**Agile Software Development**

The course is designed to provide you with practical knowledge around Agile software development.

If you are new to the software industry, I have included the following two videos to give you an introduction to software development and some traditional software development models.

* What software development looks like?
* Intro to Software Development Models

If you are familiar with these topics, you can skip these two videos. There are no assessments based on these two videos.

We **start the course** by learning what led to the emergence of Agile methodologies (Why Agile?).

After that we talk about:

* The Agile mindset,
* Where to use Agile,
* The benefits and challenges in using Agile,
* How to apply the Agile mindset to software development.

**NOTE: During some of the videos in the first module, we will use terms that may not be familiar to you, but rest assured that we will cover most of those terms in some of the later videos.**

In the **second module**, we will learn about how to discover user needs in an agile way. We learn about **User Stories** which are a **basic unit of user needs** (some call them requirements) when using agile. Many teams use **cards**(index cards or sticky notes)to write down user stories. After learning about user stories, we learn about ways to **generate user stories**. We will cover two ways of doing this: 1) Story Workshop and 2) User Story Mapping

We will then learn about **estimating**and **release planning** using the agile mindset.

In the **third module**, we will learn about **Scrum**, one of the most popular agile frameworks. We will learn how to do sprint planning and tracking, Sprint retrospective, Sprint Review, Standups and what happens during sprint execution.

Finally in the **fourth module**, we will learn about **XP (eXtreme Programming)**. XP is also an agile framework that defines specific practices and values to apply agile mindset to software development.In the software industry**,**XP is known for its engineering practices. We will introduce XP and focus specifically on its values and primary practices. We will also cover how software is developed using XP.

I hope you enjoy this course and find this valuable. I would appreciate your feedback as I iterate :) to make this course better

What does software development look like?

Requirements – Design – Implementation – Verification – Operations and Maintenance

Continuous Integration – Automated Testing – Automated Deployment

**Why Agile**

Assumptions behind a predictive model – Requirements are accurate and wont change much.

Software is a creative development process

Agile is 4 values,

* Individuals and interactions over processes and tools
* Working software over comprehensive documentation
* Customer collaboration over contract negotiation
* Responding to change over following a plan

12 principles

Values:

Principles

1. Our highest priority is to satisfy the customer through early and continuous devliery of valuable software
2. Welcome changing requirements even late in development. Agile processes harness change for the customers competitive advantage
3. Delivery 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 tryst 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
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 – cost of exploration
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 behaviour accordingly

Adaptive

* Deliver working software frequently
* Welcome chang
* Technical excellence and good designs
* Continuous improvement

Detect translation issues early. Validate user needs early. Detect integration issues early

People and Interaction

* Business and developer work together
* Face to face conversations
* Self organizing teams
* Promote sustainable development
* Motivated individuals

Detect translation issues early

Adaptive

* Deliver working software frequently
* Welcome change
* Technical excellence and good designs
* Continuous improvement

Architecture, design, database modelling is a challenging. Lack of control, unpredictable journey, very uncomfortable for leaders/ organizations

People and Interaction

* Business and developer work together
* Face to face conversations
* Self organizing teams
* Promote sustainable development
* Motivated individuals

Requires participation from customers throughout the development process

* Ability to manage changing priorities
* Project visibility
* Increased team productivity
* Delivery speed/time to market
* Team morale
* Business/IT alignment
* Software quality
* Project predictability
* Project risk reduction
* Engineering discipline
* Software maintainability
* Managing distributed teams
* Project cost reduction

When to use and not to use Agile…

Agile only works for small projects – False

Will Agile work for big projects, mission critical projects, Govt projects ---

Is Agile a good fit for all situations – no, not valuable for predictive, repeatable work

Prerequisites to use Agile methods

* Requires close collaboration with business & users
* Team should be setup to digest change, engineering practices

**Barriers to Agile adoption**

* Company philosophy or culture at odds with core agile values
* Lack of experience with agile methods
* Lack of management support
* General organization resistance to change
* Lack of business, customer, project owner
* Insufficient training
* Pervasiveness of traditional development
* Inconsistent agile practices and process
* Fragmented tooling, data and measurements
* Ineffective collaboration
* Regulatory compliance and governance
* Don’t know

**Applying an Agile mindset to a project**

Define, design, build, test

Plan, build, learn, Repeat

Build small bite size chunks rather than big batch

Collaboration of teams and areas, rather than handout / handoff to team once section finished

