Agile Software Development

Week 1

Benefits and Challenges of Agile

Predictive model recap

1. We can predict the requirements accurately
2. Translation is going to be perfect

Agile principles recap

Adaptive

- Deliver working software frequently
- Welcome change
- Technical excellence and good design
- Continuous improvement
- Simplicity

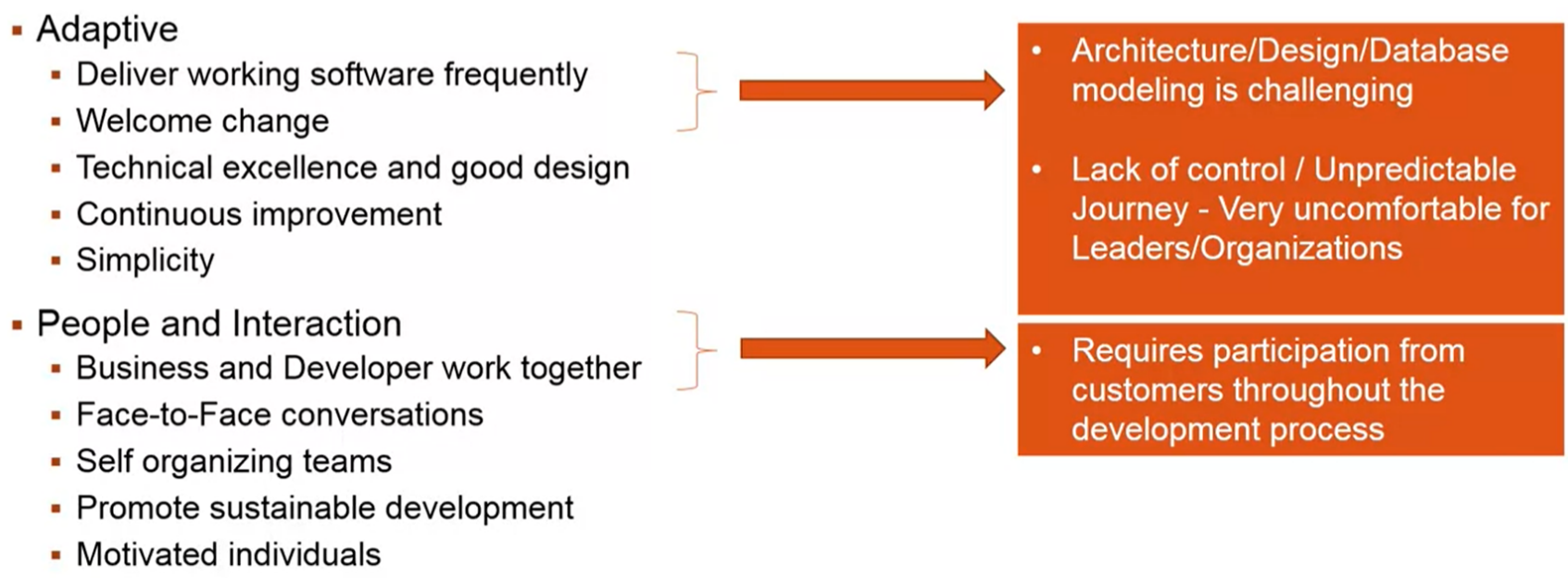
- Detect translation issues early
- Validate user needs earlier

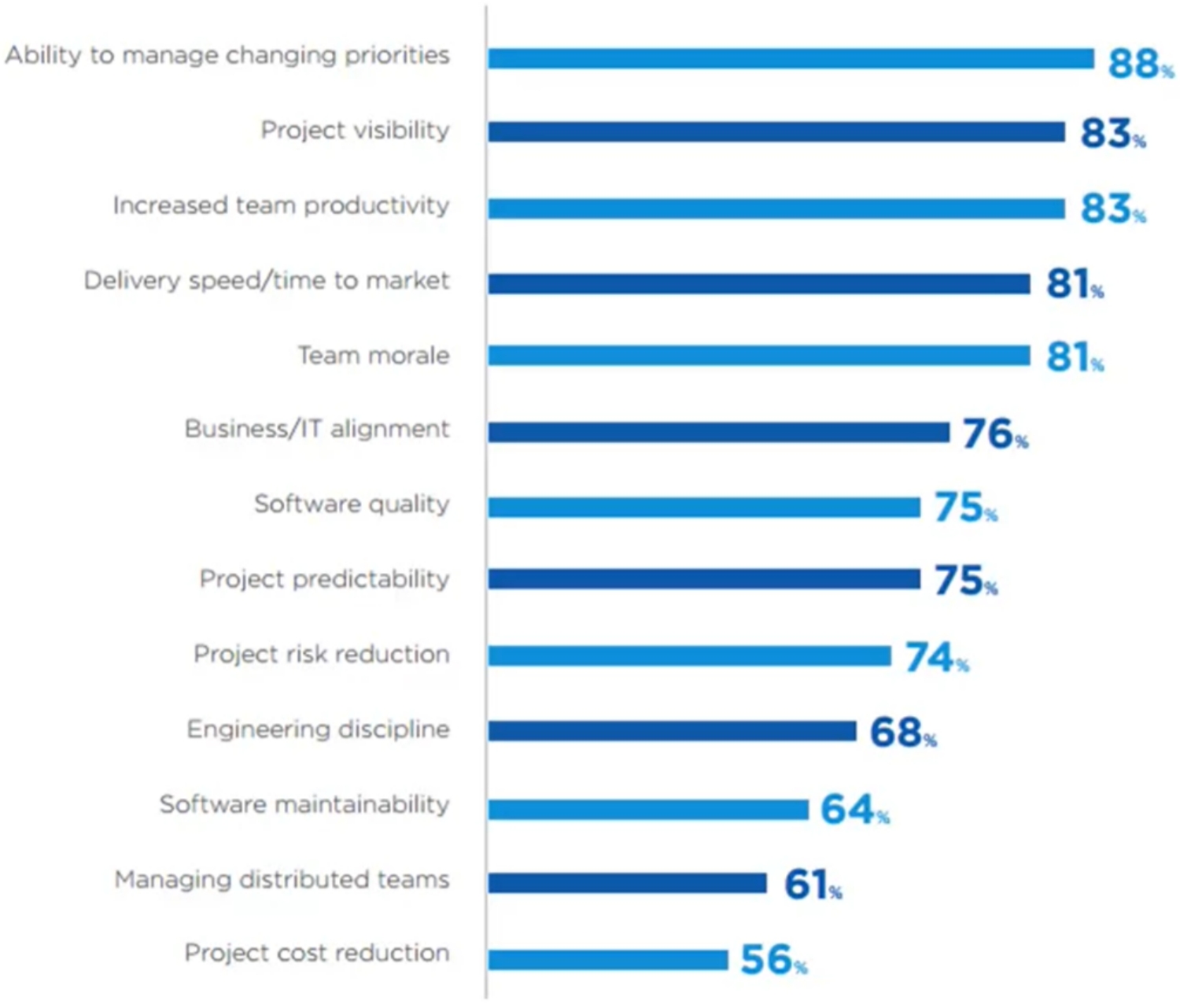
People and Interaction

- Business and Developer work together
- Face-to-Face conversations
- Self organizing teams
- Promote sustainable development
- Motivated individuals

- Detect translation issues early

New problems agile bring





Agile: When to Use and When NOT to!

there was a conception that Agile only works for a small projects.

for the small-size projects, Agile was 58% successful and Waterfall was 44%.

for large-size projects, Agile was 18% successful and Waterfall was 3% successful.

even for mission critical projects or government projects, I would say Agile has found its way and people are using it and finding it useful.

if you are **doing something that is very predictive and very repeatable,** Agile can work but it's not going to be valuable. So in that case, a traditional model or a predictive model will work much better.

Prerequisites to use Agile methods

Requires close collaboration with business, users

Team should be setup to digest change - Engineering practices

Barriers to Agile adoption

Applying an Agile Mindset to a Project

Big batch vs Small bite-size chunks

at the Waterfall method, you do your requirements, takes a couple months, and then you get your requirements document, then you do your design and you get a couple more documents, and then implementation and verification. And after all this work, all of these phases, you get your product in one big batch.

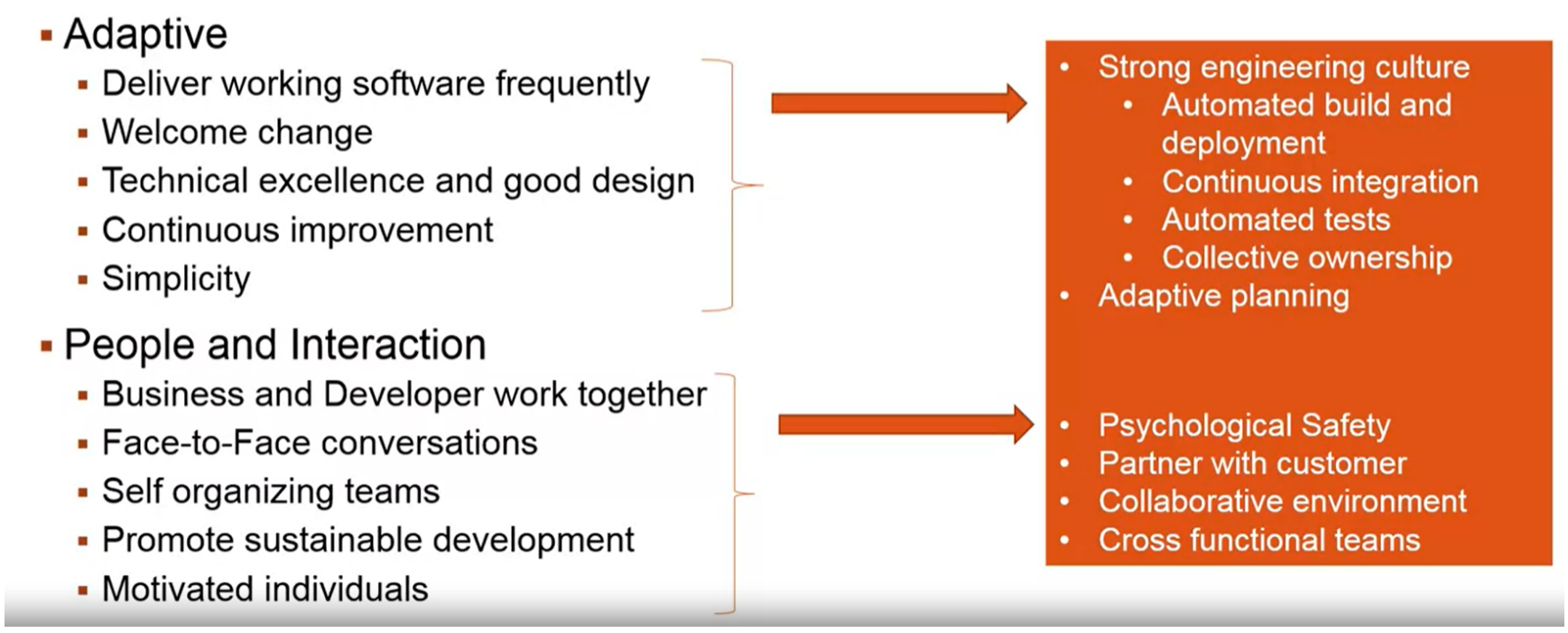
doing Agile, you work in these small cyclical, small cycles. So after every cycle, you get a part of the product. And after the first two weeks or three weeks, whatever is the size of your cycle, you get some product out, and then after the next sprint, you will get some more product built iteratively and incrementally. And then you may realize, we need to adjust and then you might get a different or adjusted or changed product in the next portion. And then again you add more functionality and then again you may realize, we need to pivot again, and then again you may change the direction of the project or the product. And then, finally, you get your products

