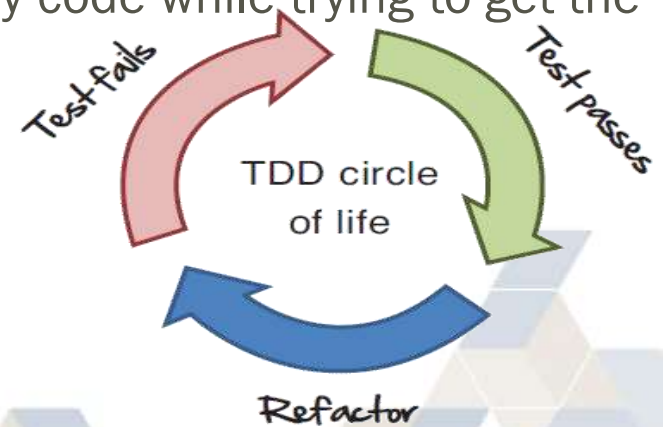
#19066 target: fetchArtifacts-compile No artifacts pavel.sher (1) 1h:26m left Stop

#19065 Tests failed: 3, passed: 4133, ignored: 6 No artifacts Changes (3) 31m:35s left Stop

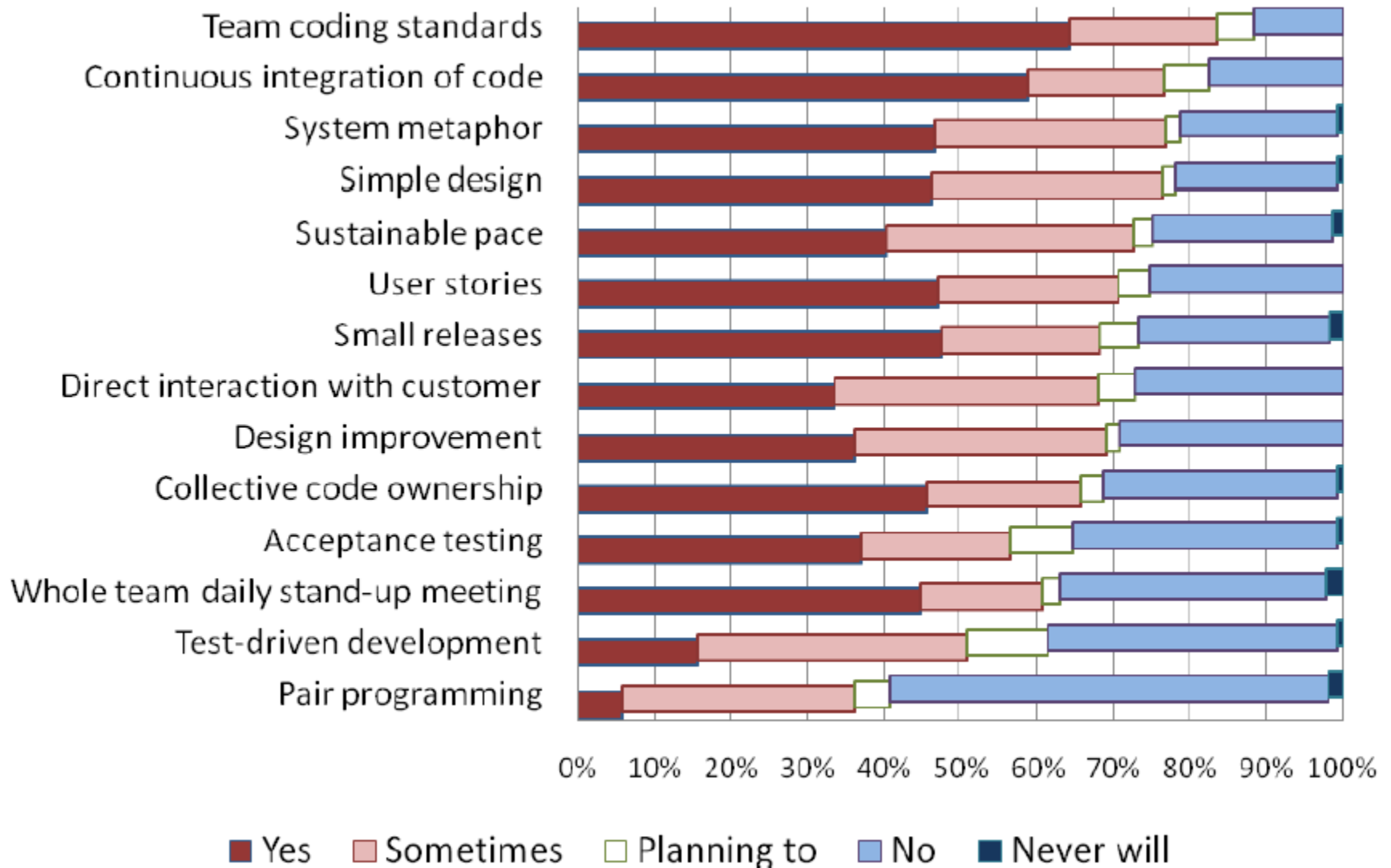
#19064 Tests failed: 3, passed: 5638, ignored: 12 No artifacts Changes (2) 05 Feb 11 15:41 (1h:51m)

Test Driven Development

- ∞ Test-driven development (TDD) is a software development technique that uses really short development cycles to incrementally design your software.
- Red: Before you write any new code for the system, you first write a failing unit test, showing the intent of what you would like the new code to do. Here you are thinking critically about the design.
 - Green: Then you do whatever it takes to make the test pass. If you see the full implementation, add the new code. If you don't, do just enough to get the test to pass.
 - Refactor: Then you go back and clean up any code while trying to get the test to pass.