**Adaptive**

* Delivery working software frequently
* Welcome change
* Technical excellence and good design
* Continuous improvement
* Simplicity

Strong engineering culture – automated build and deployment, continuous integration, automated tests, collective ownership. Adaptive Planning

**People & Interaction**

* Business and Developers work together
* Face to face conversation
* Self organizing teams
* Promote sustainable development
* Motivated individuals

Psychological safety (happy to raise hand and ask questions. Partner with customer. Collaborative environment. Cross functional teams

Agile project journey, step by step

**Chartering**

* Alignment
* Team building
* Spot issues

1 -2 days

**Discover user needs**

* Story workshop
* Story mapping

1 week

**High level architecture**

* Make some of the architecture decisions that must be made before proceeding further

2-3 days

**Release planning**

* Plan our next step

1-2 days

**Delivery**

* Define, build, test, learn collaboratively
* Improve

2-3 weeks, Repeat

**Optional – UAT, business readiness, communication**

* Final check
* SOP’s
* Newsletters etc
* Training etc

Deployment

Changing backlog, stories, architecture

Once deployment is done it goes back to the release planning

**Agile Frameworks**

How do I apply agile mindset?

**Scrum**

1 to 4 week sprint

Define, design, build, test

Framework contains: Product owner, Scrum master and The team

Product owner – Inputs from executives, team, stakeholders, customer users to create what is to be done, list of what is required, definitions etc.

Product backlog – Sprint Planning meeting, work out what order from the product backlog things will be done.

Sprint backlog, what exactly needs to be done for each task, during each sprint.

Daily scrum meeting (stand-up) to review what was done yesterday and for today. Sprints are 1-4 weeks, and the sprint end date and team deliverable do not change. – Finished work.

Sprint review – full team gets together with all involved, customers, stakeholders etc to go through what was done and if any changes are needed

Sprint retrospective – how can it be better next time, what went well, badly, how it can be improved.

Burndown / Burnup charts – breaksdown what daily work is needed and to be finalised for that sprint

**Kanban**

Set or properties and principals so long as there is a continuous flow.

Backlog – Software development pipeline – end result

Backlog is a list of work to be done

Backlog, Analyze, Develop, Rest, Release – all steps are put into columns, with Doing, Done breakdown within each

From the backlog each task with the breakdown is moved from Doing to Done as it progresses.

1 – Visualize your workflow

2 – Limit WIP (work in progress) to a set number so to prevent log jam in testing for example, once work is done if there are too many items in a WIP then it should be looked as to why, code is wrong, testing taking too long etc.

3 – Manage the flow, are there issues, is the code wrong, does it require more people.

4 – Make process policies explicit – When is something done to move to testing

Kanban Principles

* Start with what you do know
* Agree to pursue incremental evolutionary change
* Respect the current process, roles, responsibilities and titles

Kanban Properties

* Visualize the workflow
* Limit WIP – work in progress
* Manage flow
* Make process policies explicit
* Improve collaboratively

SCRUMBAN is a combination of both

Agile is a mindset. Frameworks should be customised

Scrum – 1 to 4 week cycle – most popular framework

Kanban – continuous flow model

Scrumban – continuous flow in a cycle

XP – Extreme Programming

Scrum XP hybrid

Lean startup

Feature Driven Development – FDD

Crystal

Custom Hybrid

Dynamic Systems Development Methodology – DSDM

**Gathering Requirements: The Agile Way**

Build what user really needed

Help build shared understanding between all stakeholders involved

Conversation

* Conversation as primary form of communication

Adaptive

* Discover user needs vs collect user needs

Agile approach: Conversation

Hand off requirements document? Requirements gone wrong. It does not mean no documents, just enough just in time

Progressive Refinement – Just Enough Just in time

Story Process

Card – conversation – confirmation

Agile Approach – Adaptive

Plan, build, test, repeat – the requirements change, update and develop over time

**User Stories: The currency of agile development**

User stories 3 C’s

* Card – Token for conversation
* Conversation to build shared understanding
  + Who (wants) What (and) Why?
  + Record facts/info to help you recall the conversation
  + Discuss what happens outside the software
  + Discuss what can go wrong
* Confirmation – Acceptance tests

**Templates**

* As a <role type> I want to <perform task> so that <achieve this goal>
* <persona> wants to <perform a task> to <achieve goal>
* Create your own

