Process Models

Chapter 2-3

CSCI-P465/565

Hamidreza Bahramian
Indiana University Bloomington

A generic process model

Process flow

Prescriptive Models (plan-driven)

- Prescriptive process models advocate an orderly approach to software engineering

  That leads to a few questions …
- If prescriptive process models strive for structure and order, are they inappropriate for a software world that thrives on change?
- Yet, if we reject traditional process models (and the order they imply) and replace them with something less structured, do we make it impossible to achieve coordination and coherence in software work?

Framework Activities

- Communication
- Planning
- Modeling
  - Analysis of requirements
  - Design
- Construction
  - Code generation
  - Testing
- Deployment

Umbrella Activities

- Software project tracking and control
- Risk management
- Software quality assurance
- Technical reviews
- Measurement
- Software configuration management
- Reusability management
- Work product preparation and production

Incremental versus Iterative

- **Design, Implementation & Test incrementally**
- **Iteratively improve the design**
- ***"If you were writing an essay under the <u>Incremental</u> Model, you'd attempt to write it perfectly from start to finish one sentence at time. If you wrote it under the <u>Iterative</u> Model, you'd write a quick rough draft first and then work to improve it through a set of revision phases".***

Prescriptive Models

- The Waterfall Model
- Prototyping Process Model
- Evolutionary Process Model
- Unified Process Model

The Waterfall Model

The Waterfall Model

- It is often difficult for the customer to state all the requirements explicitly at the beginning of most projects.
- A working version of the software will not be available until late the project time span.
- Major blunders may not be detected until the working program is reviewed.

Prototyping Process Model

- This paradigm assists you and other stakeholders to better understand what is to be built when requirements are fuzzy.

Prototyping Process Model

- The paradigm begins with communication. You meet with other stakeholders to define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory.
- A prototyping iteration is planned quickly, and modeling (in the form of a “quick design”) occurs. A quick design focuses on a representation of those aspects of the software that will be visible to end users (e.g., human interface layout or output display formats).
- The quick design leads to the construction of a prototype.
- The prototype is deployed and evaluated by stakeholders, who provide feedback that is used to further refine requirements.
- Iteration occurs as the prototype is tuned to satisfy the needs of various stakeholders, while at the same time enabling you to better understand what needs to be done.

Prototyping Process Model

Prototyping Process Model

- Ideally, the prototype serves as e mechanism for identifying software requirements.
- Both stakeholders and software engineers like the prototyping paradigm. Users get a feel for the actual system, and developers get to build something immediately.

Prototyping Process Model

- But prototyping can be problematic for two reasons:
  - Stakeholders see what appears to be a working version of the software. They may be unaware that the prototype architecture (program structure) is also evolving. This means that the developers may not have considered the overall software quality or long-term maintainability.
  - As a software engineer, you may be tempted to make implementation compromises to get a prototype working quickly. If you are not careful, these less-than-ideal choices have now become an integral part of the evolving system.

Prototyping Process Model

- The key is to define the rules of the game at the beginning; that is, all stakeholders should agree that the prototype is built in part to serve as a mechanism for defining requirements.
- It is often desirable to design a prototype so it can be evolved into the final product. The reality is developers may need to discard (at least in part) a prototype to better meet the customer’s evolving needs.

The Spiral Process Model

The Spiral Process Model

- Business and product requirements often change as development proceeds, making a straight-line path to an end product unrealistic.
- Tight market deadlines may make completion of a comprehensive software product impossible.
- It might be possible to create a limited version of a product to meet competitive or business pressure and release a refined version once all system features are better understood.
- In a situation like this you need a process model that has been explicitly designed to accommodate a product that grows and changes.

The Spiral Process Model

- Originally proposed by Barry Boehm [Boe88], the spiral model is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model.
- It provides the potential for rapid development of increasingly more complete versions of the software.
- 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 Process Model

- The first circuit around the spiral might result in the development of a product specification.
- subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software.
- Each pass through the planning region results in adjustments to the project plan. Cost and schedule are adjusted based on feedback derived from the customer after delivery.
- In addition, the project manager adjusts the planned number of iterations required to complete the software.
- Unlike other process models that end when software is delivered, the spiral model can be adapted to apply throughout the life of the computer software.