Overall Usage



Overall Usage

Table 9. Usage of specific agile practices

Practices	n	Mean	Median
Prioritized work list	204	4,2	4
Iteration/sprint planning	203	4,1	4
Daily stand-up meetings	209	3,7	4
Unit testing	199	3,7	4
Release planning	196	3,9	4
Active customer participation	196	3,5	4
Self-organizing teams	194	3,5	4
Frequent and incremental delivery of working software	189	4,1	4
Automated builds	185	3,5	4
Continuous integration	182	3,8	4
Test-driven development (TDD)	179	2,7	3
Retrospectives	177	3,6	4
Burn-down charts	174	3,2	3
Pair programming	174	2,4	2
Refactoring	163	3,4	3
Collective code ownership	159	3,3	3
Other	9	1,8	1

1. Agility
2. Common Agile Practices
3. Extreme Programming
4. Scrum
5. Kanban



Extreme Programming

∞ 3.3 ∞

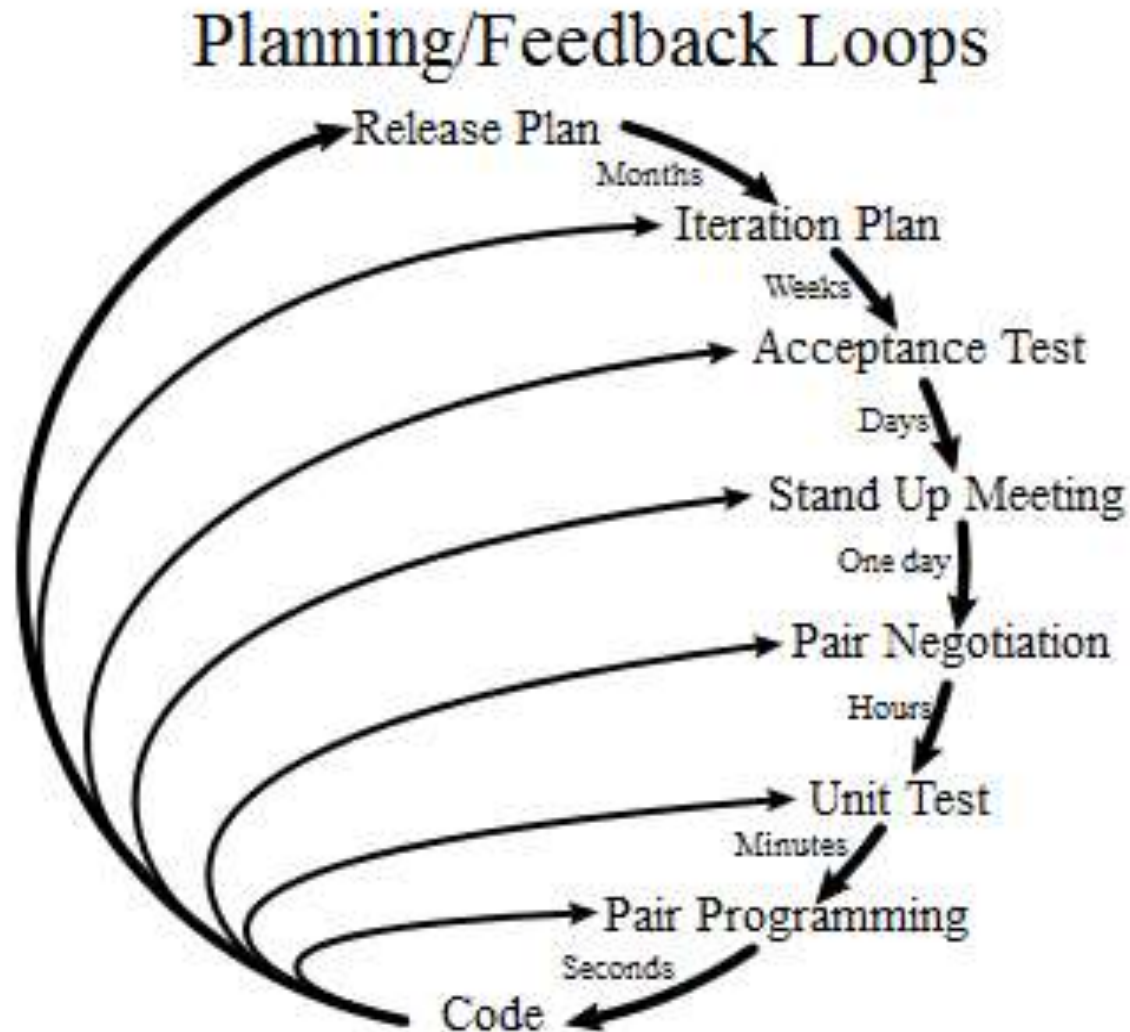
Extreme Programming

- ✎ The most widely used agile process, originally proposed by Kent Beck
- ✎ Extreme Programming (XP) takes an 'extreme' approach to iterative development.
 - New versions may be built several times per day;
 - Increments are delivered to customers every 2 weeks;
 - All tests must be run for every build and the build is only accepted if tests run successfully.

XP Principles

- ⌘ Incremental development is supported through small, frequent system releases.
- ⌘ Customer involvement means full-time customer engagement with the team.
- ⌘ People not process through pair programming, collective ownership and a process that avoids long working hours.
- ⌘ Change supported through regular system releases.
- ⌘ Maintaining simplicity through constant refactoring of code.