It doesn’t need to be written in a template to be considered a story

**How to write acceptance tests**

Another way to think about them:

Once the story is done what client/product owner will check to validate that it is indeed done

What will we test to confirm that this story is done

How will we demonstrate this software at a product review

Use simple business/user lingo that team members can understand

Specify ‘what’ not ‘how’ eg a manager can approve or disapprove an audit form rather than a manager can click an approve / disapprove radio button to approve an audit form

Why write accetance tests

* Product owner
  + Helps you think through
* Team
  + Helps build common understanding between team members (esp PO, Dev and tester
  + Team respsose time will improve
  + Helps write test cases to confirm that story is done
  + Helps you slice out work

Stories can be written and many levels of abstraction

Epic, Feature, Capability – Right sized for business

User Story – Right sized for users

Dev Story, Small User Story, Right sized for development

**Spikes**

* Knowledge gathering story
* These are stories for research / exploratory work
* Time boxed
* Clear definition of done

**Non functional requirements**

* Handle them 2 ways
  + Definition of done (applicable to all stories)
  + Create specific stories (applicable to small part of system)
* Make you have measurements for the ‘ilities’ (faster, more accurate, quickly)

**Characteristics of good user stories**

INVEST:

* Independent – stories independent of each other
* Negotiable – discussion take place later on
* Valuable – add value to the user
* Estimable – create a story that we can estimate
* Small – stories should be short to deliver
* Testable – validate what was designed

These are general guidelines

Don’t let it come in the way of building right stuff effectively

Examples

As a developer I want to finalize the database table changes

Delete it – this is not a story this is a task

**Gathering User Stories**

**User story writing workshop**

Logistics

Goal – Write as many stories as you can for the selected theme

Who to invite – Product Owner and other stakeholders who knows user needs. Scum Master. Development Team

How Long – Few hours to few days

Identify Users - User analysis

Create personas

Everybody start writing stories silently around the selected them. Top down (big functionality stories. Bottom up (group them later). Free form (group them, split them etc

Characteristics of good product backlog

* Detailed appropriately
* Emergent
* Estimated
* Prioritized

**Story Mapping**

Technique to: discover user needs. Organize and prioritise story backlog. Understad and communicate user needs. Plan releases and development

**Story Map Structure**

Left to right (time)

Top to bottom (priority)

Top left high priority to be done quickly.

Bottom right, low priority end of timeline

Broken into User Activities. User tasks (walking skeletons). User stories

Step 1 – Frame the problem

Story – product teaser/story

What – name of the product, problem we are trying to solve

Who – who are the users and what benefit they get

Why – benefit to organization. What users do and how it results in benefits

Step 2 – Map the big picture: Activities

Tell a big story of the product by starting with the major user activities the system will be used for

Arrange activities left to right in the order you’d explain them to someone when aksed the question – What do people do with this system

Add task centric stories under each activity in workflow order left to right

If you were to explain to someone what a person typically does in this activity arrange tasks in the order you’d tell the story, don’t get too uptight about the order

Step 3 – Explore variations

Overlap user tasks vertically if a user may do one of several tasks at approximately the same time, or -> vertical and then -> horizontal

Explore;

* Discuss, fill in, refine the map and test for completeness
* Sky is the limit, go crazy, don’t worry things will be prioritized / sliced out
* Look for exceptions
* Consider other users
* Involve others

Step 4 – Slice out viable releasees

Slice map to holistic meanginful releases

Focus on outcome – slice away what is not needed

For each release identify – outcome and impact & success criteria

Why create them?

* Discover user needs – especially help discover missing pieces
* Understand and communicate user needs
  + Help communicate at different levels
  + Help tell a story

Planning – provide a useful context for prioritization

Plan releases in complete and valuable slices of functionality

Organise and prioritise story backlog

Foster co ownership

Flexible

**Agile Estimation and Planning**

Agile Planning

* Multi level
* Less upfront but frequent
* Just enough just in time
* Adapt and re-plan

Vision / strategy – create product backlog

Release planning – sprint planning

Daily Standups

**Agile approach for estimation**

3 concepts

* Effort vs duration
* Accuracy vs precision
* Relative vs absolute

**Effort vs Duration**

Ideal days – how many work days will it take to complete a story if you work on it uninterrupted