The Spiral Process Model - Issues

- It may be difficult to convince customers (particularly in contract situations) that the evolutionary approach is controllable.
- It demands considerable risk assessment expertise and if a major risk is not uncovered and managed, problems will undoubtedly occur.
- In many cases, time to market is the most important management requirement. If a market window is missed, the software project itself may be meaningless.

Unified Process Model

UP Phases

UP Phases
Inception
Elaboration
Construction
Transition
Production
Workflows
Requirements
Analysis
Design
Implementation
Test
Support
Iterations
#1
#2
#n-1
#n

Unified Process Model - Inception

- The inception phase of the UP is where customer communication and planning takes place.
- Fundamental business requirements are described through a set of preliminary use cases (Chapter 7) that describe which features and functions each major class of users desires that will become realized in the software architecture.
- Planning identifies resources, assesses major risks, and defines a preliminary schedule for the software increments.

Unified Process Model - Elaboration

- The elaboration phase encompasses the planning and modeling activities of the generic process model.
- Elaboration refines and expands the preliminary use cases and creates an architectural baseline that includes five different views of the software—the use case model, the analysis model, the design model, the implementation model, and the deployment model.
- Modifications to the plan are often made at this time.

Unified Process Model - Construction

- The construction phase of the UP is identical to the construction activity defined for the generic software process.
- All necessary and required features and functions for the software increment (i.e., the release) are then implemented in source code.
- Unit tests, component assembly and integration testing are conducted.
- Use cases are used to derive a suite of acceptance tests that are executed prior to the initiation of the next UP phase.

Unified Process Model - Transition

- The transition phase of the UP encompasses the latter stages of the generic construction activity and the first part of the generic deployment (delivery and feedback) activity.
- Software and supporting documentation is given to end users for beta testing, and user feedback reports both defects and necessary changes.
- At the conclusion of the transition phase, the software increment becomes a usable software release.

Unified Process Model - Production

- The production phase of the UP coincides with the deployment activity of the generic process.
- During this phase, the ongoing use of the software is monitored, support for the operating environment (infrastructure) is provided, and defect reports and requests for changes are submitted and evaluated.

Comparing Process Models

Waterfall pros	It is easy to understand and plan. It works for well-understood small projects. Analysis and testing are straightfoward.

Waterfall cons	It does not accommodate change well. Testing occurs late in the process. Customer approval is at the end.

Comparing Process Models

Prototyping pros	There is a reduced impact of requirement changes. The customer is involved early and often. It works well for small projects. There is reduced likelihood of product rejection.

Prototyping cons	Customer involvement may cause delays. There may be a temptation to "ship" a prototype. Work is lost in a throwaway prototype. It is hard to plan and manage.

Comparing Process Models

Spiral pros	There is continuous customer involvement. Development risks are managed. It is suitable for large, complex projects. It works well for extensible products.

Spiral cons	Risk analysis failures can doom the project. The project may be hard to manage. It requires an expert development team.

Comparing Process Models

Unified Process pros	Quality documentation is emphasized. There is continuous customer involvement. It accommodates requirements changes. It works well for maintenance projects.

Unified Process cons	Use cases are not always precise. It has tricky software increment integration. Overlapping phases can cause problems. It requires an expert development team.

Summary

- A generic process model for software engineering encompasses a set of framework and umbrella activities, actions, and work tasks.
- Prescriptive process models have been applied for many years in an effort to bring order and structure to software development.
- Each of these models suggests a somewhat different process flow, but all perform the same set of generic framework activities: communication, planning, modeling, construction, and deployment.

Summary

- Sequential process models, such as the waterfall model suggest a linear process flow that is often inconsistent with modern realities (e.g., continuous change, evolving systems, tight time lines) in the software world. They do, however, have applicability in situations where requirements are well defined and stable.
- Incremental process models are iterative in nature and produce working versions of software quite rapidly.