XP loops



XP Phases

∞ XP Planning

- Begins with the creation of “user stories”
- Agile team assesses each story and assigns a cost
- Stories are grouped to for a deliverable increment
- A commitment is made on delivery date
- After the first increment “project velocity” is used to help define subsequent delivery dates for other increments

∞ XP Design


- Follows the KIS principle
- Encourage the use of CRC cards
- For difficult design problems, suggests the creation of “spike solutions”—a design prototype
- Encourages “refactoring”—an iterative refinement of the internal program design

∞ XP Coding

- Recommends the construction of a unit test for a store before coding commences
- Encourages “pair programming”

∞ XP Testing

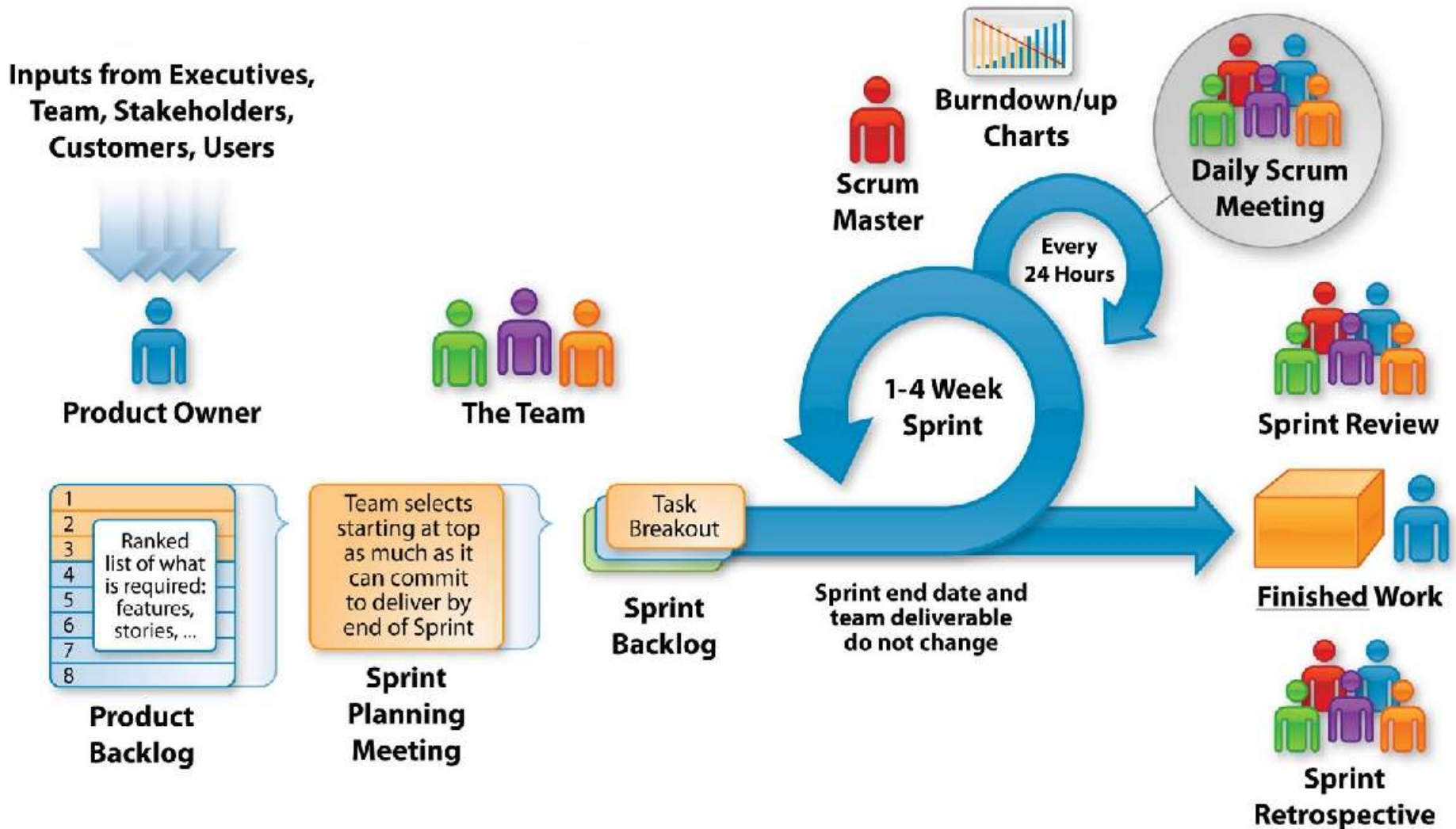
- All unit tests are executed daily
- “Acceptance tests” are defined by the customer and executed to assess customer visible functionality

1. Agility
2. Common Agile Practices
3. Extreme Programming
4. Scrum 
5. Kanban

Scrum

∞ 3.4 ∞

Scrum Process



Scrum Process

Scrum Events

- Sprint Planning Meeting
- Daily Scrum
- Sprint Review
- Sprint Retrospective

The Scrum Team

- The Product Owner
- The Development Team
- The Scrum Master

Scrum Artifacts

- Product Backlog
- Sprint Backlog
- Increment

Scrum Master

Scrum Master performs following duties

☞ Service to the Product Owner

- Finding techniques for effective Product Backlog management;
- Clearly communicating vision, goals, and Product Backlog items to the Development Team;
- Understanding long-term product planning in an empirical environment;