**Accuracy over Precision**

Sizing buckets – estimation scale

* Popular scales:
* 1,2,4,8
* Fibonacci series 1,2,3,5,8…
* 1,5,10,15
* SX,S,M,L,XL
* Small, medium large

What if I cant estimate a story? Add ? as an option

How many buckets? Depends

* Less buckets -> faster estimation less precision
* More buckets -> slower estimation, more precision
* Diminishing return with more precision

**Relative vs Absolute sizing**

Relative sizing examples

* If story A is 1 point, then story B is 5 points (in agile you will hear story points)
* Story A is 5 apples, story B is 15 apples, story C is 10 apples
* Small, Medium, Large
* T-shirt sizing, XS, S, M, L, XL

Absolute sizing examples

Story A will take 1 day, Story B will take 5 days

Story A will take 4 hours, story B will take 2 hours

|  |  |
| --- | --- |
| Story Point (Relative | Ideal Day Buckets (absolute) |
| Difficult to get accustomed to | Natural |
| Faster (once you get used to) | Tedious |
| No 'myideal days are not your day issue) | my ideal days are not your ideal days' |
| Difficult to explain outside the team or to new member | Very natural to explain |
| Makes it hard for management to behave badly | Management may try to drive team to ideal days = regular days |

**Estimation Styles & Process**

Who estimates?

* Development team

How long does it take?

* Depends on style, method, number of stories, understanding, knowledge, expertise, number of team members

Estimation styles

* Simple – free form
* Planning poke
* Card sorting

**Planning Poke**

* Everybody gets the estimation cards
* Explain the story
* Understand the story
* Estimate and put one card down
* Open cards
* If consensus move to next time
* Else discuss variations and go back to step 3

Pros and cons

* Time consuming
* Uncover misunderstanding
* Collective ownership
* Engaged
* Good for backlog grooming session

**Card Sorting**

* Place smallest (left) and largest (right) – on a board
* Silently everybody start placing stories between these 2 cards
  + Separate column for stories with questions
  + Stack same size stories vertically
  + Disagree? Move the story silently
* Discuss changes
  + Discuss stories with question and put them back on the board
  + Take another look
  + Discuss disagreements and move stories if needed
* Create buckets
* Assign size to buckets

Team can / should use existing buckets team currently use. Can start with those in step 1

When to use card sorting? Useful for estimating large number of stories together

**Velocity**

Roughly amount of work getting done in a sprint

Changes based on team, project, and many other factors

Calculating Velocity:

* For a given sprint, if you finished 3 stories with estimate of 5, 3, 10 then your velocity for that sprint is 18
* To calculate velocity to use for planning upcoming spring:
  + Last sprint velocity 23
  + Average of last x sprints – next sprint using last 3, Avg(25+27+23 = 25
  + Velocity range of last x sprints – next sprint using last 3, 23-25
* Skip anomalies
  + Skip sprints outside of the norm range if using last 5

Adjusting Velocity / Estimates

Calculating velocity for 1st sprint

* Take a few sample stories from backlog that team think they can deliver in a sprint.
* Team may want to task out the story to understand the work involved
* Sum up the estimate of stories

More people joining the nest sprint? Few people leaving the team? People are going on vacation?

* Continue to use velocity of previous sprints (except if most of the team is going – like holidays). It takes time for people to get up to speed and it is difficult to predict the impact of a personal change on teams productivity

What if your estimates were way off?

* Generally it averages out, no need to worry about one offs. If there is a new learning and bunch of similar stories are going to move from one estimate bucket to another then update their estimates

**Release Planning**

1 – Fixed Scope – how long will it take

2 – Fixed Date / Time – What can we deliver

Fixed Scope Release

Fixed scope, variable date

When will you deliver release 1

* Decide sprint length
* Calculate velocity or velocity range
* Total up estimate for selected stories
* Total estimate / velocity ~= #sprints
* # springs x sprint length = duration

Total of story points = 120 points

2 week sprints

Cost = 50k per week

Velocity range (last 3) 15 to 20

At high velocity, 120/20 = 6 sprints

At low velocity, 120/15 = 8 sprints

So it will take between 6 & 8 sprints

Duration = 12 to 16 weeks

Cost = 12\*50k to 16\*50k = 600k-800k

How to select stories?