Summary

- Prototyping and the spiral model, produce incremental work products (or working versions of the software) quickly.
- The Unified Process is a “use case driven, architecture-centric, iterative and incremental” software process designed as a framework for UML methods and tools.

Process Models

Chapter 2-3

CSCI-P465/565

Hamidreza Bahramian
Indiana University Bloomington

Outline

- Last time:
  - Prescriptive models: Waterfall, Prototyping, Spiral and Unified Model
- Today:
  - Agile: Scrum, XP, Kanban and DevOps

Prescriptive Models

- The Waterfall Model
- Prototyping Process Model
- Evolutionary Process Model
- Unified Process Model
  - Rational Unified Process (RUP)

RUP - some of the artifacts

- Inception:
  - Vision
  - Business case
  - Risk list
  - Software plan
  - Iteration plan
  - .Main use-cases
  - Configuration management environment

RUP - some of the artifacts

- Elaboration:
  - Prototypes
  - Risk list
  - Development case
  - Tools
  - Software architecture document
  - Design model
  - Data model
  - Implementation model
  - Vision
  - Use case model
  - Tests suite

RUP

	Inception	Elaboration	Construction	Transition
Effort	10%	30%	55%	5%
Elapsed time	20%	35%	40%	5%

Agility and Process

- In 2001, a group of noted software developers, writers, and consultants [Bec01] signed the **“Manifesto for Agile Software Development”** in which they argued in favor of
  - individuals and interactions over processes and tools
  - working software over comprehensive documentation
  - customer collaboration over contract negotiation
  - responding to change over following a plan.

Agile Software Development

- Effective (rapid and adaptive) response to change
- Effective communication among all stakeholders
- Adopting the customer as a part of the team
- Emphasizes rapid delivery of operational software
- Deemphasizes the importance of the intermediate work products (not always a good thing)

Agility and the Cost of Change

An Agile Process

- Key assumptions about the majority of software projects:
  - It is difficult to predict in advance which software requirements will persist and which will change. It is equally difficult to predict how customer priorities will change as the project proceeds.
  - It is difficult to predict how much design is necessary before construction is used to prove the design.
  - Analysis, design, construction, and testing are not as predictable (from a planning point of view) as we might like.

An Agile Process

- Given these three assumptions, an important question arises: How do we create a process that can manage unpredictability?
  - The answer: adaptability (to rapidly changing project and technical conditions)
  - But continual adaptation without forward progress accomplishes little.
  - Therefore, an agile software process must **adapt incrementally**.

An Agile Process

- How to accomplish incremental adaptation?
  - an agile team requires customer feedback (so that the appropriate adaptations can be made).
  - An effective catalyst for customer feedback is an operational prototype or a portion of an operational system.
  - Software increments (executable prototypes or portions of an operational system) must be delivered in short time periods so that adaptation keeps pace with change (unpredictability).

An Agile Process

- Is driven by customer descriptions of what is required (scenarios)
- Recognizes that plans are short-lived
- Develops software iteratively with a heavy emphasis on construction activities in fixed time
- Delivers multiple ‘software increments’
- Adapts as changes occur

Agility Principles - I

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

Agility Principles - II

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

Tao Te Ching (Daodejing)

Chapter 23

1. Abstaining from speech marks him who is obeying the spontaneity of his nature. A violent wind does not last for a whole morning; a sudden rain does not last for the whole day. To whom is it that these (two) things are owing? To Heaven and Earth. If Heaven and Earth cannot make such (spasmodic) actings last long, how much less can man!

2. Therefore when one is making the Tao his business, those who are also pursuing it, agree with him in it, and those who are making the manifestation of its course their object agree with him in that; while even those who are failing in both these things agree with him where they fail.

3. Hence, those with whom he agrees as to the Tao have the happiness of attaining to it; those with whom he agrees as to its manifestation have the happiness of attaining to it; and those with whom he agrees in their failure have also the happiness of attaining (to the Tao). (But) when there is not faith sufficient (on his part), a want of faith (in him) ensues (on the part of the others).