Handoff vs collaboration

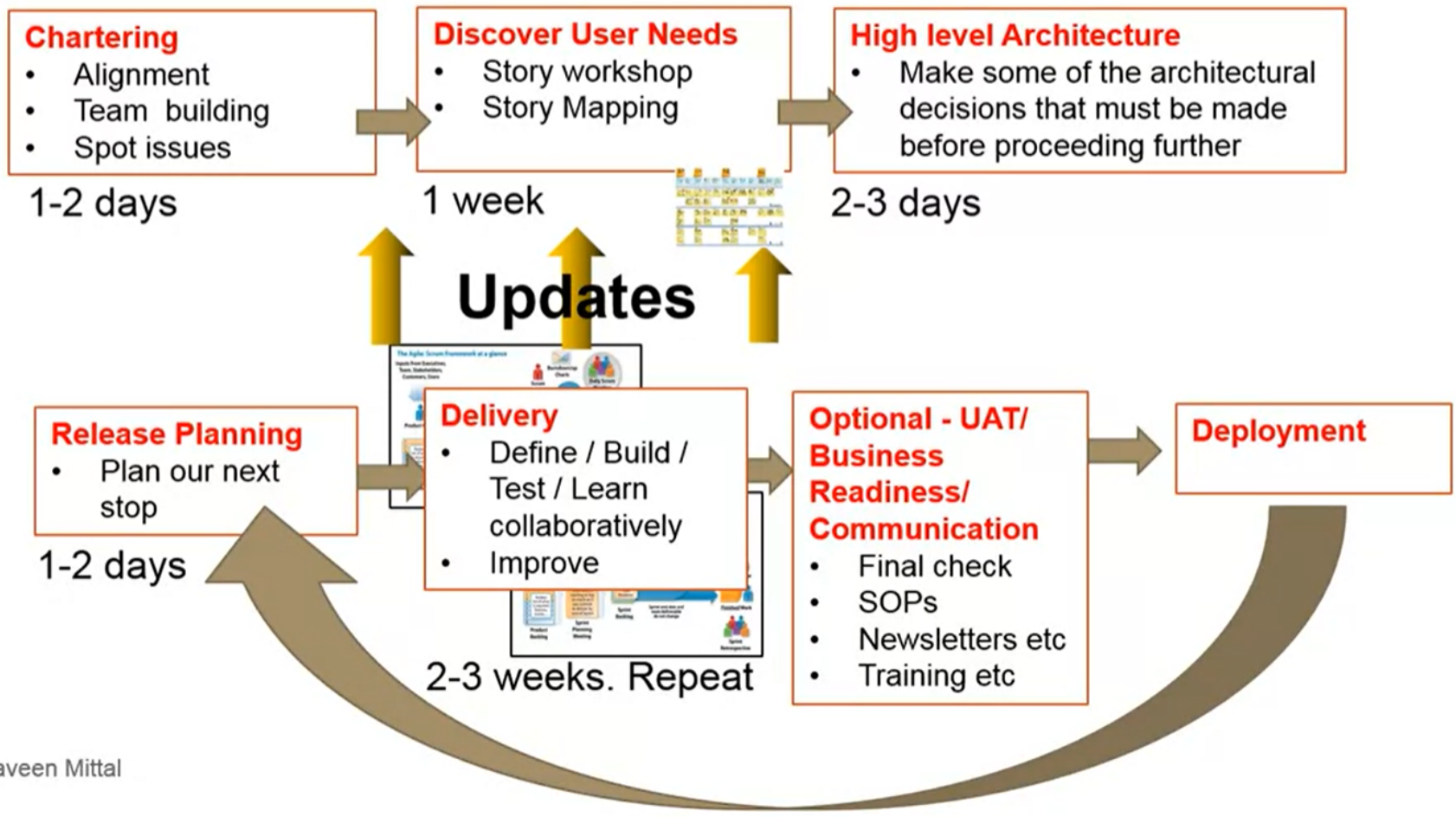
Waterfall method: handoff

Agile: collaboration

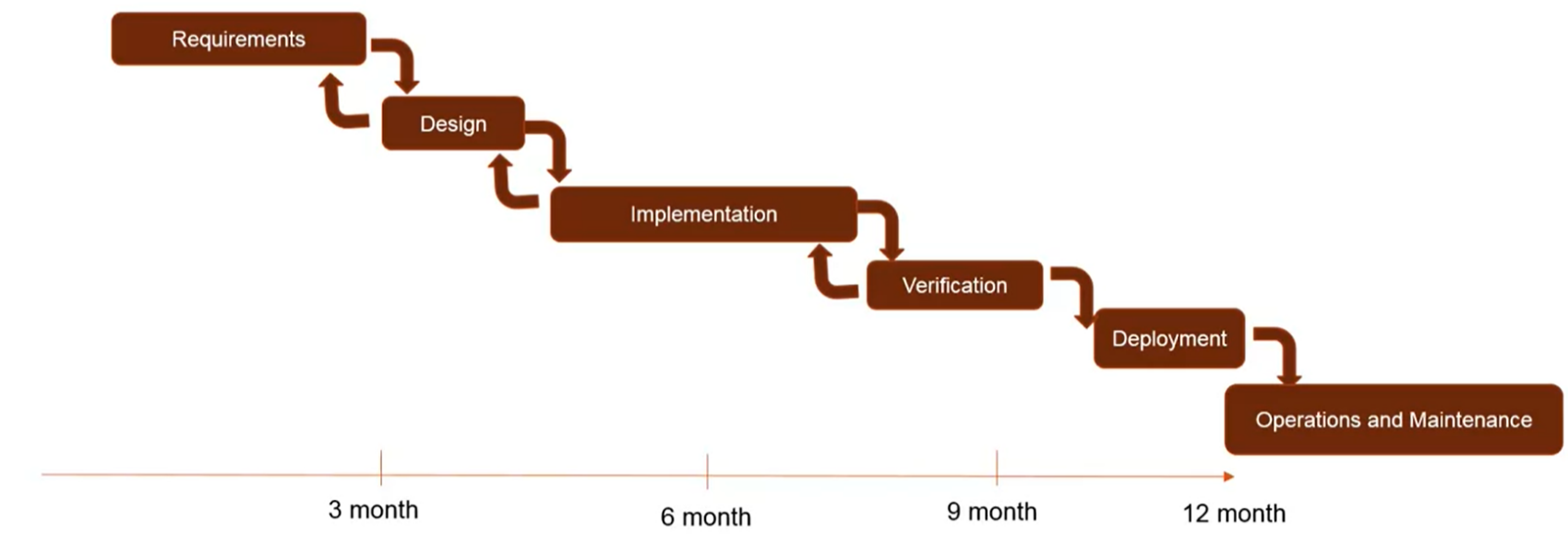
Culture change



Agile project journey



raveen Mittal



Week 2

Gathering Requirements: The agile way

What a User Requirements Process is good?

If the User Requirement Process helps you build what the user really needs

it helps build a shared understanding among all the stakeholders of the product.

Agile approach

Encouring conversation
+ Conversation as primary form of communication

Adaptive in the requirements
+ Discover user needs vs Collect user needs

Documentation
+ Just enough - Just in time (Progressive refinement)

Story process

3Cs of user stories: Card → Conversation → Confirmation

Agile approach - Adaptive

We don’t know the needs upfront and we are going to discover the needs

Allow requirements to change over time

User Stories: The currency of agile development

User stories - 3C’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

Some key part of a user story
+ The story title
+ The estimated development time
+ WHO/WHAT/WHY
+ Acceptance test

Template to capture WHO/WHAT/WHY

Templates

- As a `role type`, I want to `perform a task` so that `achieve this goal`
- `Persona` wants to `perform a task` to `achieve this goal`
- Create your own...

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

Examples

- **Title:** User login
  As a `registered user`, I want to `log in`, so I can `access subcriber content`

Other information: title, estimate, sketches, relevant notes and details

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”
(e.g: `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`)

Example

User story: *As an Administrator, I want to be able to create User Accounts so that I can grant users access to the system*

Acceptance tests:

- If I am an Administrator, I can create User Accounts
- I can create a User Account by entering the following information about the User: a. Name, b. Email address, c. Phone number, d. License number, e. Account status, f. Reports to (from a list of “active” user)
- I cannot assign a new User to report to an “inactive” user
- I cannot assign a new User to report to a User if it creates a cyclical relationship (e.g: User 1 reports to User 2 who reports to User 1)
- The system notifies me that it sent an email to the new User’s email address, containing a system-generated initial password and instruction to log in and change their password

Why write acceptance tests?

Product owner

- Helps you think enough

Team

- Helps build common understanding between team member (esp. PO, DEV, tester)
- Team response time will improve
- Helps write test cases to confirm that story is done
- Helps you slice out work

Stories can be written at many levels of abstraction

Epic, Feature, Capability → right sized for business

User story → right sized for user

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 the system)

Make sure you have measurements for the ‘ilities’ (faster, more accurate, quickly)

Characteristics of good user stories

INVEST

- Independent
- Negotiable
- Valuable
- Estimate
- Small
- Testable

These are general guidelines

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

Example

As a developer, I want to finalize the database table changes
→ Delete it, Convert this into a task

As a Manny’s food service customer, I want to see different food item types displayed in different colors - RGB, #FF0000 for meats, #A52AFA for grains, and #808000 for vegetables and fruits - so that I can quickly identify my food items by food type.
→ As a Manny’s food service customer, I want my custom item code to **stand out** so that I can find it on the screen more quickly

- As a business user, I would like a report of item profitability so that I can identify and highlight profitable items and consider what to do about underperforming items.

✓As a marketing manager considering how to spend limited marketing dollars, I need a report of the **most/least profitable items** so that I can identify and highlight profitable items and consider what to do about underperforming items.

- *As a customer ordering food, I want to locate previous food order lists so that I can see all the lists that I have.*

✓*As a customer ordering food, I want to see my saved food order lists so that I can* ***reuse the list for future orders, making ordering faster and more accurate.***

- *As a tester, I want to have detailed test plans so that when the system is completed, I can test the system.*

✓*Delete it. Covert this into a task*

- A company can pay for a job posting with a Visa card
- A company can pay for a job posting with a master card
- A company can pay for a job posting with a American express card

✓A company can pay for a job posting with one type of card.

✓A company can pay for a job posting with two additional type of card.

- A user must find the software easy to use

✓A novice user can use the software without a need of training

Generating User Stories

Two methods

User story writing workshop

Story mapping

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

Agenda

Identify Users - User analysis

Create personas

Everybody start writing stories silently around the selected theme

- 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

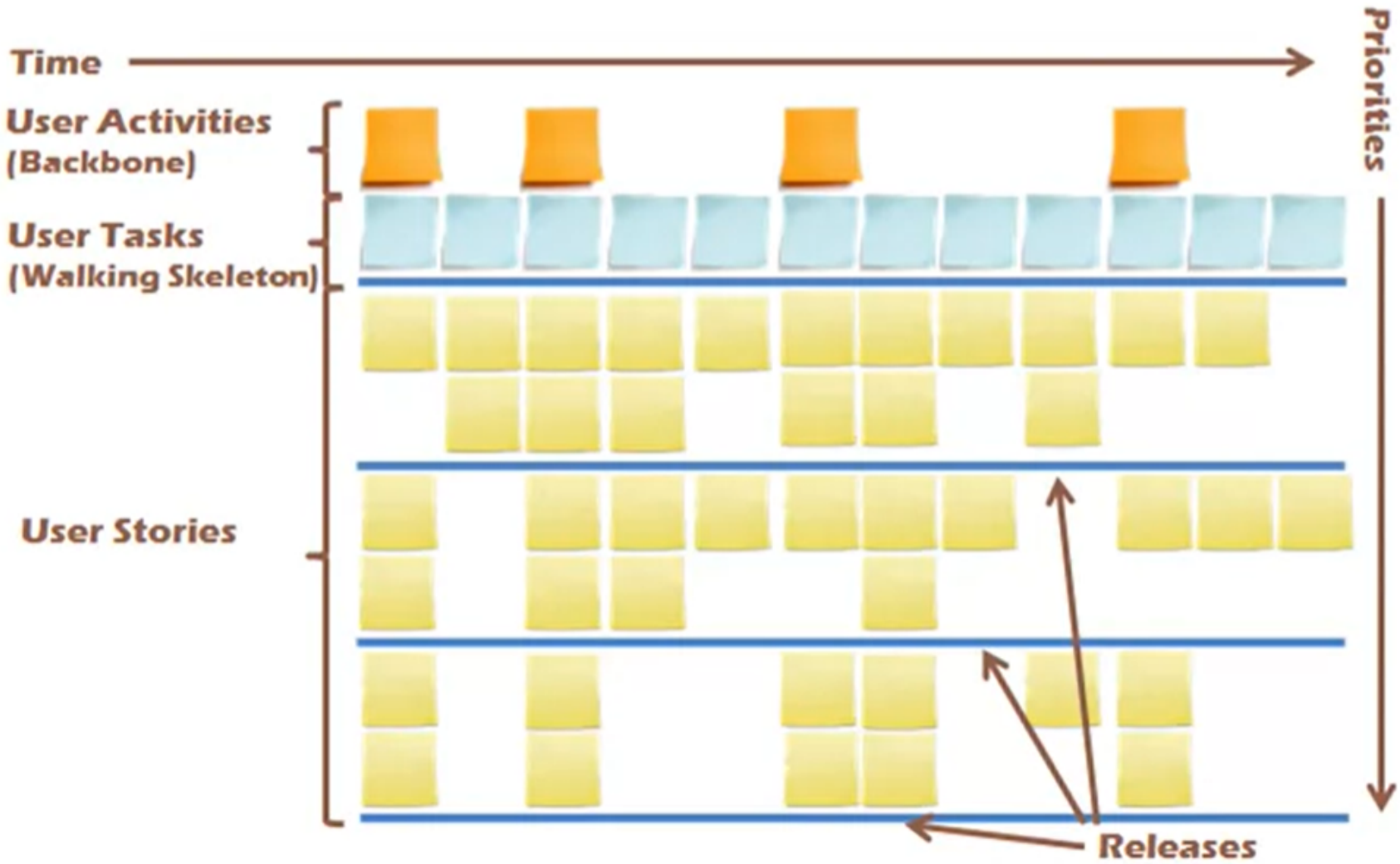
Story mapping

Popularized by Jeff Patton

Technique to:

- Discover user needs
- Organize and prioritize story backlog
- Understand communicate user needs
- Plan releases and development

Story map structure



Left to right is chronological

Top to bottom is the priority

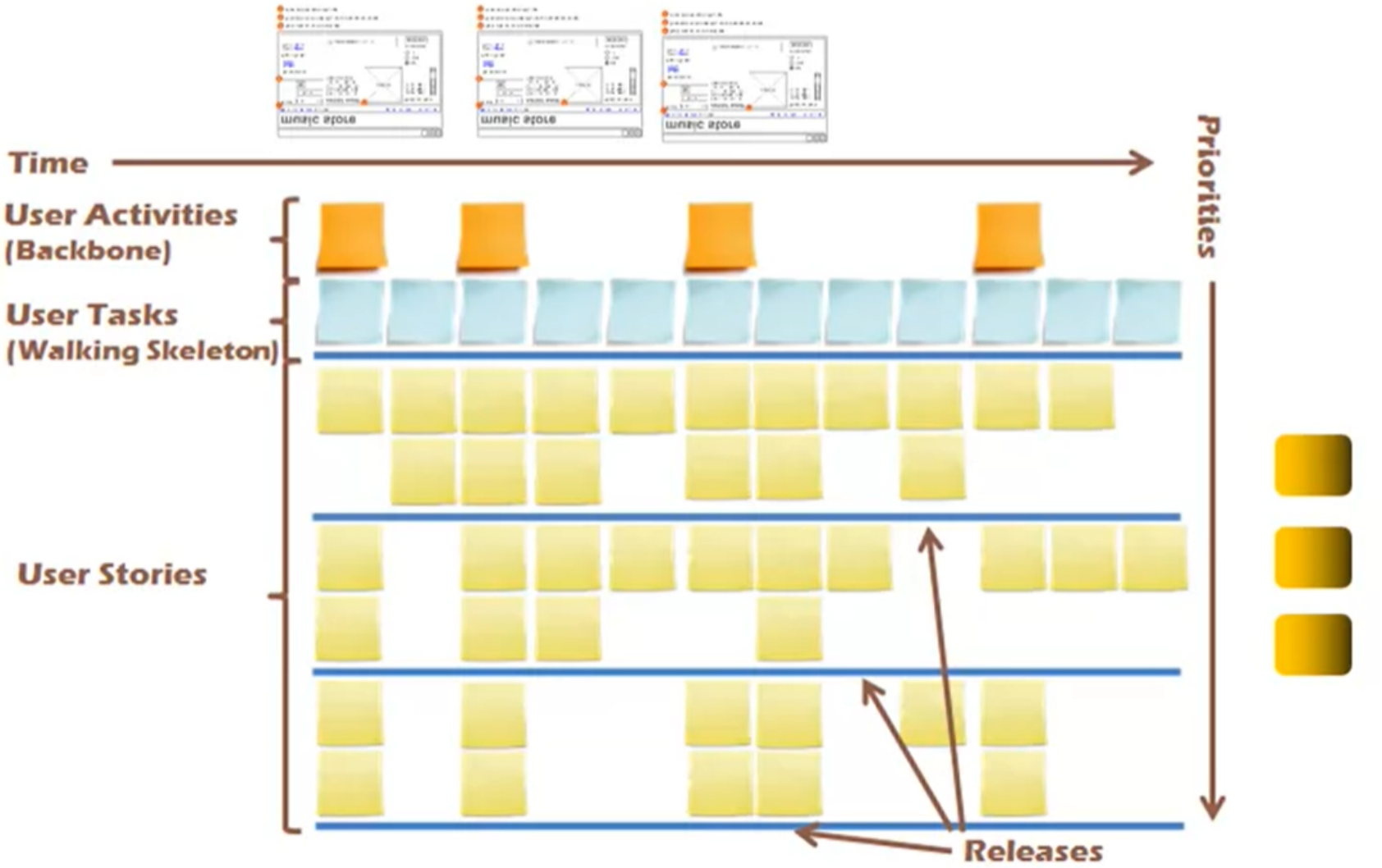
The orange represent the user activities - big things that a user wants to achieve from the system

The blue show the basic steps that the user is going to do to achieve those activities

Yellow are the user task and they are the variations: to do the blue, can do it in multiple ways and more advance

Can put screenshot/mockup on the top

Include non-functional or other requirements on the right



STORY MAP EXAMPLE...

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 benefits they get

Why - benefit to organization, what users do and how it results in benefits

Step 2: Map the big picture

Activities

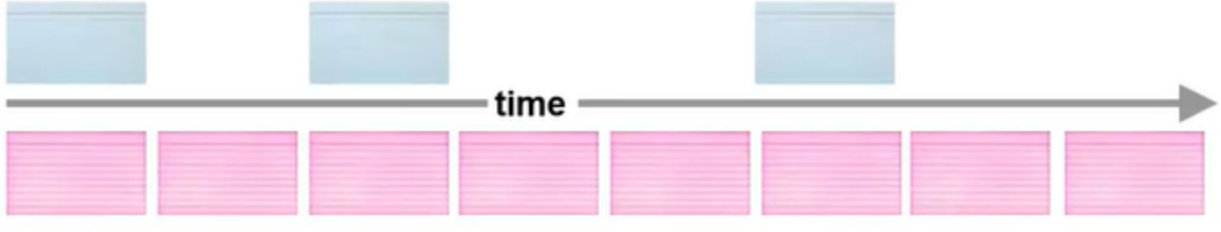
Tell a big story of the product by starting with the major user activities the system will be used for

Arrange activities left to right in the order you'd explain them to someone when asked: "what do people do with this system?"

User task

Add task-centric stories in 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

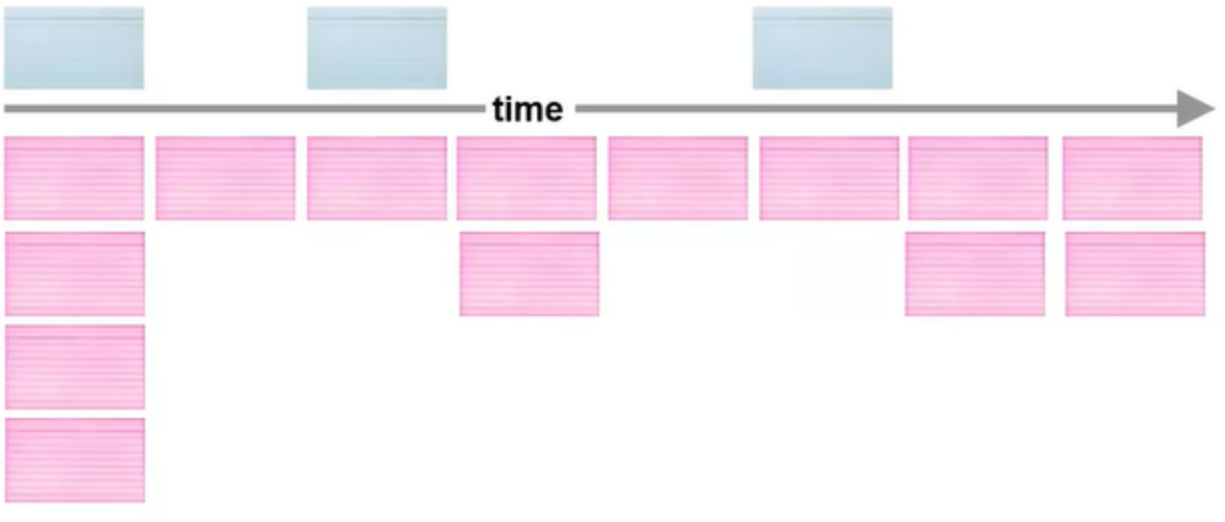


Step 3: Explore

Variations

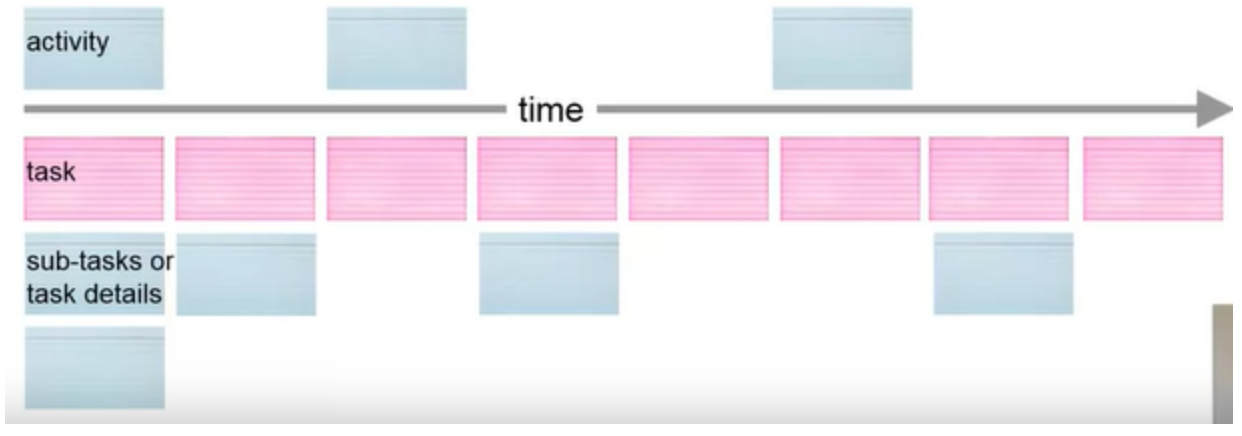
Overlap user task vertically if a user may do one of several tasks at approximately the same time

- Or → vertical
- And then → horizontal



How to capture details

Record details so they're not lost, and it helps acknowledge to participants that details are not lost
+ Consider tucking them under tasks cards to 'hide them' from the discussion



Go crazy

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 release

Slice map to holistic meaningful releases

Focus on outcome, slice away what’s not needed

For each release identify
+ Outcome and impact
+ Success criteria

Why create them?