* Use story map
* Shore feedback loop
* Learning
* Value to users / market position
* Risk mitigation
* Dependencies

Fixed Date Release

* Date fixed so scope changes
* What are you going to deliver
* Groom backlog – if not done so
* Calculate velocity range
* Select spring length
* Calculate # of sprints
* Calculate release capacity = # sprint x velocity (use 2 times to get range)
* Include items from backlog (starting at top) until total point exceed points range: will have, might have

Each story is give an point total which will be part of the calculation

Nest release in 8 weeks

2 week sprints

Velocity range = 15-20

Number of sprints = 8/2 = 4 sprints

Min scope 4 \* 15 = 60 points

Max scope = 4 \* 20 = 80 points

**Release Tracking**

* Release burn up – total work vs stories completed on a graph
* Story board – backlog, development, testing, done
* Cumulative Flow - total work vs stories completed vs development vs testing vs deployment - all on a graph

**Scrum Overview**

1 to 4 week sprint

Define, design, build, test

Framework contains: Product owner, Scrum master and The team

Product owner – Inputs from executives, team, stakeholders, customer users to create what is to be done, list of what is required, definitions etc.

Product backlog – Sprint Planning meeting, work out what order from the product backlog things will be done.

Sprint backlog, what exactly needs to be done for each task, during each sprint.

Daily scrum meeting (stand-up) to review what was done yesterday and for today. Sprints are 1-4 weeks, and the sprint end date and team deliverable do not change. – Finished work.

Sprint review – full team gets together with all involved, customers, stakeholders etc to go through what was done and if any changes are needed

Sprint retrospective – how can it be better next time, what went well, badly, how it can be improved.

Burndown / Burnup charts – breakdown what daily work is needed and to be finalised for that sprint

**Scrum Framework**

**3 roles**

Product owner – what needs to be done and what order – talks to exectutives, teams, stakeholds, customers, users – product backlog (very high level)

Scrum master – helps team stay true to scrum values and principles – facilitates meetings, drives resolution of road blocks

Team – self organising, developers, testings, build the XYZ

Sprint planning meeting, work on what will be done next or going forward and lets the product owner know

Team gets together to discuss the sprint backlog, commit to stories then -> sprint then is executed – daily stand up on what was done yesterday and what will be done today

End of sprint work will be finished

Sprint review – full team gets together with all involved, customers, stakeholders etc to go through what was done and if any changes are needed

Sprint retrospective – how can it be better next time, what went well, badly, how it can be improved.

Burndown / Burnup charts – breakdown what daily work is needed and to be finalised for that sprint

**Sprint Planning and Tracking**

**Sprint planning meeting – Burndown / burnup**

**Backlog grooming**

* Removing user stories that no longer appear relevant
* Creating new user stories in response to newly discovered needs
* Re-assessing the relative priority of stories
* Assigning estimates to stories which have yet to receive one
* Correcting estimates in light of newly discovered information
* Splitting user stories which are high priority but too coarse grained to git in an upcoming sprint

Selecting and preparing stories

* How do you select potential stories
* Story map
* Prioritized backlog and velocity
* Alternatives, themes, anything team wants to learn

Make sure stories are ready to be work upon ( who, what, why, acceptance test and any major dependancies)

Sprint planning begins

Two possible ways

1 step, select one story at a time, task it out until capacity is reaches

2 step – select stories based on velocity. Task out and gain confidence – some stop after first step

Sprint planning steps

* Determine sprint capacity
* Review sprint goal- if any
* Review potential stories
* Acquire confidence: design discussion and task out stories
* Refine sprint goals if required
* Make commitment
* Put the stories and task on the task wall

Determine capacity based on days, other scrum activities, hours per day, available effort hours

Total task estimates from last sprint

Avg of total task estimates from last 3 sprints

Step 2 – define sprint goal

At the send of sprint x, persona will be able to do XYZ

Something that team can rally around

Step 3 – Review stories

Product owner goes through selected stories

Tip – Elicit feedback, people not asking questions is generally a bad sign

Tip – setup a definition of ready for stories

Step 4 – Acquire confidence. Design discussion and tasking out

Not everybody has to stick around. PO to answer questions is helpful – could be IM based or interrupt basis

Face to face conversation helps

Tip – time box activity / discussions

Tip – estimate taks in ideal days/hours

Step 5 – refine goal if needed

Step 6 – Make Commitment

Step 7 – Create task board

Sprint Planning Logistics

Who to invite – tip, setup a recurring meeting

Core team members (PO, SM, dev, test, analysts etc)

Users / clients / business people who can answer questions if any

How long?