☞ Service to the Development Team

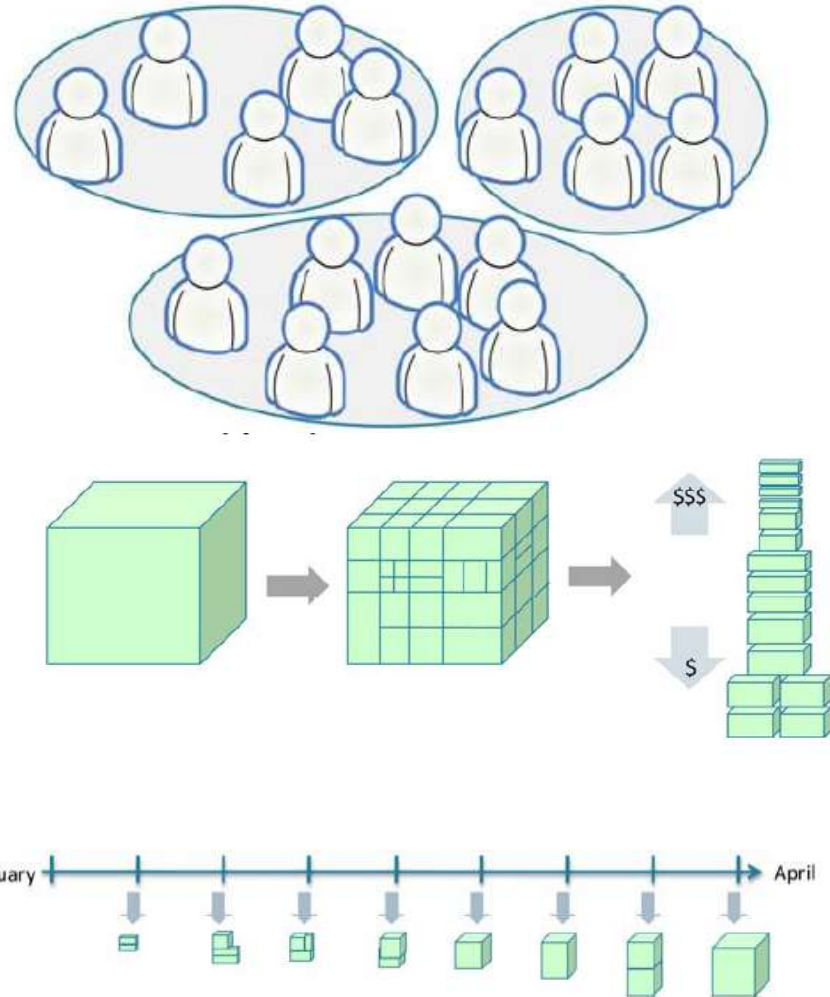
- Coaching the Development Team in self-organization and cross-functionality
- Removing impediments to the Development Team's progress;
- Facilitating Scrum events as requested or needed


☞ Service to the Organization

- Leading and coaching the organization in its Scrum adoption and implementations
- Causing change that increases the productivity of the Scrum Team
- Working with other Scrum Masters

Scrum in a Nutshell

- Split your organization into small, cross-functional, self-organizing teams.
- Split your work into a list of small, concrete deliverables. Sort the list by priority and estimate the relative effort of each item.
- Split time into short fixed-length iterations (usually 1 – 4 weeks), with potentially shippable code demonstrated after each iteration.
- Optimize the release plan and update priorities in collaboration with the customer, based on insights gained by inspecting the release after each iteration.
- Optimize the process by having a retrospective after each iteration.



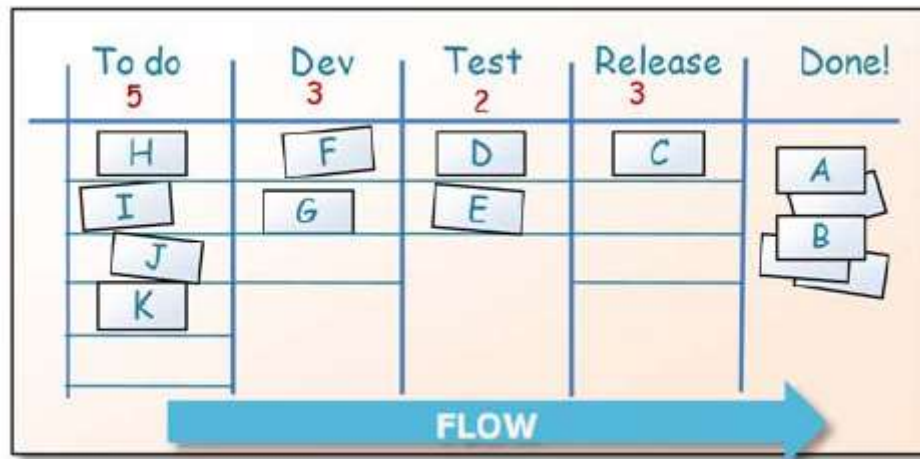
1. Agility
2. Common Agile Practices
3. Extreme Programming
4. Scrum
5. Kanban 