Discover user needs - esp help discover missing pieces

Understand and communicate user needs
+ Help communicate at multiple levels
+ Help tell a story

Planning
+ Provide a useful context for prioritization
+ Plant releases in complete and valuable slices of functionality
+ Organize and prioritize story backlog

Foster co-ownership

Flexible

Agile Estimation and Planning

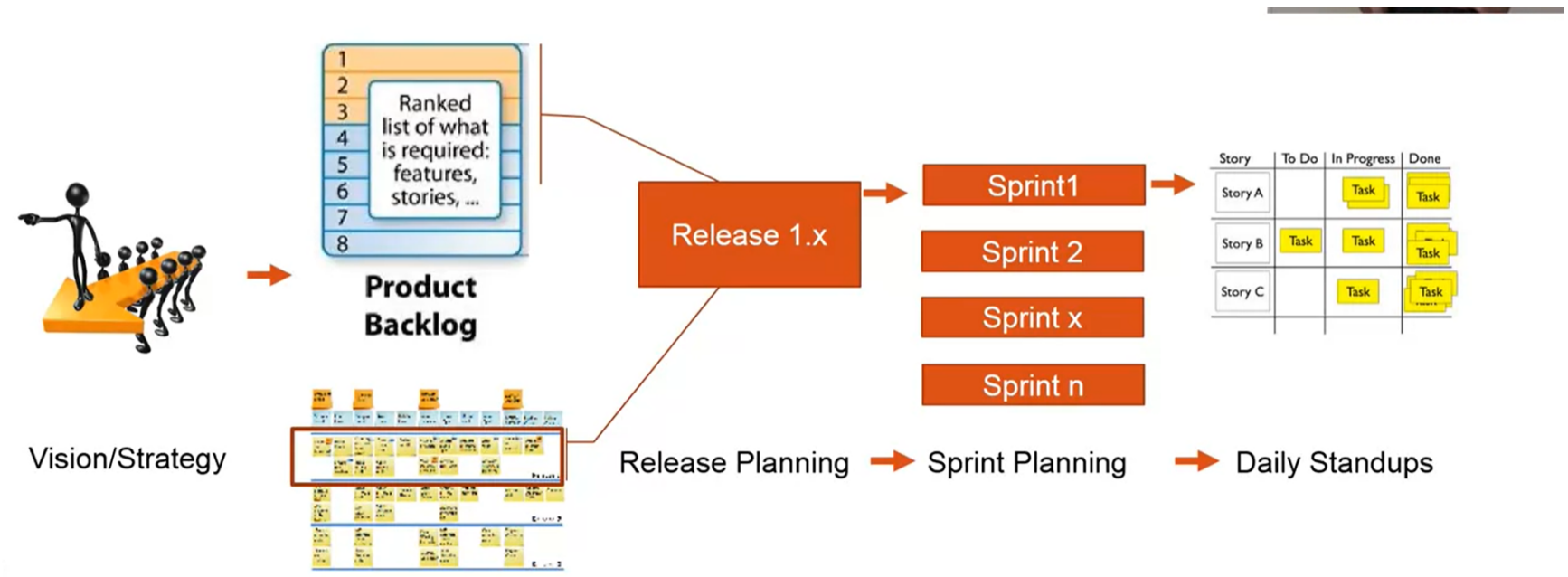
Agile Planning

Multi-level

Less upfront but frequent

Just enough, just in time

Adap and Re-plan



Agile approach to Estimation

Three concepts

- Effort vs Duration
- Accuracy vs Precision
- Relative vs Absolute

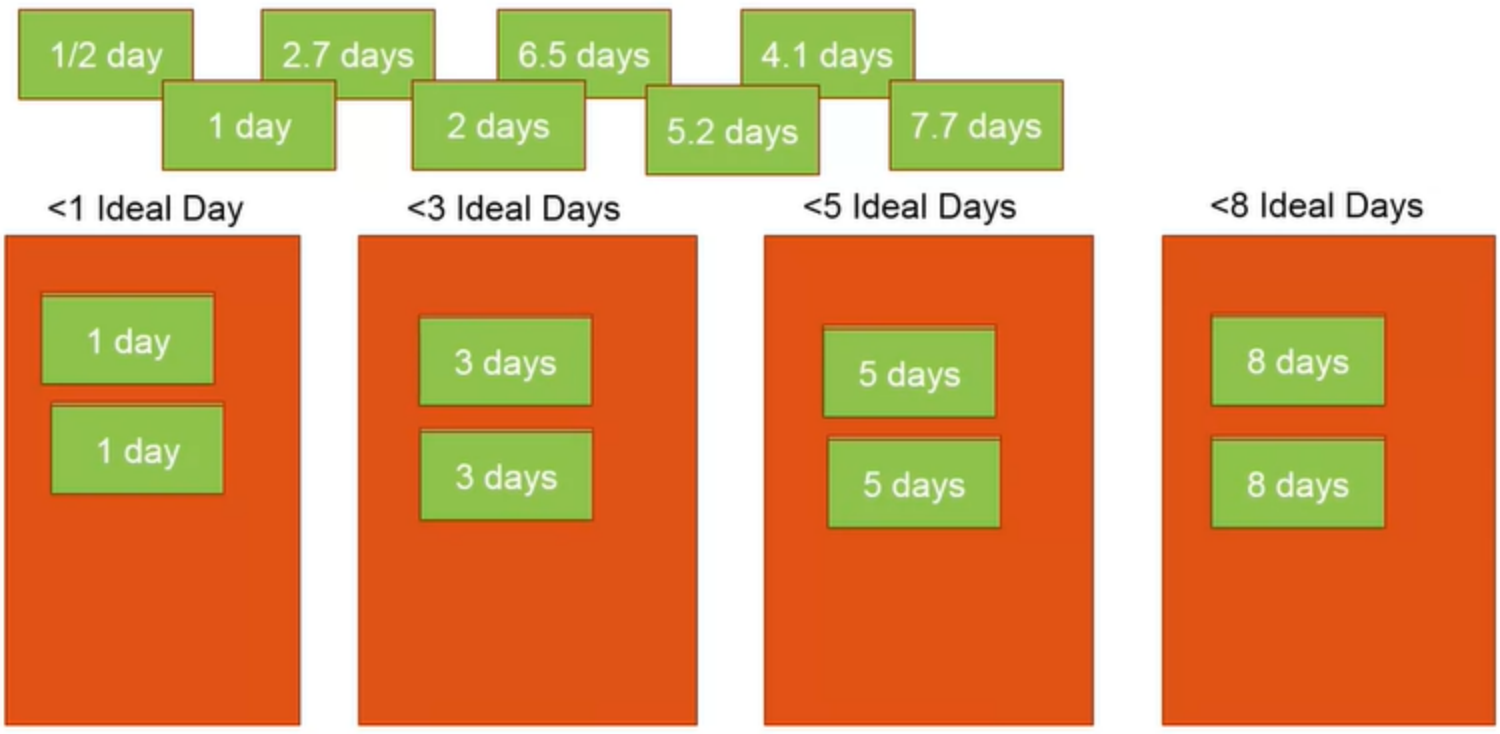
Effort vs Duration

Duration: 3 days

Effort: 1 day (0.25 + 0.5 + 0.25)

`IDEAL DAYS` How many work days will it take to complete a story if you work on in uninterrupted

Accuracy over Precision



Sizing buckets - Estimation scale

Popular Scales:
+ 1, 2, 4, 8
+ Fibonacci series: 1,2,3,5,8,13,...
+ 1, 5, 10, 20,...
+ XS, S, M, L, XL
+ Small, Medium, Large

What if I can’t 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 day
- Story A will take 4 hrs, Story B will take 2 hrs

RELATIVE VS ABSOLUTE

Story Point (Relative)	Ideal Day Buckets (Absolute)
Difficult to get accustomed to	Natural
Faster (once you get used to)	Tedious
No “My ideal days are not your Ideal Days” issue	“My ideal days are not your Ideal Days” issue
Difficult to explain outside the team or to new member	Very Natural to explain
Makes it harder for management to behave badly ☺	Management may try to drive team to ideal days = regular day

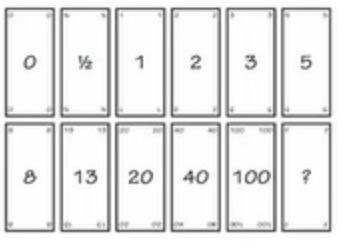
Estimation Styles and Process

ESTIMATION PROCESS

- Who Estimates?
  Development team
- How long does it takes?
  Depends on style, method, number of stories, understanding, knowledge, expertise, # of team members
- Estimation Styles
  - Simple – Free form.
  - Planning Poker
  - Card Sorting

PLANNING POKER

1. Everybody gets the estimation cards
2. Explain the story
3. Understand the story
4. Estimate and put one card down
5. Open cards
6. If consensus, move to next item
7. Else discuss variations and go back to step 3

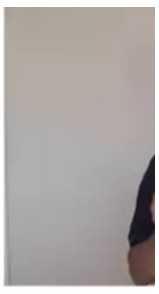


PLANNING POKER PROS AND CONS

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

CARD SORTING

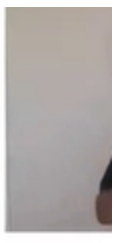
1. Place smallest(left) and Largest Story(right)
2. **Silently,** everybody start placing stories between these two cards.
   - Separate column for stories with questions
   - Stack same size stories vertically
   - Disagree? Move the story silently.

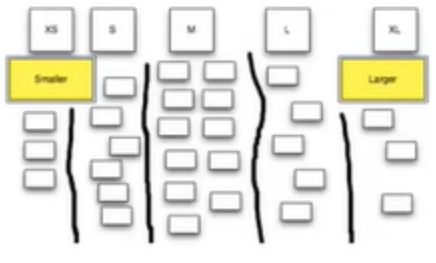


CARD SORTING

3. Discuss changes.
   - Discuss stories with question and put them on the board.
   - Take a another look
   - Discuss disagreements and move stories if needed
4. Create buckets
5. 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?

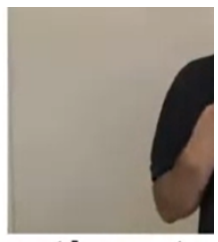
Useful for estimating large number of stories together

Velocity

Roughly, amount of work getting done in a sprint

Changes based on team, project, and many other factors

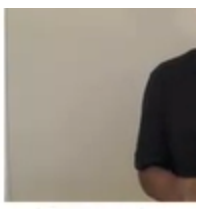
CALCULATING VELOCITY



- For a given sprint, if you finished three stories with estimate of 5, 3,10 then your velocity for that sprint is 18.
- To calculate velocity to use for planning upcoming sprint:
  - **Last Sprint Velocity** → 23
  - **Average** of last x sprints
    - Sprint 28 (Using Last 3) → Avg(25+27+23) → 25
  - **Velocity Range** of last x sprints
    - Sprint 28(Using Last 3) → 23-25

Sprint	Velocity
Sprint 21	15
Sprint 22	17
Sprint 23	54
Sprint 24	25
Sprint 25	25
Sprint 26	27
Sprint 27	23

CALCULATING VELOCITY

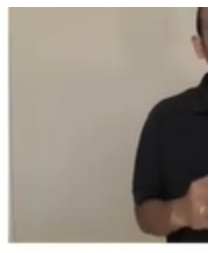


- For a given sprint, if you finished three stories with estimate of 5, 3,10 then your velocity for that sprint is 18.
- To calculate velocity to use for planning upcoming sprint:
  - **Last Sprint Velocity** → 23
  - **Average** of last x sprints
    - Sprint 28 (Using Last 3) → Avg(25+27+23) → 25
  - **Velocity Range** of last x sprints
    - Sprint 28(Using Last 3) → 23-27

- Skip Anomalies
  - Skip sprint 23 if using last 5

Sprint	Velocity
Sprint 21	15
Sprint 22	17
Sprint 23	54
Sprint 24	25
Sprint 25	25
Sprint 26	27
Sprint 27	23

ADJUSTING VELOCITY/ESTIMATES



- Calculating velocity for first sprint?
  - Take 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 next 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 get up to speed and it is difficult to predict the impact of a personal change on team's productivity.

- What if your estimates were 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

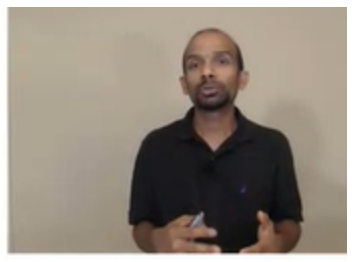
Type of release planning

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?
  1. Decide sprint length
  2. Calculate Velocity (or velocity range)
  3. Total up estimate for selected stories
  4. Total Estimate / Velocity ~= # sprints
  5. # sprints X sprint length = duration

FIXED SCOPE RELEASE



Total of story points = 120 points

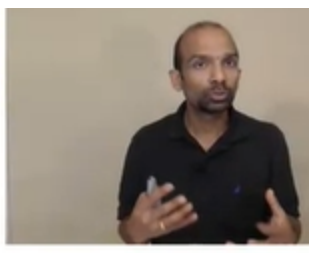
2 week sprints | Cost = 50K per week

- At high velocity, 120 / 20 = 6 sprints
- At low velocity, 120 / 15 = 8 sprints
- So it will take between 6 to 8 sprints
- Duration = 12 to 16 weeks
- Cost = 12*50k to 16*50K = 600K-800K

Sprint	Velocity
Sprint 21	15
Sprint 22	17
Sprint 23	18
Sprint 24	19
Sprint 25	15
Sprint 26	20
Sprint 27	18

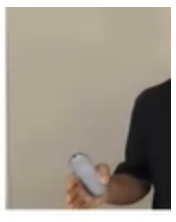
Velocity Range(last 3) = 15 to 20

HOW TO SELECT STORIES?