The politics of Agile Development

- The debate:
  - Pro-agility camp (“agilists”): “Traditional methodologists are a bunch of stick-in-the-muds who would rather produce flawless documentation than a working system that meets business needs.”
  - The traditional software engineering camp: “Lightweight, ‘agile’ methodologists are a bunch of glorified hackers who are going to be in for a big surprise when they try to scale up their toys into enterprise-wide software.”

The politics of Agile Development

- No one is against agility. The real question is: What is the best way to achieve it?
  - Keep in mind that working software is important, but don’t forget that it must also exhibit a variety of quality attributes including reliability, usability, and maintainability.
- The bottom line:
  - **There is much that can be gained by considering the best of both schools and virtually nothing to be gained by denigrating either approach.**

Scrum

- Scrum is a popular agile software development method.
- The name is derived from an activity that occurs during a rugby match.
- It was conceived by Jeff Sutherland and his development team in the early 1990s. Further development on the Scrum methods was performed by Schwaber and Beedle [Sch01b].
- Framework activities: requirements, analysis, design, evolution, and delivery.
- Within each framework activity, work tasks take place in a relatively short time-boxed period called a sprint.
- The work conducted within a sprint is adapted to the problem at hand and is defined and often modified in real time by the Scrum team.

Scrum

Product Backlog:
Prioritized product features desired by the customer
every 24 hours
Scrum:
15-minute daily meeting.
30 days
Sprint Backlog:
Feature(s) assigned to sprint
Backlog items expanded by team
New functionality is demonstrated at end of sprint

Scrum Teams and Artifacts

- The Scrum team is a self-organizing interdisciplinary team consisting of a **product owner**, a **Scrum master**, and a small (three to seven people) **development team**.
- The principle Scrum artifacts are **the product backlog**, **the sprint backlog**, and **the code increment**.

Scrum Teams and Artifacts

- Development proceeds by breaking the project into a series of incremental prototype development periods 2 to 4 weeks in length called **sprints**.
- The product backlog is a prioritized list of product requirements or features that provide business value for the customer.
- The product owner orders the items in the product backlog to meet the most important goals of all stakeholders.
- The product owner is the only person who decides whether to end a sprint prematurely or extend the sprint if the increment is not accepted.
- The sprint backlog is the subset of product backlog items selected by the product team to be completed as the code increment during the current active sprint.
- The increment is the union of all product backlog items completed in previous sprints and all backlog items to be completed in the current sprints.
- The development team creates a plan for delivering a software increment containing the selected features.
- No new features can be added to the sprint backlog unless the sprint is cancelled and restarted.

Scrum Teams and Artifacts

- Scrum master:
  - serves as facilitator to all members of the Scrum team.
  - runs the daily Scrum meeting
  - is responsible for removing obstacles identified by team members during the meeting.
  - coaches the development team members to help each other complete sprint tasks.
  - helps the product owner find techniques for managing the product backlog items and helps ensure that backlog items are stated in clear and concise terms.

Sprint Planning Meeting

- Prior to beginning:
  - Any development team works with the product owner and all other stakeholders to develop the items in the product backlog.
  - The product owner and the development team rank the items in the product backlog by the importance of the owner's business needs and the complexity of the software engineering tasks (programming and testing) required to complete each of them.

Sprint Planning Meeting

- Prior to starting each sprint:
  - The product owner states her development goal for the increment to be completed in the upcoming sprint.
  - The Scrum master and the development team select the items to move from to the sprint backlog.
  - The development team determines what can be delivered in the increment within the constraints of the time-box allocated for the sprint and, with the Scrum master, what work will be needed to deliver the increment.
  - The development team decides which roles are needed and how they will need to be filled.