Kanban

∞ 3.4 ∞

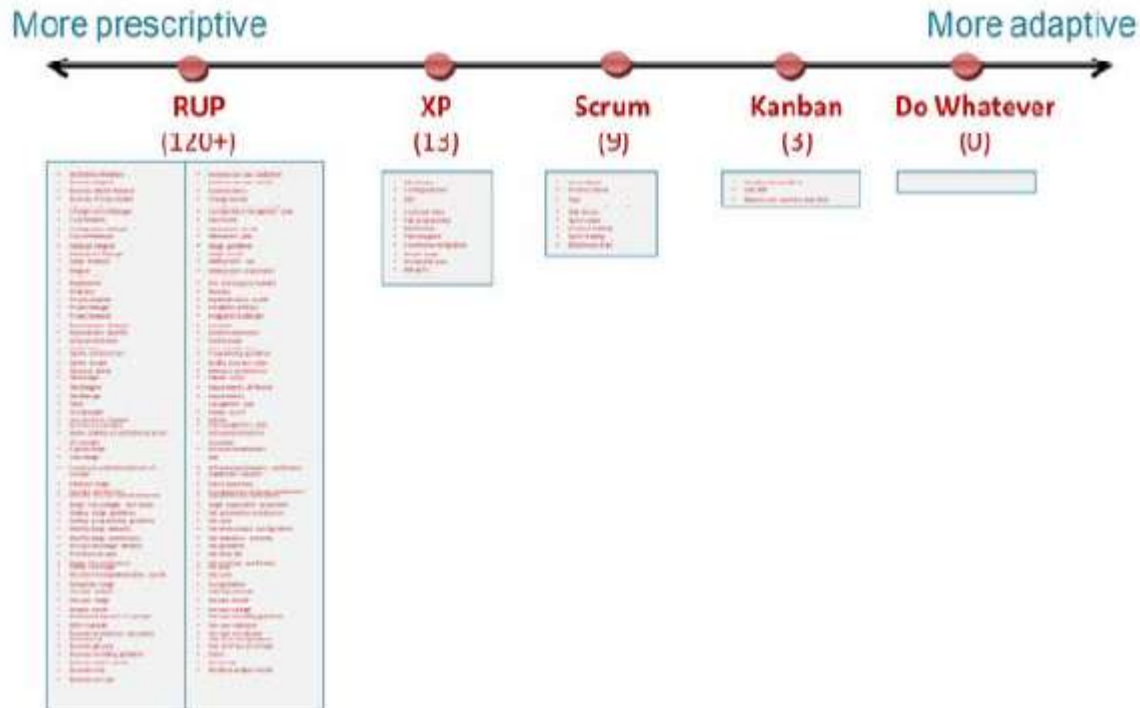
Kanban in a Nutshell

- Visualize the workflow
- Limit Work In Progress (WIP) – assign explicit limits to how many items may be in progress at each workflow state.
- Measure the lead time (average time to complete one item, sometimes called “cycle time”), optimize the process to make lead time as small and predictable as possible.



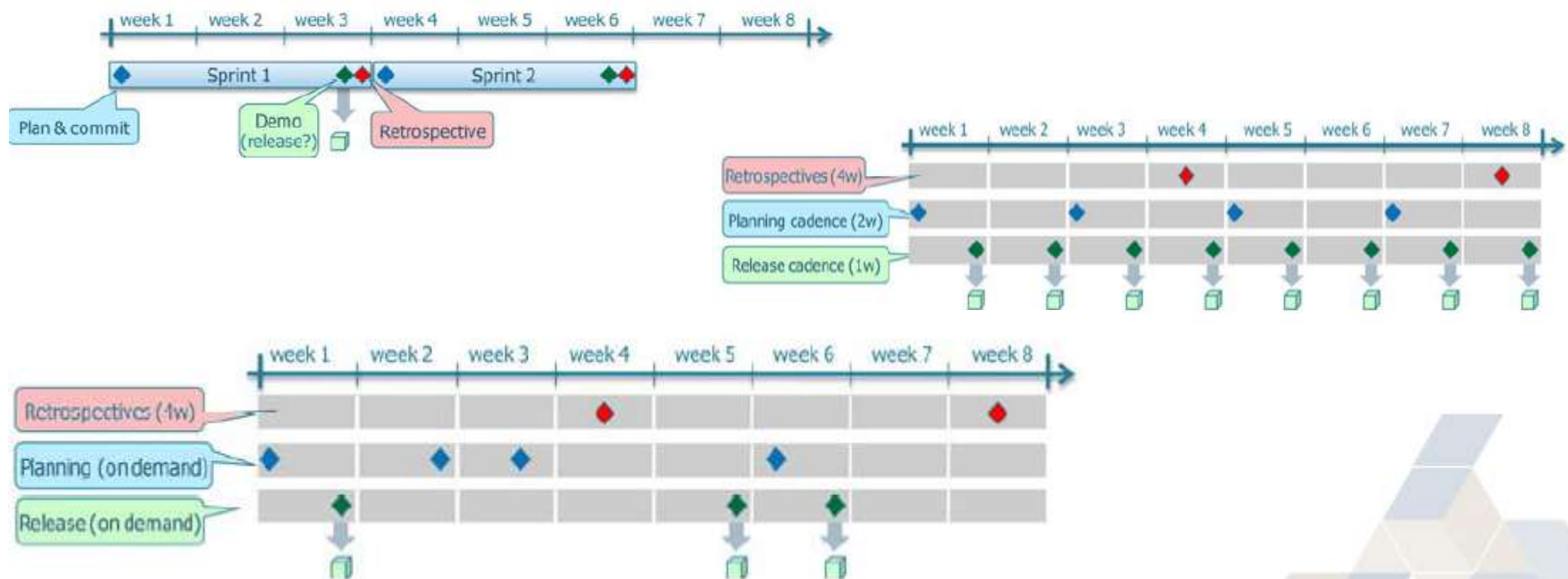
Kanban in a Nutshell

- Visualize the workflow
- Limit Work In Progress (WIP) – assign explicit limits to how many items may be in progress at each workflow state.
- Measure the lead time (average time to complete one item, sometimes called “cycle time”), optimize the process to make lead time as small and predictable as possible.



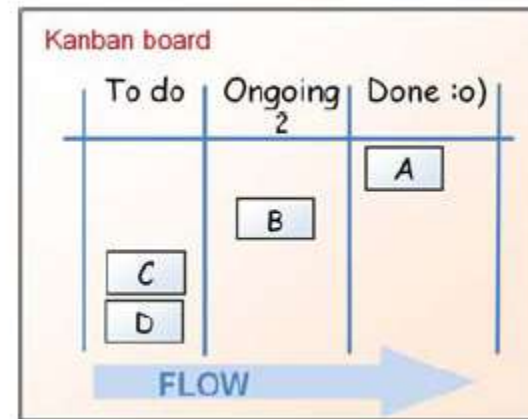
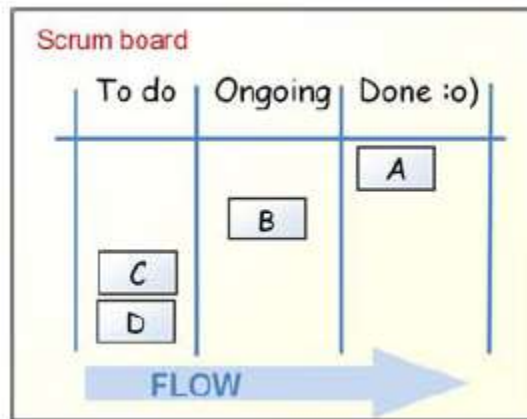
Scrum vs Kanban

- Scrum is more prescriptive
- Kanban doesn't prescribe any roles at all.
- In Kanban time-boxed iterations are not prescribed. You can go using
 - Single Scrum iterations
 - Multiple periodic cadences
 - Multiple Event-driven cadences. E.g. We trigger a release whenever there is a set of Minimum Marketable Features (MMFs) ready for release