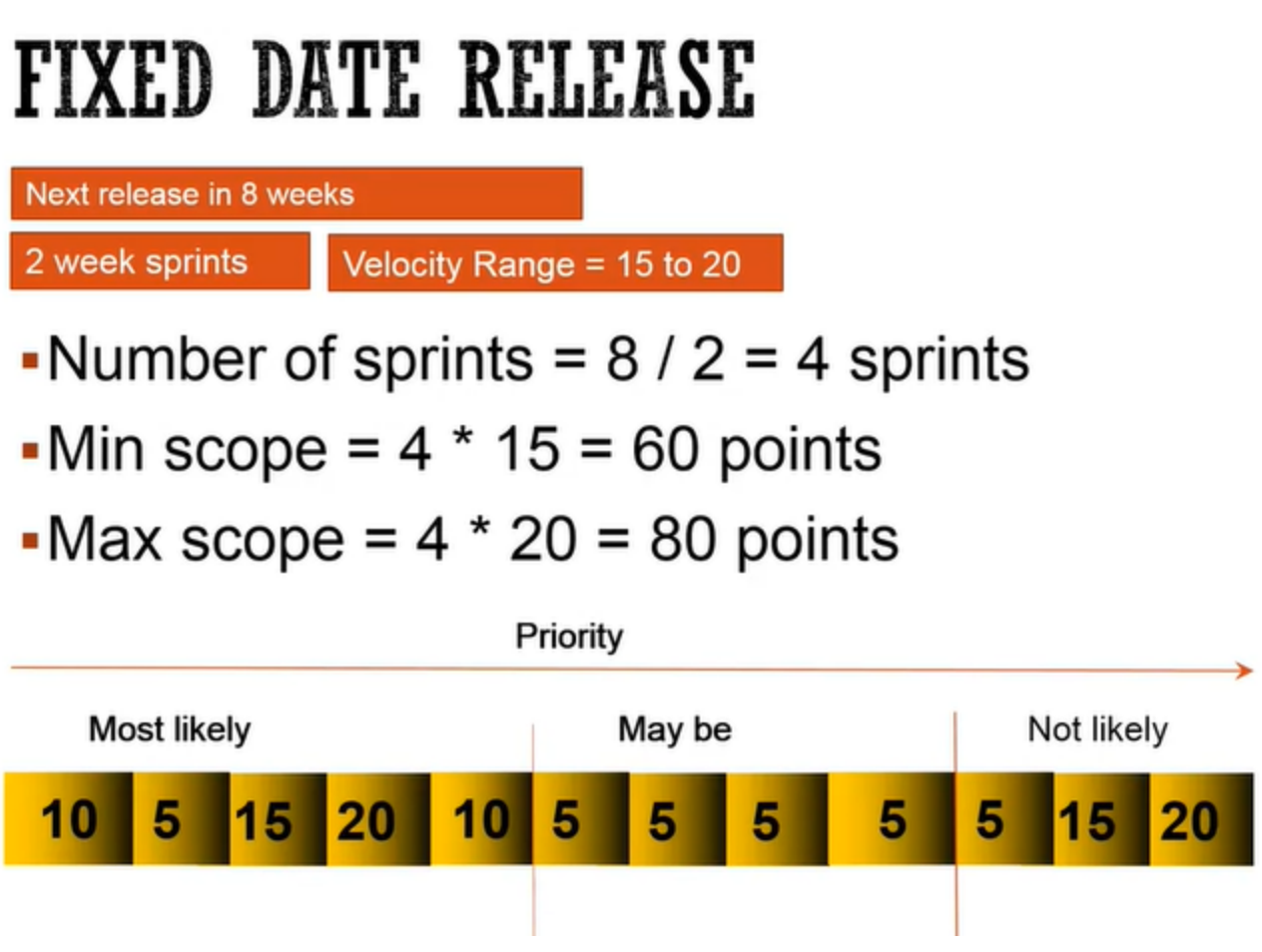


- Use story map
- Short feedback loop
- Learning
- Value to users / Market position
- Risk mitigation
- Dependencies

FIXED DATE RELEASE



- Date fixed so scope changes
- What are you going to deliver?
    1. Groom backlog (if not done so)
    2. Calculate Velocity Range
    3. Select sprint length
    4. Calculate # sprints
    5. Calculate Release Capacity = #sprint* velocity (use two times to get range)
    6. Include items from backlog (starting at top) until total point exceed points range: will have, might have.

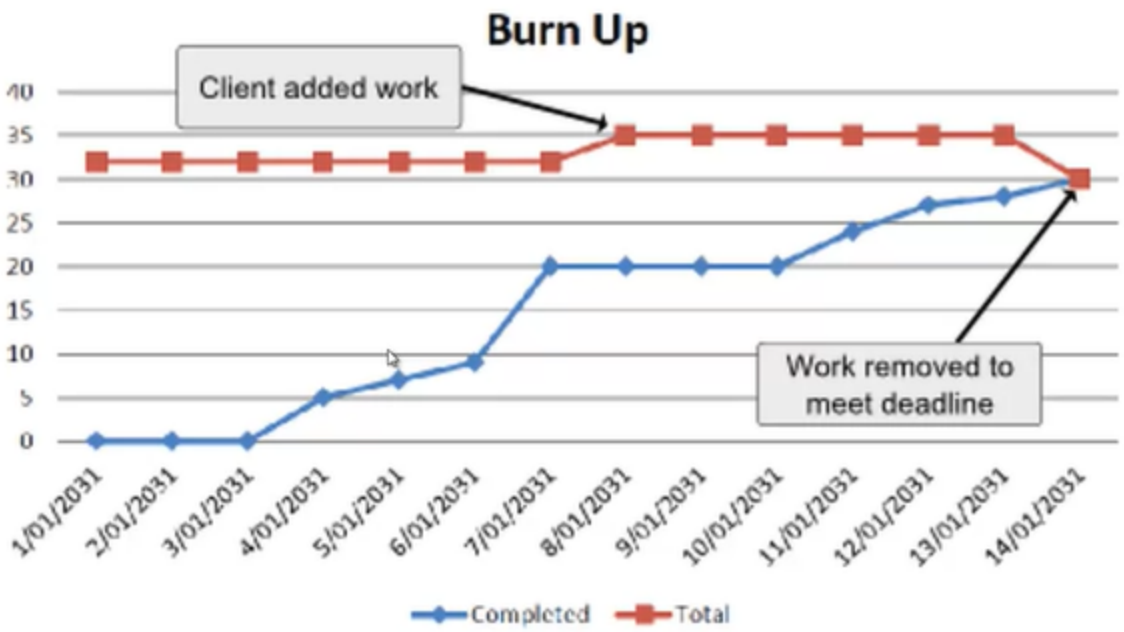


Release tracking

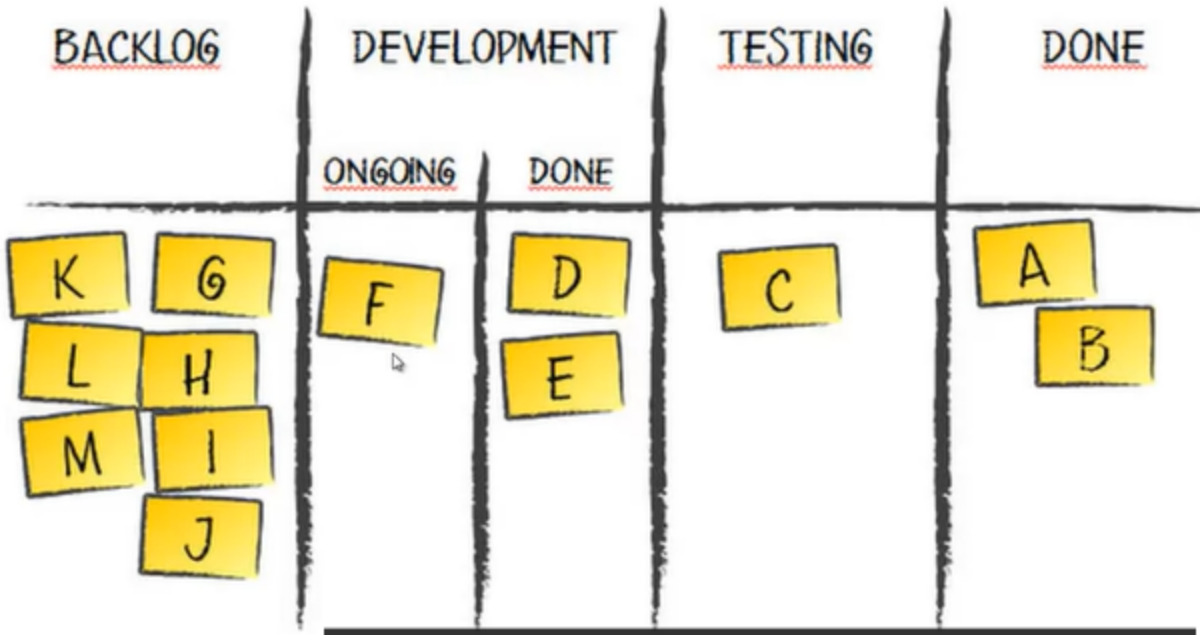
RELEASE TRACKING

1. Release Burn up
2. Story Board
3. Cumulative Flow

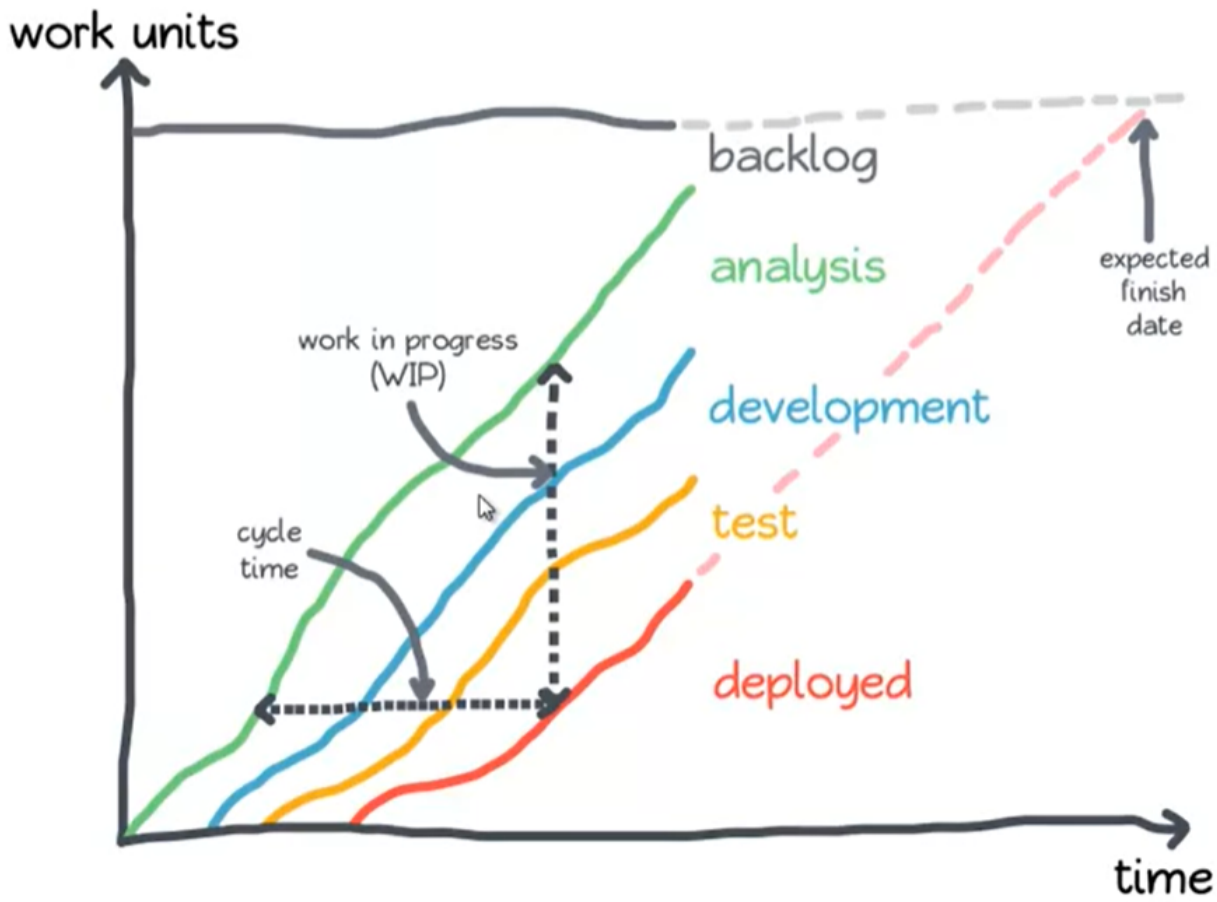
RELEASE BURNUP



STORY BOARD



CUMULATIVE FLOW DIAGRAM



Week 3

Scrum overview

Works on 1-4 week sprint

Sprint planning

We need to do some sprint planning prep work

Backlog grooming

Sometimes people call it backlog refinement

- 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
- Correcting estimates in light of newly discovered
- Splitting user stories which are high priority but too coarse grained to fit in an upcoming sprint

Once the backlog is up-to-date, next task is to select the stories for next sprint.