Daily Scrum Meeting

- The daily Scrum meeting is a 15-minute event scheduled at the start of each workday to allow team members to synchronize their activities and make plans for the next 24 hours.
- Attendants: the Scum master and the development team and occasionally the product owner

Daily Scrum Meeting

- 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?
- The Scrum master leads the meeting and assesses the responses from each person.
- The Scrum meeting helps the team to **uncover potential problems as early as possible**.
- **It is the Scrum master's task to clear obstacles presented** before the next Scrum meeting if possible.

Daily Scrum Meeting

- These are not problem-solving meetings, those occur off-line and only involve the affected parties.
- Some teams use these meetings to declare sprint backlog items complete or done.
- When the team considers all sprint backlog items complete, the team may decide to schedule a demo and review of the completed increment with the product owner.

Sprint Review Meeting

- The sprint review occurs at the end of the sprint when the development team has judged the increment complete.
- The sprint review is often time-boxed as a 4-hour meeting for a 4-week sprint.
- The Scrum master, the development team, the product owner, and selected stakeholders attend this review.
- The primary activity is a demo of the software increment completed during the sprint.
- The product owner may accept the increment as complete or not. If it is not accepted, the product owner and the stakeholders provide feedback to allow a new round of sprint planning to take place.

Sprint Retrospective

- Before beginning another sprint planning meeting, the Scrum master will schedule a 3-hour meeting (for a 4-week sprint) with the development team called a sprint retrospective.
- During this meeting the team discusses:
  - What went well in the sprint?
  - What could be improved?
  - What the team will commit to improving in the next sprint?
- The Scrum master leads the meeting and encourages the team to improve its development practices to become more effective for the next sprint.

Group work 2

Problem ID	Problem
1	“WE JUST WASTED HOURS GRINDING THROUGH SPAGHETTI CODE TO FIND THAT BUG!”
2	“OK, WE’VE GONE OVER THE USER STORIES. NOW LET’S FIGURE OUT HOW THEY FIT TOGETHER SO WE CAN PLAN OUT THE NEXT FEW WEEKS OF WORK.”
3	“WE ALWAYS SEEM TO RUN INTO THE SAME KIND OF PROBLEMS OVER AND OVER AGAIN WITH EVERY RELEASE.”
4	“I JUST DEMOED A NEW FEATURE TO ONE OF OUR VIDEO STREAMING USERS. SHE SAID IT WON’T ACTUALLY FIX THE PROBLEM IT’S SUPPOSED TO ADDRESS FOR HER.”
5	“I THOUGHT WE’D BE DONE UPDATING THE SONG DATABASE CODE BY FRIDAY. NOW YOU’RE TELLING ME THAT IT’LL BE THREE MORE WEEKS?”

Group work 2

Practice ID	Practice
1	A **retrospective** is a meeting in which everyone talks about how the last part of the project went, and talks about what lessons can be learned.
2	A **user story** is a way to express one very specific need that a user has, usually written out as a few sentences on a sticky note or index card.
3	A **task board** is an agile planning tool in which user stories are attached to a board and categorized into columns based on their status.
4	A **burndown chart** is a line chart, updated daily, that tracks the amount of work left on the project, “burning” down to zero when the work is done.
5	Developers fix code problems by constantly **refactoring** their code, or improving the code structure without changing its behavior. It’s OK if you haven’t run across these practices yet.

Process Models

Chapter 3

CSCI-P465/565

Hamidreza Bahramian
Indiana University Bloomington

Outline

- Last time:
  - Agile, Scrum, kanban
- Today:
  - Lean principles
  - Kanban, XP, DevOP

Scrum, the first day!

Kate thinks Mike and his team are being irrational, but maybe they have a point. What do you think?

Scrum, the first day!

In Kate's less-than-effective daily standup, each person tunes out everyone else while they're waiting to give their updates, then says as little as possible when it's time to speak. She still gets some useful information, but at the cost of conflict and boredom—and nobody in the meeting is collaborating.

Scrum, the first day!