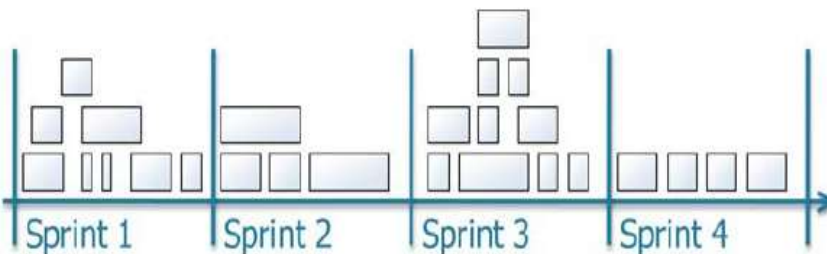


Scrum vs Kanban

- ☞ Kanban limits WIP per workflow state, Scrum limits WIP per iteration.



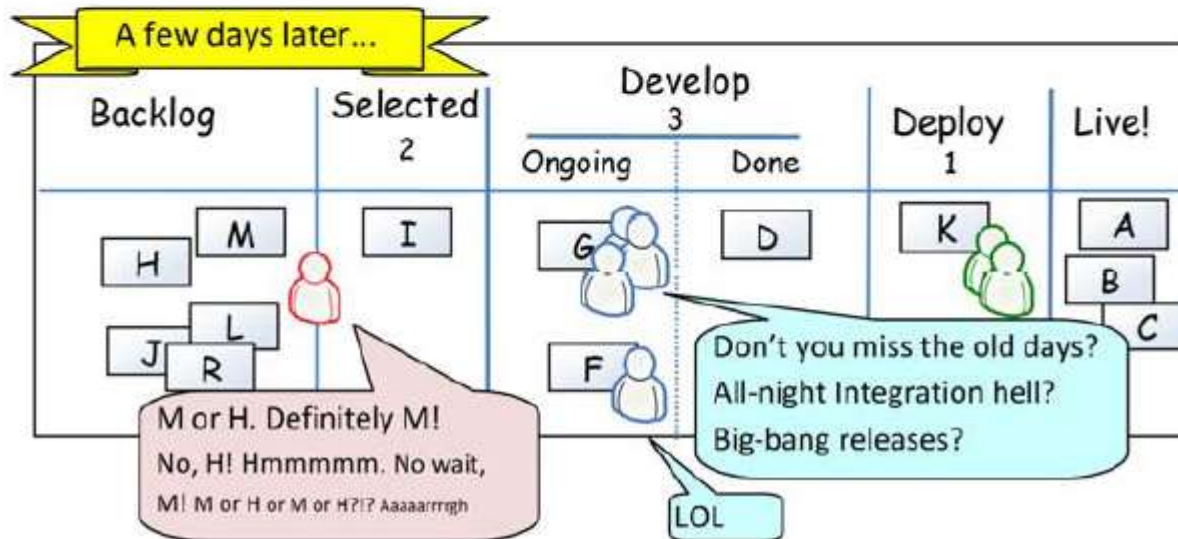
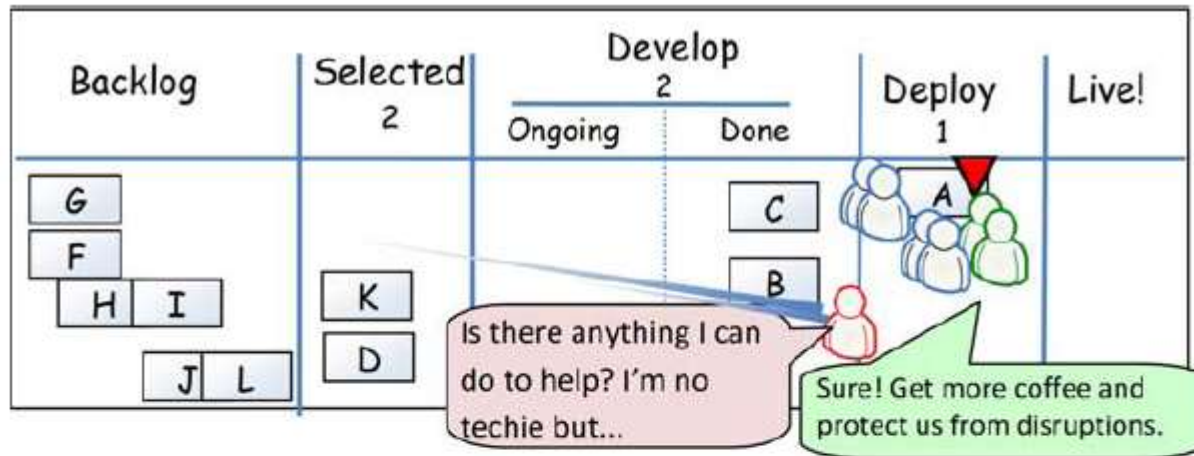
- ☞ Scrum board is reset between each sprint
- ☞ Scrum prescribes cross-functional teams, Kanban boards are cross-team
- ☞ Scrum backlog items must fit in a sprint