Selecting & preparing stories

How do you select potential stories?

- Story map
- Prioritized backlog and Velocity
- Alternatives:
  + Themes?
  + Anything team want to learn?

Make sure stories are ready to be worked upon (who, what, why, acceptance tests, any major dependencies)

Sprint planning

Two possible ways

- One step
  Select one story at the time, task it out until capacity reached
- Two step
  1. Select stories based on velocity
  2. Task out and gain confidence (some teams stop at step 1)

Sprint planning steps

1. Determine sprint capacity
2. Review sprint goal (if any)
3. Review potential stories
4. Acquire confidence: Design discussion and task out stories
5. Refine sprint goals if required
6. Make commitment
7. Put the stories and tasks on the task wall

Determine capacity

	Days – less personal time off, days spent off team	Days – other scrum activities	Hours per day	Available Effort Hours
John	9	2	5-6	35-42
Erik	4	2	5-6	10-12
Kevin	8	2	3-4	18-24
Josh	7	2	2-3	10-15
			Total	**73-103**

(9 – 2) * 5 = 35
(9 – 2) * 6 = 42

35+10+18+10 = 73
42+12+24+15 = 103

Simpler Alternatives:

- Total task estimates from last sprint
- Avg of total task estimates from last three sprints

Define sprint goal

At the end of sprint X, persona will be able to do <xyz>

Something that team can rally around

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

Acquire confidence

Not everybody has to stick around. PO to answer questions is helpful (Could be IM based or interrupt basis)

Face-to-Face conversation helps

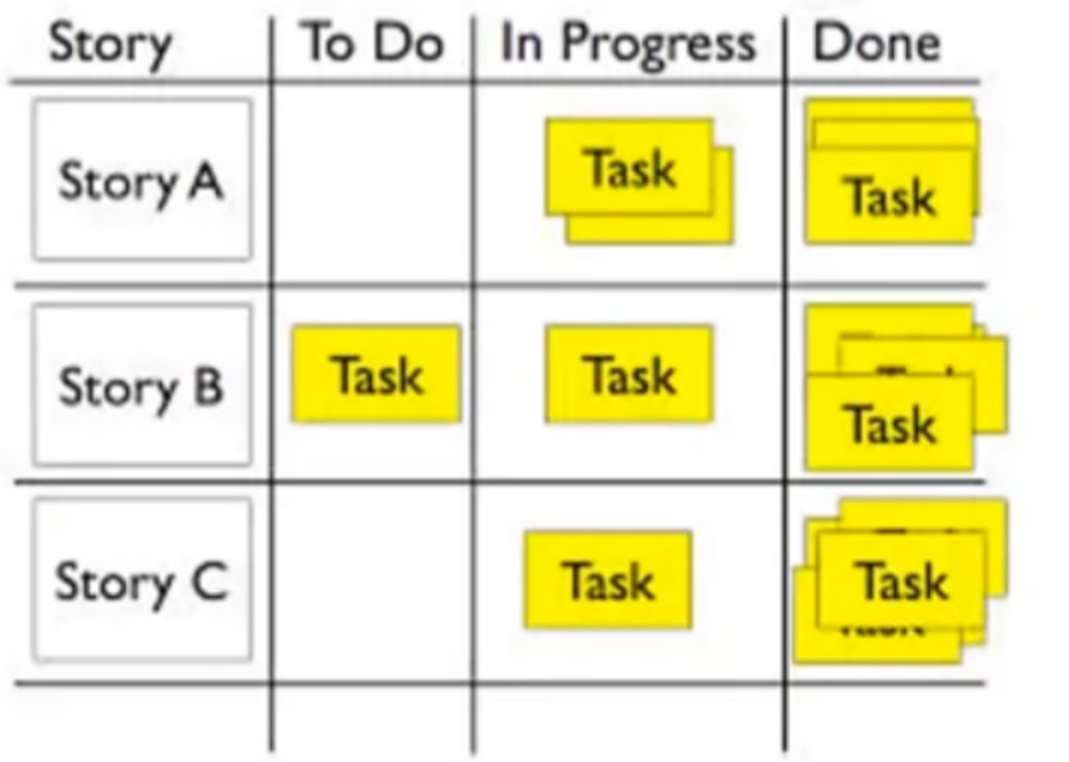
TIP: Time box activity/ discussions

TIP: Estimate tasks in Ideal days/hours

Refine goals if needed

Make commitment

Create task board



Sprint planning logistics

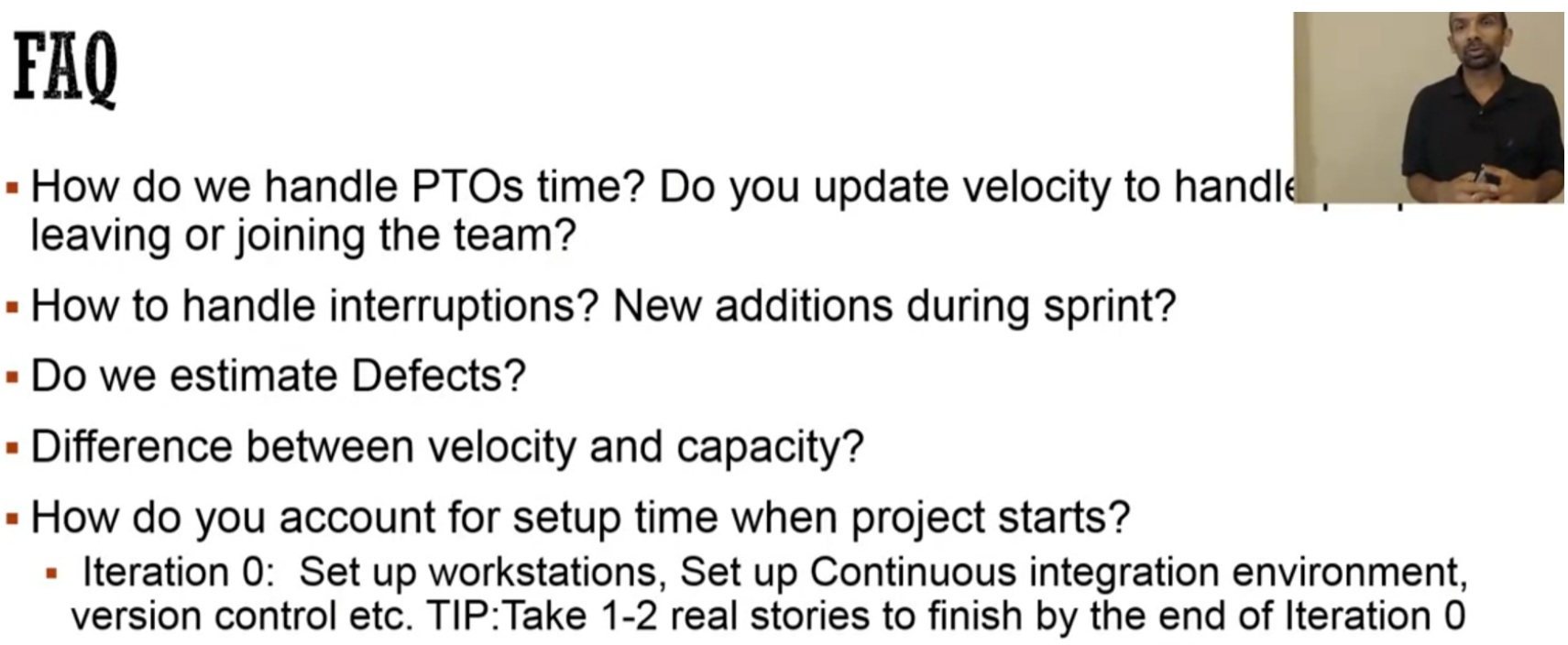
Who to invite? TIP: Setup a recurring meeting

- Core team members (PO, SM, dev, test, analyst etc)
- Users/Clients/Business people who can answer question if any

How long?

- It depends on complexity/preparedness/team's maturity but general guideline - If you include "tasking-out" activity - around 4 hrs for a 2 week sprint

> 💡 Velocity: Based on story points
> Capacity: Based on the availability of hours and days of individual team members



Sprint tracking

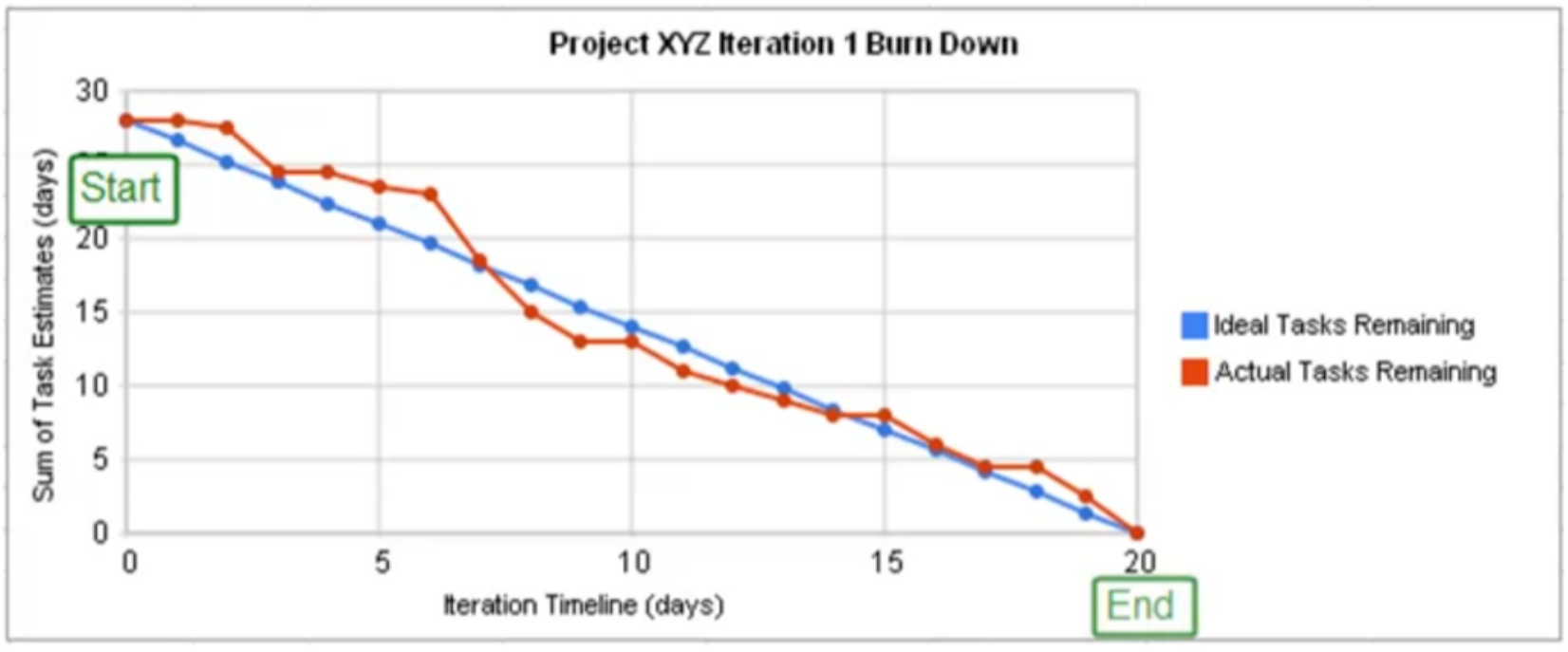
1. Burn down - work left
2. Burn up - work done + total work
3. Task board