A better mindset makes the practice better

A better mindset makes the practice better

The Agile way

- The Agile Manifesto was created in 2001 by people who came together to find common ground between different “lightweight” methods, methodologies, and approaches to building software.
- The Agile Manifesto has four values that help agile teams get into the right mindset.
- Agile teams **value processes and tools** because they help the team get organized and work effectively.
- But they **value people and interactions more** because teams work best when you pay attention to the human element.

The Agile way

- Agile teams **value comprehensive documentation** because it's an effective way to communicate complex requirements and ideas.
- But they **value working software more** because it's the most effective way to communicate progress and get feedback from users.
- Agile teams **value contract negotiation** because sometimes it's the only way to work effectively in an office culture where mistakes are punished.
- But they **value customer collaboration more** because it is much more effective for building software than having a legalistic or antagonistic customer relationship.
- Agile teams **value following a plan** because without planning, complex software projects go off the rails.
- But they **value responding to change more** because the team that works the wrong plan ends up building the wrong software.

Practices

DICTIONARY DEFINITION

timeboxed, adjective

setting a hard deadline for an activity to be completed, and adjusting the scope of that activity to meet the deadline

The team couldn't fit all of the requested features into the current ***timeboxed*** *iteration, so they concentrated on the most valuable ones*

Practices

Backlog: a great way to manage changing requirements
The backlog is a list of features waiting to be built. Any feature that hasn't been included in an iteration yet is fair game for the users and the product owner to change.
Backlog 1
Backlog 2
Backlog 3
Backlog 4
Iteration #1
Iteration #2
Iteration #3
When the team plans each iteration, they pull features to include off of the backlog.

Judgment call

Working software is the primary measure of progress.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

Judgment call

Working software is the primary measure of progress.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

Judgment call

OK, WHICH OF YOU IDIOTS WROTE THIS BUGGY BLOCK OF SPAGHETTI CODE? IT'S YOUR FAULT THAT WE'RE BEHIND.

COMPATIBLE

INCOMPATIBLE

Working software is the primary measure of progress.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

Judgment call

Working software is the primary measure of progress.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

Other Agile Frameworks

- Extreme Programming(XP)
- Kanban
- DevOps

XP

simple design
CRC cards
spike solutions
prototypes
user stories
values
acceptance test criteria
iteration plan
design
planning
coding
refactoring
testing
pair programming
Release
software increment
project velocity computed
acceptance testing
Unit test
continuous integration

XP - the planning activity

- The planning activity (also called the planning game) begins with a requirements activity called listening.
- 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 user story (described in Chapter 7) 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.

XP - the planning activity

- Customers and developers work together to decide how to group stories into the next release.
- The XP team orders the stories that will be developed in one of three ways:
  - all stories will be implemented immediately (within a few weeks)
  - the stories with highest value will be moved up in the schedule and implemented first
  - or the riskiest stories will be moved up in the schedule and implemented first.
- After the first project release has been delivered, the XP team computes **project velocity** (the number of customer stories implemented during the first release).
- Project velocity can then be used to help estimate delivery dates and schedule for subsequent releases. The XP team modifies its plans accordingly.

XP - the design activity

- XP design rigorously follows the KIS (keep it simple) principle.
  - The design of extra functionality (because the developer assumes it will be required later) is discouraged.
- XP encourages the use of CRC (class-responsibility-collaborator) cards (Chapter 8) as an effective mechanism for thinking about the software in an object-oriented context.
- CRC cards are the only design work product produced as part of the XP process.
- A central notion in XP is that design occurs both before and after coding commences.
  - In fact, the construction activity itself will provide the XP team with guidance on how to improve the design.

XP - the coding activity

- After user stories are developed and preliminary design work is done, the team does not move to code, but rather **develops a series of unit tests** that will exercise each of the stories that is to be included in the current release.
- “pair programming”: XP recommends that two people work together at one computer to create code for a story.
  - real-time problem solving (two heads are often better than one)
  - real-time quality assurance (the code is reviewed as it is created)

XP - the testing activity

- The unit tests that are created should be implemented using a framework that enables them to be automated (hence, they can be executed easily and repeatedly).
- This encourages implementing a regression testing strategy (Chapter 20) whenever code is modified (which is often, given the XP refactoring philosophy).
- 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.