It depends on complexity / preparedness / teams maturity but general guideline – if you include ‘tasking out’ activity around 4 hours for a 2 week sprint

FAQ

How do we handle PTO;s do you update velocity to handle people leaving or joining the team – no unless large numbers

How do you handle interruptions? PO should make decisions, evualute sprint if commitments can be met

Do we estimate defects? – Yes or not it can depend

**Sprint Tracking**

Burn Down – Work left

Burn Up – Work done + total work

Task Board – Visual way of tracking on pace or not – task board, visible during standup, definition of done, too many in progress smell

**Sprint execution and standups**

* Who works on what
  + Order the cards in priority
* Limit work in progress
  + Find the right balance
* Parallel work vs Swarming
  + Practical? Skill x helping skill y to finish story
* Generalists vs Specialists
  + Skill map, pairing
* Discipline, Discipline, Discipline
  + To follow what team decides to do
* Engineering Practices

**Daily standup**

* What
  + Common 3 questions
  + Alternatives: work items attend / story focused standup
* Who
  + Core team + any stakeholder who wants to attend
* Purpose
  + Daily team planning
  + Collaborate
  + Identify blockers
  + Status check
* Tips
  + Show the board
  + Parking lot
  + Keep it short

**Sprint Review**

Happens end of sprint where work is demod / shown off

* Purpose
  + Review work done and learnings from the sprint
  + Get feedback and adjust future direction
  + Celebrate
* Who?
  + Core team
  + Stakeholders
  + Anybody and everybody

How long – 1-2 hours

What happens? – Summarize (roadblocks, issues), demo, discuss, adapt

Tips

* Don’t wait for review to show your work – early regular feedback
* Some teams follow this rule -> demo done things only
* Make effort to get stakeholders to attend
* Presentation by individuals
* Prep work

**Sprint Retrospective**

* Who?
  + Core team
* When?
  + At the end of each iteration (sprint)
* Time?
  + 1 hour length
* Purpose?
  + Continuous improvement
* What Happens?
  + Whats working
  + Whats not working
  + Action items

**Alternative – Focused Retro**

* Steps
  + Set the atmosphere
  + Share context
  + Identify insights
  + Determine actions
  + Close the retrospective
* Follow through
* Change the retro format

**Tips**

* Avoid finger pointing
* Select few action items
* Get people talking
  + Non work question
  + Round robin
  + Manager opt out
* Individual – Group – All

**XP (Extreme Programming) Overview**

What is it about?

Engineering practice

Social change

Lightweight

Adaptive to changing requirements

Values  
Principles

Practices

Controversy

Prescriptive

Pair programming

Incremental Design

Scalability

**XP Values**

Simplicity

Communication

Courage

Feedback

Respect

Simplicity

We will do only what is necessary and nothing more -> maximize value

What is the simplest thing that could possibly work?

We will take smaller steps to our goals and iterate

Communication

Everyon is part of the team and we communicate face to face daily

We will work together on everything from requirements to code

Important for creating a sense of team and effective cooperation

Courage

We will tell the truth about the progress

We done document excuses for failure because we plan to succeed

We will adapt to changes whenever they happen

Feedback

Getting feedback is important if you want to iterate and improve

Generate as much feedback as a team can handle as quickly as possible

Slow down feedback if you cant adapt

Feedback comes in many forms

Respect

Foundation of previous four

Respecting each other is key for XP to succeed

No on is intrinsically worth more than anyone else

**XP Practices**

Do practices while keeping their purpose in mind

Practices were written with ideal state in mind. Keep making progress towards them

Experiment and see if it helps

Practices work well together

Primary and Corollary practices

**Sit together**

Open working environment

Highly collaborative environment

Not before team is ready

**Whole Team**

Everyone needed for project success is part of the team

Team composition is Dynamic

No fractional people

**Informative Workspace**

Get an idea in 15 seconds

Dynamic information

Cleanliness and order

**Energized Work**

Working hours – productive

Sick – stay out – rest get well

Incremental improvement

**Pair Programming**

2 people working together – work alone when needed

Benefits

Keep each on task

Brainstorm refinements to the system

Clarify ideas

Take initiative when their partner is stuck, thus lowering frustration

Hold each other accountable to the team’s practices

**XP Practices**

* Sit Together
* Whole Team
* Informative Workspace
* Energized Work
* Pair Programming

**Stories**

Unit of functionality

Stories are flexible

Estimate early

Keep stories visible – don’t computerize

**Weekly cycle**

* Plan weekly
* Review progress
* Select weeks work
* Break stories into tasks

Gradually reduce planning time

Task ownership

**Quarterly Cycle**

Plan work quarter at a time

Focus on the big picture where the project fits within the organization

Plan the theme or themes for the quarter

**Slack**

Build some slack time into the process

How?