Burn down - work left

Map the amount of work left in a sprint

X-axis is the time - days of the sprint

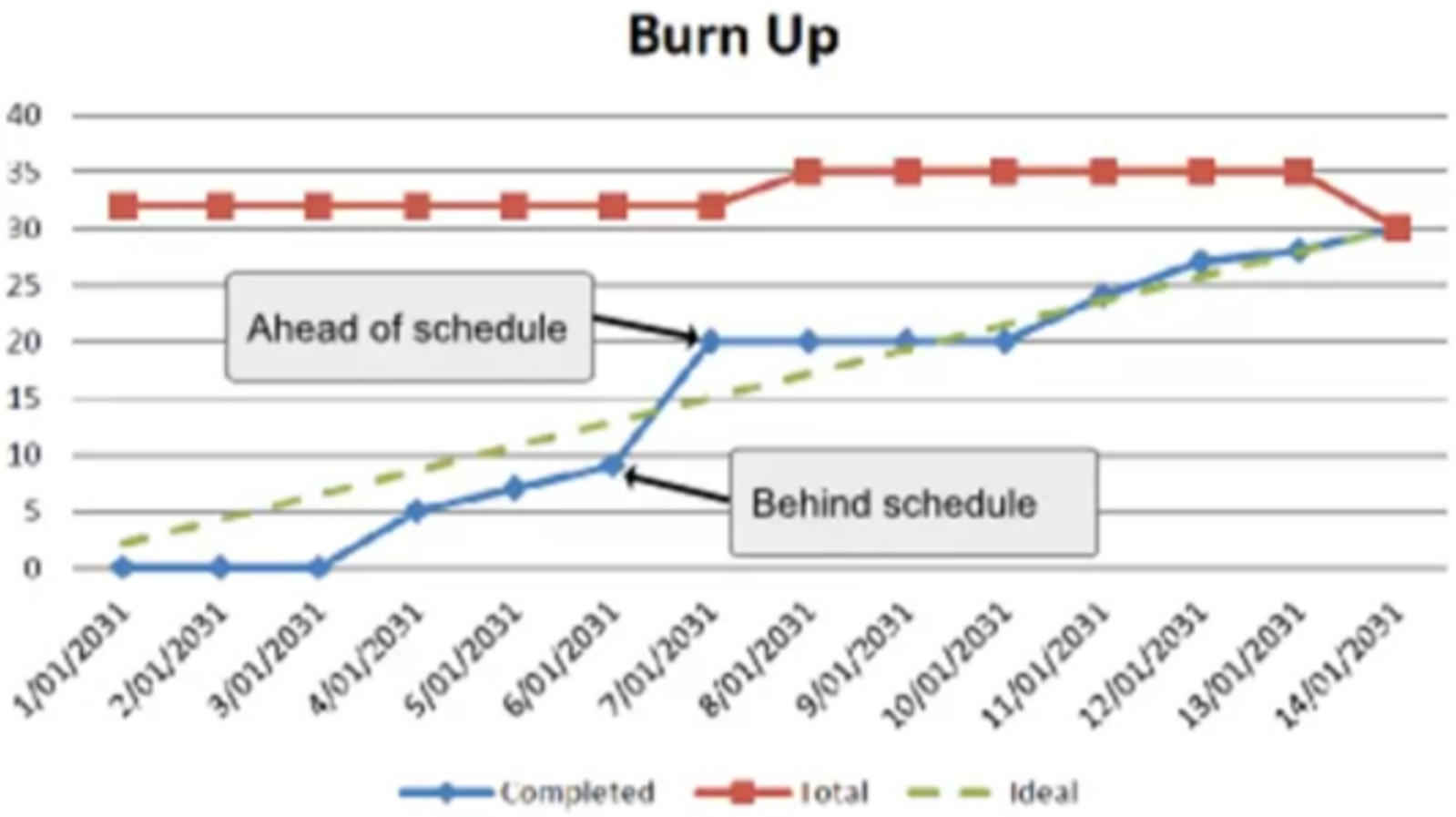
Y-axis is the sum of task estimates



Blue line is showing what is the ideal way the amount of work should go down

Red line is showing the actual task remaining

Burn up - work done + total work



Red line is the total of all the tasks that are needed to be done in the sprint.

Blue line is the number of tasks completed

Green dotted line show the ideal path

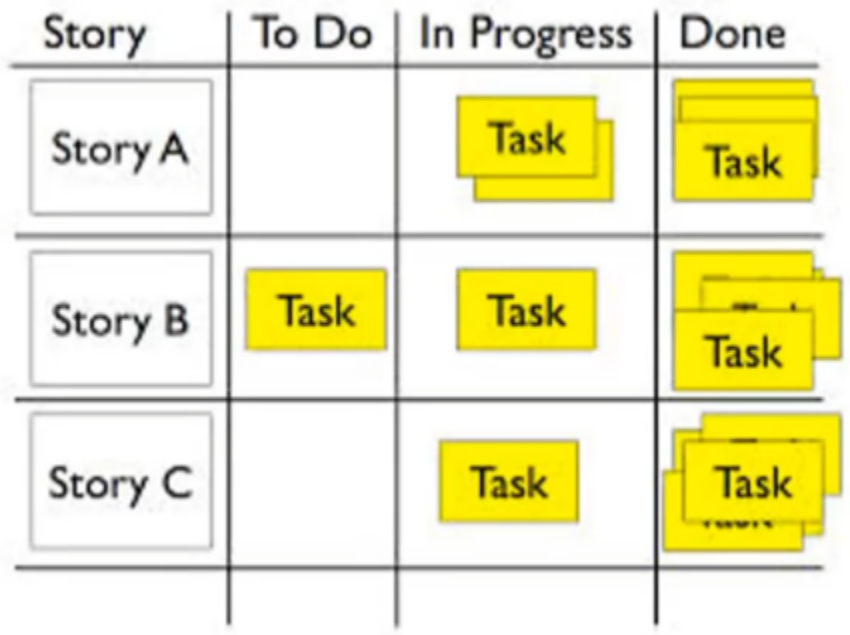
Task board

Visual way to see whether you are on track or not.

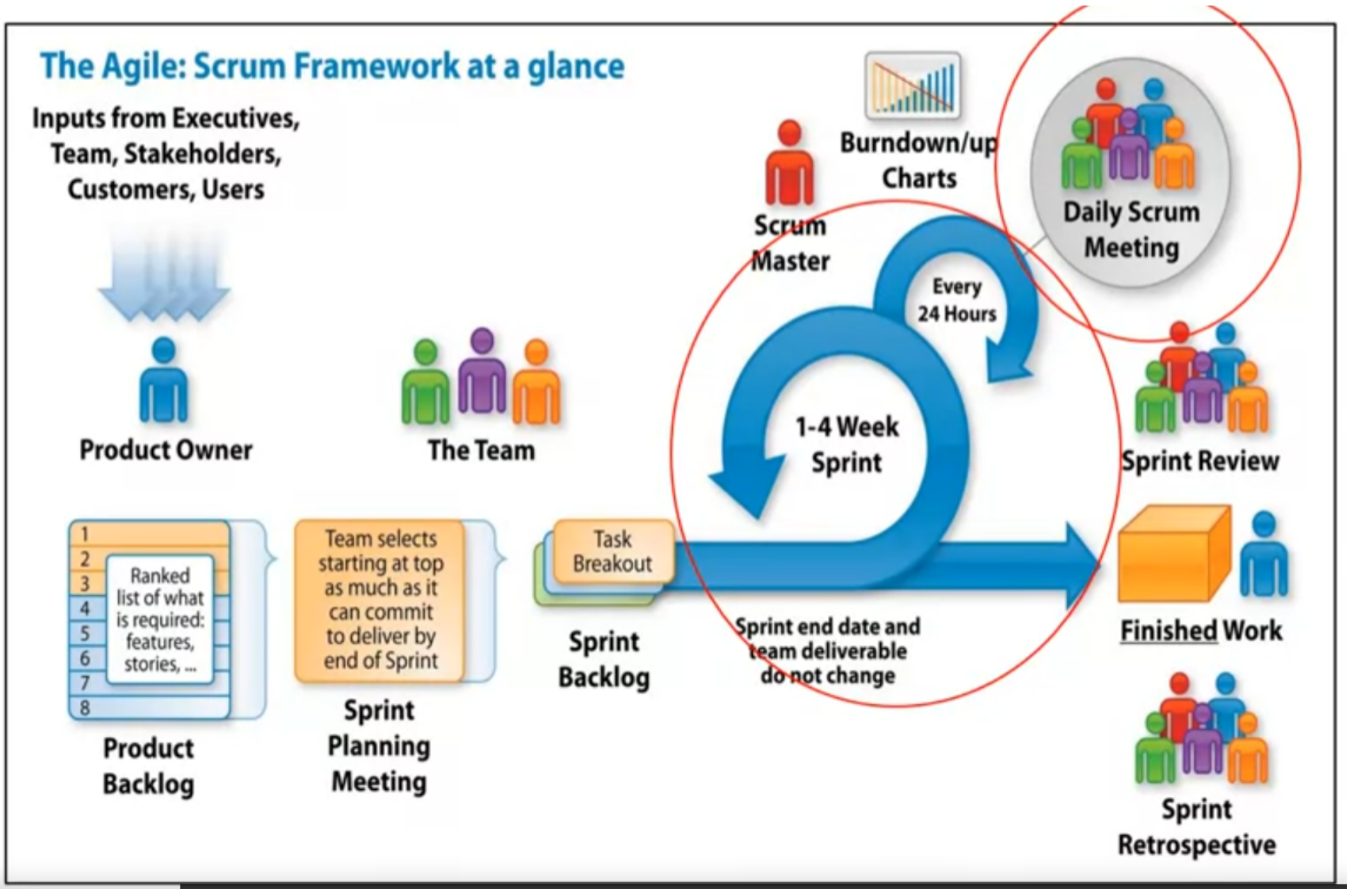
Task-board visible during standup

Definition of done

Too many in-progress smell



Sprint Execution and Daily Standups



Sprint execution

in Scrum, it's recommended that the team members pick the work from the board rather than the work getting assigned to them.

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

Generalist vs Specialists
+ Tip: skill map, pairing

Discipline... Discipline... Discipline...
+ To follow what team has decides to do

Engineering practices (continuous integration, automated deployment,...)

Daily standup

What

- Common: 3 questions
  + What did I do yesterday?
  + What am I going to do today?
  + Are there any roadblock?
- Alternatives: Work items attend / story focused stand-up

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

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 hrs

What happens?

- Summarize
- Demo
- Discuss
- Adapt

TIPS:

- Don't wait for review to show ur work
  + Early and regular feedback
- Some teams follow this rule → Demo done things only
- Make effort to get stakeholders to attend
- Presentations by individuals
- Prep work

Sprint Retrospective

Happen at the end of the sprint. The main thing the team talks about is: HOW ARE WE WORKING TOGETHER?

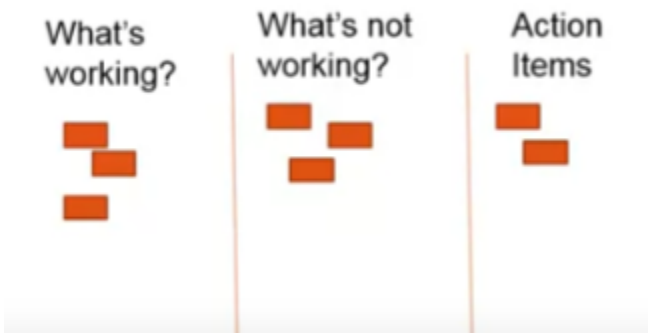
Support the process not about the product

Who? Core team

When? At the end of each iteration

Purpose? Continuous improvement

What happens? - Most common
+ What's working?
+ What's not working?
+ Action items?