Lean

- Lean thinking was born out of studying the rise of Toyota Motor Company from a bankrupt Japanese automaker in the early 1950s to today's dominant global player.
- Lean is a mindset not a methodology
- Lean teams look at the way they're working today, and figure out where they are habitually running into problems. Then they make changes to improve the way they work.

Lean principles

- It takes a lot of work to build a product. But teams often do more work than they need to: they'll add unneeded features, waste time trying to multitask, and spend time just sitting around waiting. When a team has a Lean mindset, they try to find all of this waste and work hard to eliminate it from the project, often by removing anything that distracts the team from creating the software their users need.

Lean principles

Amplify learning

- This principle is all about learning from what you do and using feedback to keep improving. As you make changes in the way your team works, observe what those changes do and then use your observations to decide what to change next.

Lean principles

Decide as late as possible

- Make every important decision for your project when you have the most information about it—at the last responsible moment. Don’t force yourself to decide things that don’t need to be decided just yet.

Lean principles

Deliver as fast as possible

- Things that delay your project are costly. Keep track of how you work, where you're running into delays, and how those delays impact your team. Set up pull systems, queues, and buffers to even out the work because that will get your product done as quickly and efficiently as possible.

Lean principles

Empower the team

- There’s no better expert in the world on how a team works than a member of that team. This principle is about helping every team member to have access to all of the information they need about the goals of the project and their progress, and about letting the team decide on the most effective way to work.

Lean principles

Build integrity in

- Users best understand the purpose of the software that the team builds and can evaluate the quality of the work when the team builds software that meets the users' needs. If the software you're building is intuitive and does something valuable for your users, it's a lot more likely to be worth the effort everyone is putting into building it.

Lean principles

- Take time to understand how the team is working on the project—and take the right kind of measurements, so that everyone is exposed to all of the information they need to make good decisions about it. When everyone on the team can see what everyone else is doing (and not just their own contributions), the team can collaborate on the best way to get the work done.

Categories of waste

- Partially done work
- Extra processes
- Extra features
- Task switching
- Waiting
- Defects

Judgment call

Task Switching

Extra Features

Partially Done Work

Judgment call

WASTE

WHILE I WAS WAITING FOR THE DEVOPS FOLKS TO DO THEIR WORK, I GOT STARTED ON ANOTHER FEATURE SO I'M NOT SITTING IDLE.

NOT WASTE

Task Switching

Extra Features

Partially Done Work

Judgment call

Task Switching

Extra Features

Partially Done Work

Judgment call

Task Switching

Extra Features

Partially Done Work

Kanban

- Kanban is focused on change management and service delivery.
  - Change management defines the process through which a requested change is integrated into a software-based system.
- Kanban originated at Toyota as a set of industrial engineering practices and was adapted to software development by David Anderson [And16].
- Kanban is based on the Lean mindset.

Kanban - six core practices

- Kanban is focused on change management and service delivery.
  - Visualizing workflow using a Kanban board.
  - Limiting the amount of work in progress (WIP) at any given time.
  - Managing workflow to reduce waste by understanding the current value flow, analyzing places where it is stalled, defining changes, and then implementing the changes.
  - Making process policies explicit (e.g., write down your reasons for selecting items to work on and the criteria used to define “done”).
  - Focusing on continuous improvement by creating feedback loops
  - Make process changes collaboratively and involve all team members and other stakeholders as needed.

Kanban Core Practices

- Visualize Workflow
- Limit WIP
- Manage Flow
- Make Process Policies Explicit
- Implement Feedback Loops
- Improve Collaboratively, Evolve Experimentally

Kanban teams need to see the whole, so the first thing the team does is to look at the way they're currently working and create an accurate visual representation of their workflow.

Once the team has a good view of where most of their work is in the workflow, they can start experimenting with WIP limits to see if it helps them focus and get more done.