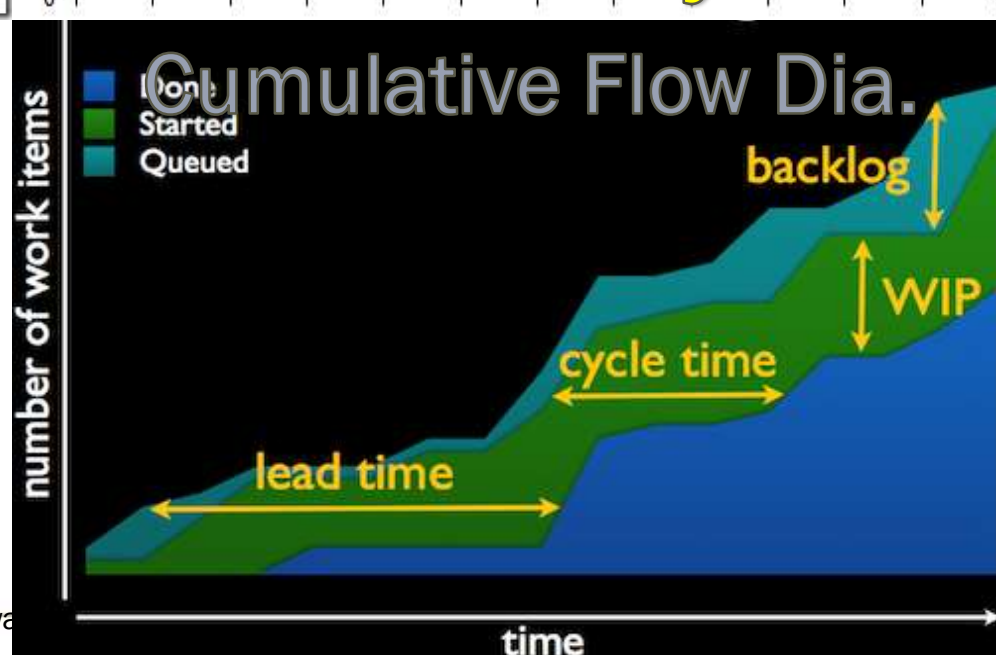
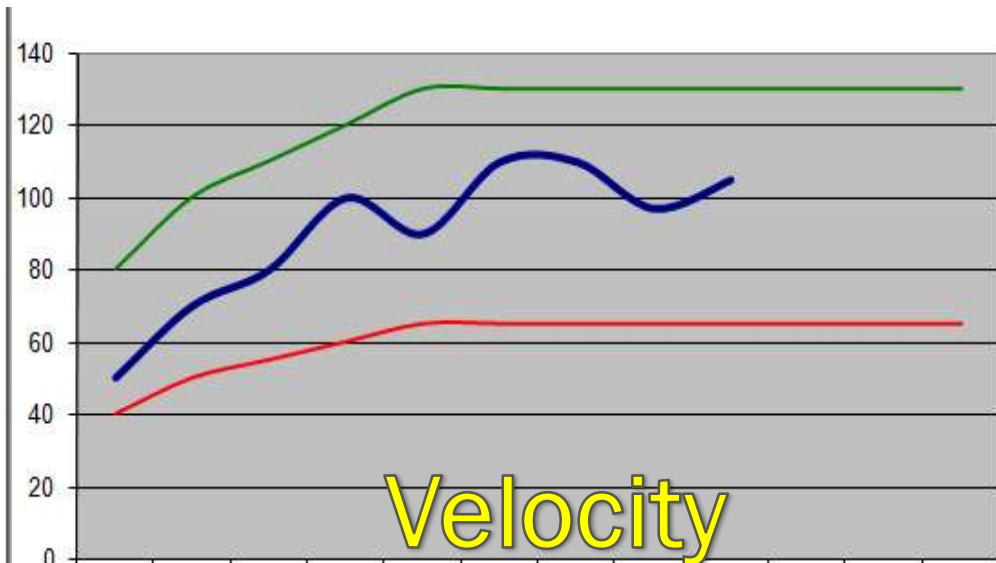
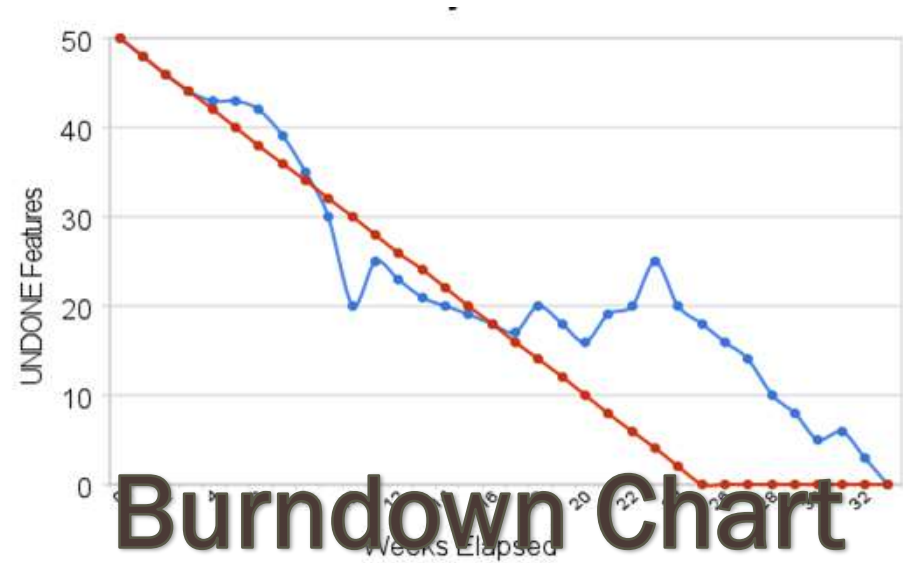
Scrum – Kanban Similarities

- Both use a pull based system
- Both allow working on multiple products simultaneously
- Both are empirical

One Day in Kanban



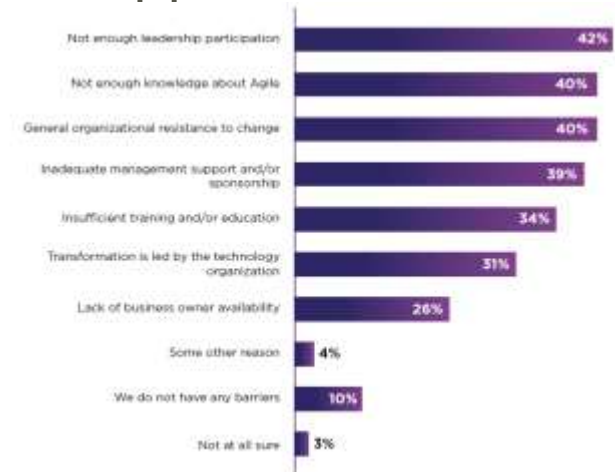
Scrum-Agile Metrics



Problems with agile methods

- ✎ It can be difficult to keep the interest of customers who are involved in the process.
- ✎ Team members may be unsuited to the intense involvement that characterizes agile methods.
- ✎ Prioritizing changes can be difficult where there are multiple stakeholders.
- ✎ Maintaining simplicity requires extra work.
- ✎ Contracts may be a problem as with other approaches to iterative development.