Alternative: Focused retro

Steps:
+ Set the atmosphere
+ Share context
+ Identify insights
+ Determine actions
+ Clsoe the retrospective

Follow through

Change the retro format

TIPS:

- Avoid finger pointing
- Select few (may be one) action items
- Get people talking
  + Non work question
  + Round robin
  + Manager opt out
- Individual → Group → All

Assignment

As a software development professional, you will run into all kinds of projects and situations within those projects. This assignment presents you with a fictitious case study and asks you to argue in support of using agile on the project. It will also ask you to suggest changes to the organization that will be crucial for the agile project's success.

Please read the attached case study before you proceed further.

Case Study Remote Deposit Capture Project.pdf

https://d3c33hcgiwev3.cloudfront.net/_b3f5a4917c5c73955e73fa5d21583cdb_Case-Study-Remote-Deposit-Capture-Project.pdf?Expires=1649980800&Signature=Ctz~D7PZoHSSJW2JfEJV3DAIg3XJC6js6fjjGJ0CjticnmNHU5twwUiejcFDfHkpI7ZzNCxRgiBjIj5hxXjI4EaK0lzvI0C2hDYlkyvYQJDVXoH~H~02qOjcmHs3WaVMQrAZZgEpT-tURACBYhKE0hgvfrKAox90QVzkpr8LReA_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A

Blue Bank has always used the traditional waterfall method to deliver IT projects. Harold Johnson and the company leadership have suggested that you also use traditional methods to build the software. You have a different idea. You think agile would be better for this situation.

In the space below, make a case to Harold as to why we should switch to agile (explain why using agile is a better choice for this project/situation). Be sure to support your argument by referring back to the case study text.

The agile methodology approach would be the most
efficient project management method to deliver the

innovative cheque deposit capture system Blue Bank
wish to implement. It would not only allow the project team to deliver a demonstration of the current state of the solution at the end of every sprint, but also invite you and your team to come to the retrospective meeting and test the demo's and input any changes and ideas you want to the project team directly. Furthermore, since you wish to have a new mobile app, a new browser input and changes to Blue Bank's internet portal, using agile will allow for prioritisation of key parts of these three high-level objectives and allow for them to be built by different teams concurrently and in collaboration with eachother. Agile would also allow for changes to be made fast within the range of two week sprints and would be more accomdating to change than the
waterfall methodology.

Assume Harold liked your argument and agreed to proceed with agile on this project. In his reply, Harold asked for further guidance on how to move forward. Please reply back with the following:

1. **What things will have to change on how this team operates and how key stakeholders interact or engage on this project. (Hint: as you have learned, for agile to be successful there are certain prerequisites and it poses certain challenges for leadership/business stakeholders as well)**
2. **The team is new to Agile and based on prerequisites for agile to be successful, please specify if you and your team need any help or training etc. If none, please state that.**

1. The team will need to interact more horizontally and with each other. This means constant collaboration and sharing of ideas, instead of working alone in the office on different floors. We would suggest moving to the same floor to increase collaboration. The key stakeholders would also need to be involved with all steps of the project, providing feedback, input and recommendations to the project team. The leaders will need to interact with staff more for this project to be of the highest success.

2. The team and I would need training in mobile development, performance testing and user experience training. However, much of this can be done during sprint times to allow for a more efficient and cheaper overall project. Also a short session on key agile processes such as automated testing and continuous integration will be very useful to the team.

Harold is pretty impressed by your response and has approved all of your recommendations in previous two responses. He then re-iterated the key criteria for success and asked you if he and his management team should expect anything different from this project in terms for tracking and status updates.
Are there any project constraints laid out in the case study that need to be changed/managed since you will be using agile and, in agile, planning, estimation and tracking happens differently?

Hint: The case study implies certain project constraints/expectations around cost, timeline, other factors that need to managed/changed.

By implementing agile, one must note that the timeframe will be need to altered fortnightly in order to accommodate for the changes at the end discovered in the two week sprint retrospective meetings. We, of course will use financial planning to stick to the budget as best as possible, but it must be noted that during agile, changes occur frequently which are welcomed but can sometimes drive up the overall cost of the project. By managing this budget and the processes we use, we aim to sick within the estimates given prior.

Week 4

XP Overview

XP - eXtreme Programming

Introduced in 1996

The word extreme came from, like, if something is good, let's take it to the extreme.

If a developer is done with their code, they get it reviewed by another developer in the team to say whether it is meeting the quality, whether it is doing what it is supposed to do, and is it implementing the functionality correctly?

What is XP all about?

Engineering practices

Social change

Lightweight

Adaptive to changing requirements

In concrete terms, XP defines a set of
+ Values
+ Principles
+ Practices

Controversy?

Prescriptive

Pair programming

Incremental design

Scalability

XP Values

The values are Simplicity, Communication, Courage, Feedback, and 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

Everyone 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 don’t 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 can’t adapt

Feedback comes in many forms (code - are the test running, are they successful, is the code solving the prob it’s supposed to do?; feedback from the customer for the delivered software;...)

Respect

Foundation of previous four

Respecting each other is key for XP to succeed

No one is intrinsically worth more than anyone else

XP Practices

XP practices overview

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

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 (not having a member as part of multiple team at the same times, it causes lack of focus and multitasking)

Informative workspace

Get an idea in 15 seconds

Dynamic information

Cleanliness and order

Energized work

Working hours - productive

Sick - stay out - rest and get well

Incremental improvement

Pair programming

Two people working together
+ Work alone when needed

Benefits

- Keep each other 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 practice

Stories

Unit of functionality

Stories are flexible

Estimate early

Keep stories visible - don’t computerize

Weekly cycle

Plan weekly
+ Review progress

+ Select week’s worth 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

Pick a quarter’s worth of stories to address those themes

Identify bottlenecks, especially those controlled outside the team

Slack

Build some slack into the process

How?

- Lower priority tasks that can be skipped
- One week every 8th week as geek week
- 20% time for programmer-chosen tasks

Avoid aggressive commitments

Ten minute build

Build and run all the tests with in 10 mins

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 the developer has made the change, a separate process integrates that code with the rest of the code and notifies the developer if there is anything broken.

Synchronous integrations
+ the developer makes a change and constantly integrates with the rest of the system and waits for the feedback before continuing with the next functionality.

Test first programming

Write test - Run tests to see it fail → Write code → Unit test pass (`TDD - Test-driven development`)

Benefits
+ Avoid scope creep
+ Build trust in team members
+ Reduce coupling

Incremental design

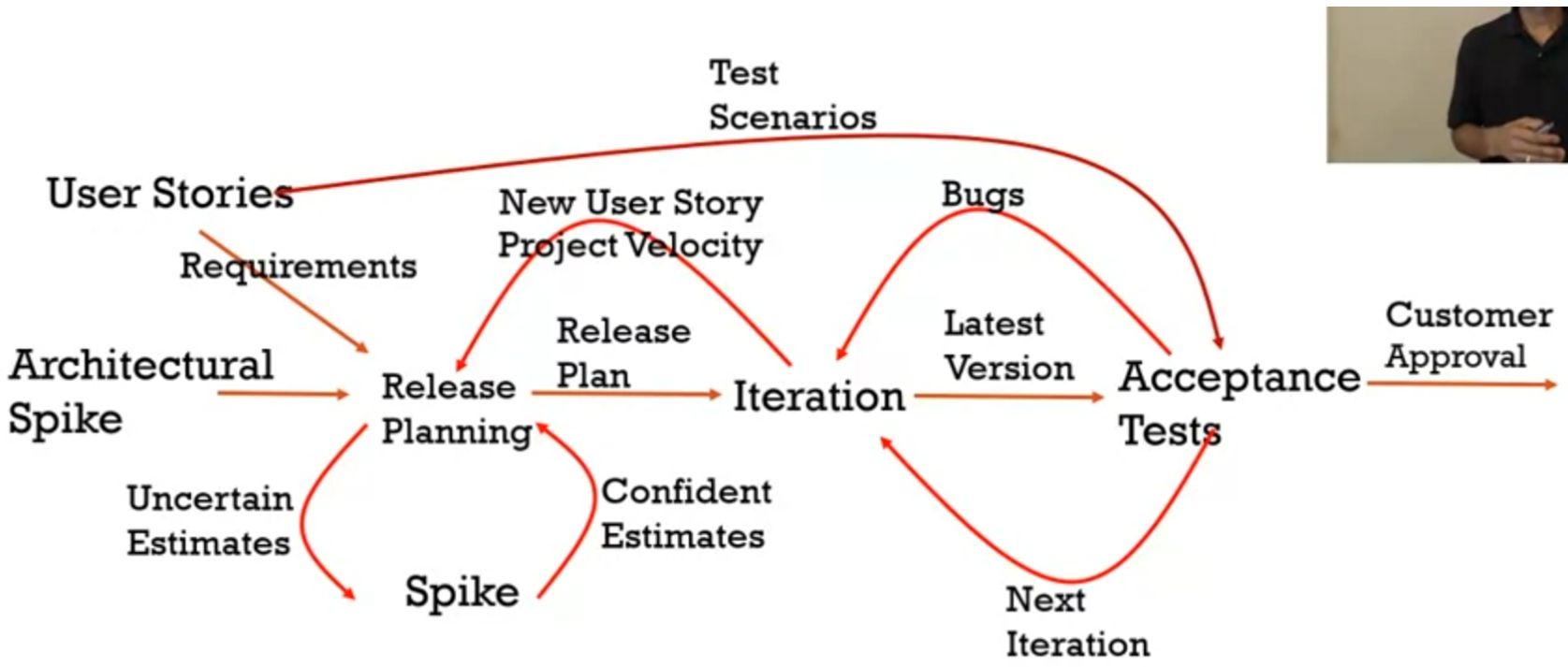
Invest a little in design everyday

Excess complexity → Refactor

Architecture emerges over time

XP Process Model

In XP, process starts with release planning where you plan next release, which means selecting what team will deliver as part of the next release

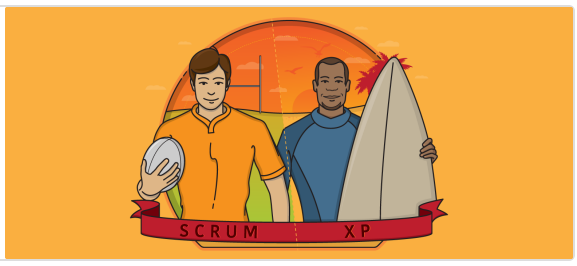


Scrum vs XP

Differences Between Scrum and Extreme Programming

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.

https://www.mountaingoatsoftware.com/blog/differences-between-scrum-and-extreme-programming



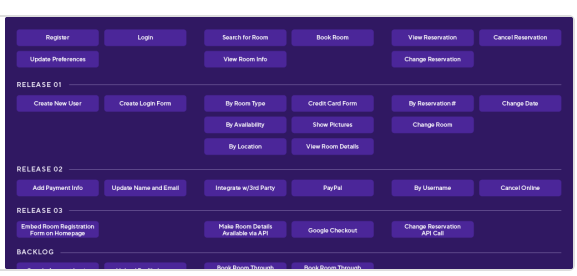
Scrum	XP
Iteration (sprint) that from 2 to 4 weeks	Iteration that are 1 or 2 weeks long
Do not allow changes into their sprints	Much more amenable to change within their iterations
The Scrum product owner prioritizes the product backlog but the team determines the sequence in which they will develop the backlog items.	Work in a strict priority order. Features to be developed are prioritized by the customer and the team is required to work on them in that order
Scrum doesn’t prescribe any engineering practices	XP does

Assignment: Story mapping

Story Mapping

Story mapping is a method for arranging user stories to create a more holistic view of how they fit into the overall user experience. Arranged on a horizontal axis, the fundamental steps of the customer journey (sometimes labeled as epics, sometimes not) are arranged in chronological order based on

https://www.productplan.com/glossary/story-mapping/



Mapping User Stories in Agile

In traditional product-development processes, teams often rely on wasteful and lengthy business requirements documents and functional design specifications to move from a vison for a digital product to outlining what it should include and how it should work. Instead of having an ongoing conversation

https://www.nngroup.com/articles/user-story-mapping/