The team can start to manage the flow through the system by paying attention to how quickly work is getting through their process.

As the team learns how to work with flow, they set policies for the team to use to guide them with their work. Kanban teams make a point of clearly communicating those policies so they can be evaluated and changed if necessary.

By setting explicit policies, the team builds feedback loops, which are the building blocks that let the team collaborate to continuously improve the process and make it more and more efficient.

Feedback loops are all about testing all of the policies and improvements the team is implementing to measure their effects and make sure they are working.

Kanban - the board

DevOps

- DevOps was created by Patrick DeBois [Kim16a] to combine Development and Operations.
- DevOps attempts to apply agile and lean development principles across the entire software supply chain.

DevOps

- The DevOps approach involves several stages that loop continuously until the desired product exists:
  - Continuous development
  - Continuous testing
  - Continuous integration
  - Continuous deployment
  - Continuous monitoring

DevOps

- Continuous development: Software deliverables are broken down and developed in multiple sprints with the increments delivered to the quality assurance members of the development team for testing
- Continuous testing: Automated testing tools are used to help team members test multiple code increments at the same time to ensure they are free of defects prior to integration.
- Continuous integration: The code pieces with new functionality are added to the existing code and to the run-time environment and then checked to ensure there are no errors after deployment.
- Continuous deployment: At this stage the integrated code is deployed (installed) to the production environment, which might include multiple sites globally that need to be prepared to receive the new functionality.
- Continuous monitoring: Operations staff who are members of the development team help to improve software quality by monitoring its performance in the production environment and proactively looking for possible problems before users find them.

DevOps

Questions

- It sounds like Scrum, XP, Lean/Kanban and DevOps are very different from each other. How can they all be agile?

- Isn’t this all just stuff I know already, only with a new name? Like, Scrum sprints are really just milestones and project phases, right?

Questions

- It sounds like Scrum, XP, Lean/Kanban and DevOps are very different from each other. How can they all be agile?
  - Scrum, XP, and Lean/Kanban focus on very different areas. Scrum is mainly focused on project management: what work is getting done, and making sure that it's in line with what the users and stakeholders need. XP is focused on software development: building high-quality code that's well-designed and easy to maintain. Lean/Kanban is a combination of the Lean mindset and the Kanban method, and teams use it to focus on continually improving the way that they build software. In other words, Scrum, XP, and Lean/Kanban are focused on three different areas of software engineering: project management, design and architecture, and process improvement. So it makes sense that they would have different practices—that's how they differ. In the next chapter you'll learn about what they have in common: shared values and principles that help teams adopt an agile mindset.

Questions

- Isn’t this all just stuff I know already, only with a new name? Like, Scrum sprints are really just milestones and project phases, right?
  - You might think Scrum sprints are the same thing as project phases but in a typical project plan, all of the project phases are planned at the beginning of the project; in Scrum, only the next sprint is planned in detail. This difference can feel very strange to a team accustomed to traditional project management.

Summary

- Many teams that want to adopt agile start with the daily standup, a meeting with the whole team where everyone stands in order to keep it short.
- Agile is a set of methods and methodologies, but it's also a mindset, or an attitude that's shared by everyone on the team.
- The daily standup meeting is much more effective when everyone on the team has the right mindset—everyone listens to each other, and they all work together to make sure the project is on track.
- Every agile methodology comes with a set of values to help the team get into the most effective mindset.

Summary

- When team members follow shared principles and share the same set of values, it can make the method that they use much more effective.
- Scrum, a framework focused on project management and product development, is the most common approach to agile.
- In a Scrum project, the team breaks the work into sprints, or cycles of equal length (often 30 days) that follow the Scrum pattern.
- Every sprint starts with a sprint planning session to determine what they'll build.

Summary

- During the sprint the team works on the project, and every day they hold a short meeting called a daily scrum.
- At the end of the sprint, the team holds a sprint review with the stakeholders to demo the working software that they built.
- To finish the sprint, the team holds a retrospective to look back at how the sprint went and discuss ways that they can improve together as a team.