Lower priority tasks that can be skipped

One week every 8th week as geek week

20% time for programmer chosen taks

Avoid aggressive commitments

**Ten Minute Build**

Build and run all the test within 10 minutes

New to agile

* Automated build
* Key tests and continue to evolve

Gives confidence and reduces stress

**Continuous Integration**

Integrate and test changes after no more than a couple of hours

Asynchronous integrations – once dev changes done, separate process integrates the code and developer notified if anything broken

Synchronous integrations – code is integrated and developer waits for feedback

Test First Programming

Write test – run tests to see it fail ->

Write code -> until test pass

Benefits

Avoid Scope creep

Build trust in team members

Reduce coupling

Incremental Design

Invest a little in design everyday

Excess complexity -> refactor

Architecture emerges over time

**XP Practices**

Stories

Weekly Cycle

Quarterly Cycle

Slack

Ten minute Build

Continuous Integration

Test First Programming

Incremental Design

**XP Process Model**

Release planning – what team will deliver in the next release

The come from user stories or architectural spike?



Scrum and Extreme Programming (XP) are definitely very aligned. In fact, if you walked in on a team doing one of these processes you might have hard time quickly deciding whether you had walked in on a Scrum team or an XP team. The differences are often quite subtle, but they are important. I think there are four main differences between [Scrum](https://www.mountaingoatsoftware.com/agile/scrum) and XP:

1. Scrum teams typically work in iterations (called sprints) that are from two weeks to one month long. XP teams typically work in iterations that are one or two weeks long.
2. Scrum teams do not allow changes into their sprints. Once the sprint planning meeting is completed and a commitment made to delivering a set of product backlog items, that set of items remains unchanged through the end of the sprint. XP teams are much more amenable to change within their iterations. As long as the team hasn’t started work on a particular feature, a new feature of equivalent size can be swapped into the XP team’s iteration in exchange for the unstarted feature.
3. Extreme Programming teams work in a strict priority order. Features to be developed are prioritized by the customer (Scrum’s [Product Owner](https://www.mountaingoatsoftware.com/agile/scrum/roles/product-owner)) and the team is required to work on them in that order. By contrast, the Scrum product owner prioritizes the product backlog but the team determines the sequence in which they will develop the backlog items. I’ve never seen a Scrum team not choose to work on the highest-priority item. And a Scrum team will very likely choose to work on the second most important. However, at some point one of the high priority items may not be a good fit for the sprint being planned—maybe a key person who should work on it will be swamped by work on higher priority items. Or maybe it makes sense to work on a slightly lower priority item (let’s say #10 on the product backlog instead of #6) because the team will be working in the code where #10 would be implemented.
4. Scrum doesn’t prescribe any engineering practices; XP does. I love the XP engineering practices, particularly things like test-driven development, the focus on automated testing, pair programming, simple design, refactoring, and so on. However, I think it’s a mistake to say to the team “you're self-organizing, we trust you, but you **must** do these specific engineering practices....” This sends a mixed message to the team that causes confusion. I love the XP practices but don’t like mandating them. I want teams to discover the value on their own.

These are small and often subtle differences between Scrum and XP. However, they can have a profound impact on the team. My typical advice to teams is “start with Scrum and then invent your own version of XP.” The XP practices are wonderful but they work best and teams commit to them the most stridently if they discover them themselves rather than having them mandated. I help teams do this in my coaching by asking questions like, “Would this bug have happened if we'd been doing test-driven development?” and “Would we have made that mistake if we were pairing?” I find true XP to be a small target off in the distance. If a team can aim at that and hit the bull’s eye, wonderful. If not, however, they are likely hacking (e.g., refactoring without any automated testing or TDD). Scrum is a big bull’s eye that on its own brings big improvements simply through the additional focus and the timeboxed iterations. That’s a good starting point for then adding the XP practices.