What is not working well with Agile?
From State of Agile Report 2022



Agile Methods Applicability

- ❧ Product development where a software company is developing a small or medium-sized product for sale.
- ❧ Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are not a lot of external rules and regulations that affect the software.
- ❧ Because of their focus on small, tightly-integrated teams, there are problems in scaling agile methods to large systems.

Agile vs Plan-Driven Methods

Plan-driven development

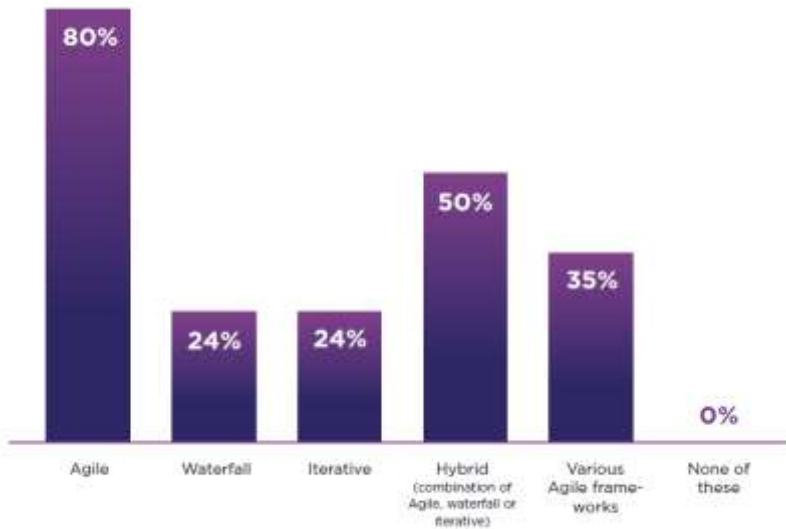
- ✎ A plan-driven approach to software engineering is based around separate development stages with the outputs to be produced at each of these stages planned in advance.
- ✎ Not necessarily waterfall model – plan-driven, incremental development is possible
- ✎ Iteration occurs within activities.

Agile development

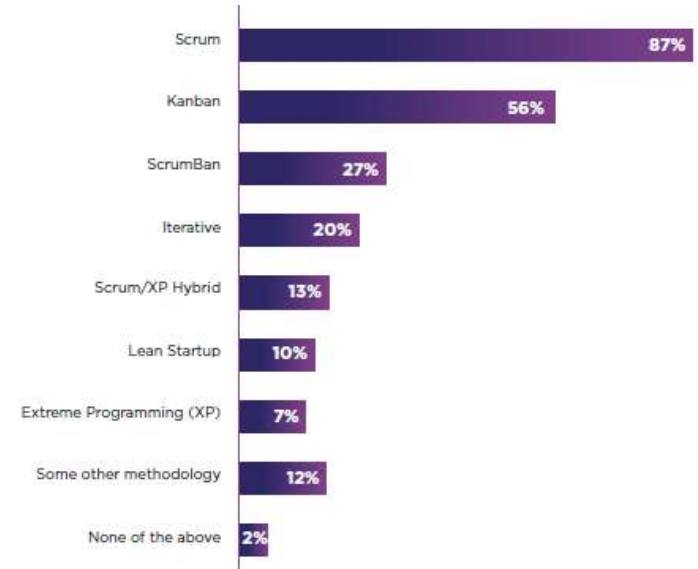
- ✎ Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process

Agile vs Plan-Driven Methods

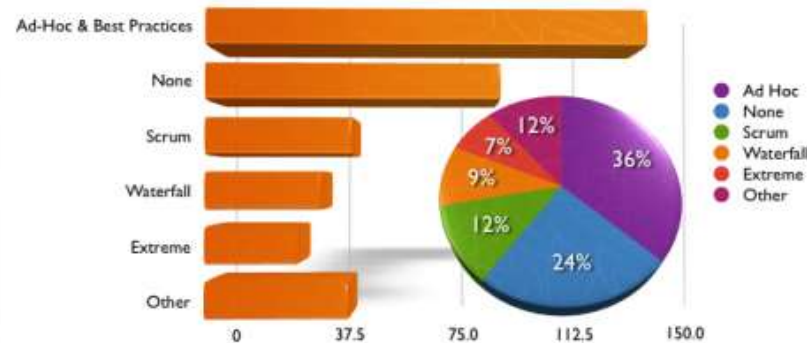
Agile	Plan-driven	Formal Methods
Expert Developers	Junior Developers	Expert Developers
Frequent Changes	Rare Changes	Constrained Changes
Few Developers	Large Groups Of Developers	Modellable Requirements
Chaotic Work Environment	Tidy Work Environment	High Quality Product
Low Critical Systems	High Critical Systems	Very High Critical Systems



From State of Agile Survey 2022



Which Development Methodology?



Wrap-up

- ✎ Different life-cycle models have been presented
 - Each with its own strengths and weaknesses
- ✎ Criteria for deciding on a model include:
 - The organization
 - Its management
 - The skills of the employees
 - The nature of the product
- ✎ Best suggestion
 - “Mix-and-match” life-cycle model

Wrap-up

This week we present

- ✎ The concept of agility
- ✎ Agile Practices
- ✎ Most frequently used agile methodologies
 - XP
 - Scrum
 - Kanban
- ✎ Best suggestion
 - Select practices and methodology based on organizational characteristics and experience level.