

Agile Foundation and Introduction to Scrum

Divay Makkad
2017





Rules of the game

- Training Schedule
- Attendance All Sessions
- Client Communication
- Assessment iCompass
- Be careful while answering questions
- Please be on time



Agile



Heads Up Golf (Baseline)

The objective of the game is to highlight the benefits of iterative and incremental development.

Instructions:

1. Listen to the instructions carefully when asked to move from point x to point y
2. Start only when the instructor says GO
3. Pause if the instructor says PAUSE
4. Stop when instructor says STOP
5. Start from any point on the dotted rectangle for the designated flag.
6. There will be a total of two rounds in this baseline activity
7. When moving from point x to Point y, draw dotted line for the path like this - - -
8. Wandering into Water Hazard (WH) would mean a penalty 5 points (-5)
9. Touching the Bunker (B) will invite a penalty of 3 points (-3)
10. Reaching the target fetches 10 points.
11. There are no winners or losers. Only learners.





Why Agile

Domain-Driven
Scrum-Master
Domain-Driven

We are losing the relay race.

“The... ‘relay race’ approach to product development...may conflict with the goals of maximum speed and flexibility. Instead a holistic or ‘rugby’ approach—where a team tries to go the distance as a unit, passing the ball back and forth—may better serve today’s competitive requirements.”

Hiroataka Takeuchi and Ikujiro Nonaka, “The New New Product Development Game”, *Harvard Business Review*, January 1986.

Continuous



Why Agile

- **Agile software development** is a group of software development methods
- Empirical process based on Iterative and incremental development approach
- Self-organizing, cross-functional communicative teams.
- Encourages rapid and flexible response to **change**.
- **Adaptive** planning, **evolutionary** development & delivery, **time-boxed iterative** approach
- Promotes **foreseen interactions** throughout the development cycle.
- The *Agile Manifesto* established its principles and values in 2001



Agile Values

“We are uncovering better ways of developing software by doing it and helping others do it.” Through this work we have come to the value:

Individuals & Interactions Over Processes & Tools

Working Software Over Comprehensive Documentation

Customer Collaboration Over Contract Negotiation

Responding to Change Over Following a Plan

That is, While there is value in the items on the **right, we value the items on the **left** more.**

Kent Beck | Mike Beedle | Arie van Bennekum | Alistair Cockburn | Ward Cunningham | Martin Fowler | James Grenning | Jim Highsmith | Andrew Hunt | Ron Jeffries | Jon Kern | Brian Marick | Robert C. Martin | Steve Mellor | Ken Schwaber | Jeff Sutherland | Dave Thomas

© 2001, the above authors this declaration may be freely copied in any form, but only in its entirety through this notice.



Value 1: Individuals and Interactions over Processes and Tools

- **Strong players:** a must, but can fail if don't work together.
- **Strong player:** not necessarily an 'ace;' work well with others!
 - Communication and interacting is **more important** than raw talent.
- **'Right' tools** are vital to smooth functioning of a team.
- **Start small.** Find a free tool and use until you can demo you've outgrown it.
Don't assume bigger is better. Start with white board; flat files before going to a huge database.
- **Building a team** more important than **building environment**.
 - Some managers build the environment and expect the team to fall together.
 - Let the team build the environment on the **basis of need**.



Value 2: Working Software over Comprehensive Documentation

- **Code** – not ideal medium for communicating rationale and system structure.
 - Team needs to produce human readable documents describing system and design decision rationale.
- **Too much documentation is worse than too little.**
 - Take time; more to keep in sync with code; Not kept in sync? it is a lie and misleading.
- **Short rationale and structure document.**
 - Keep this in sync; Only highest level structure in the system kept.



Value 3: Customer Collaboration over Contract Negotiation (1 of 2)

- Not possible to describe software requirements up front and leave someone else to develop it within cost and on time.
- Customers cannot just cite needs and go away
- Successful projects require **customer feedback on a regular and frequent basis** – and not dependent upon a contract or SOW.



Value 3: Customer Collaboration over Contract Negotiation (2 of 2)

- **Best contracts are NOT** those specifying requirements, schedule and cost.
 - Become meaningless shortly.
- **Far better are contracts that govern the way the development team and customer will work together.**
- Intense collaboration with customer and a contract that governs collaboration rather than details of scope and schedule
 - Details ideally **not** specified in contract.
 - Rather contracts could pay when a block passed customer's acceptance tests.
 - With frequent deliverables and feedback, acceptance tests never an issue.



Value 4: Responding to Change over Following a Plan (1 of 2)

- **Our plans and the ability to respond to changes is critical!**
- **Course of a project cannot be predicted far into the future.**
 - Too many variables; not many good ways at estimating cost.
- **Tempting** to create a PERT or Gantt chart for whole project.
 - This does not give novice managers control.
 - Can track individual tasks, compare to actual dates w/planned dates and react to discrepancies.
 - But the structure of the chart will degrade
 - As developers gain knowledge of the system and as customer gains knowledge about their needs, some tasks will become unnecessary; others will be discovered and will be added to 'the list.'
 - **In short, the plan will undergo changes in *shape*, not just dates.**

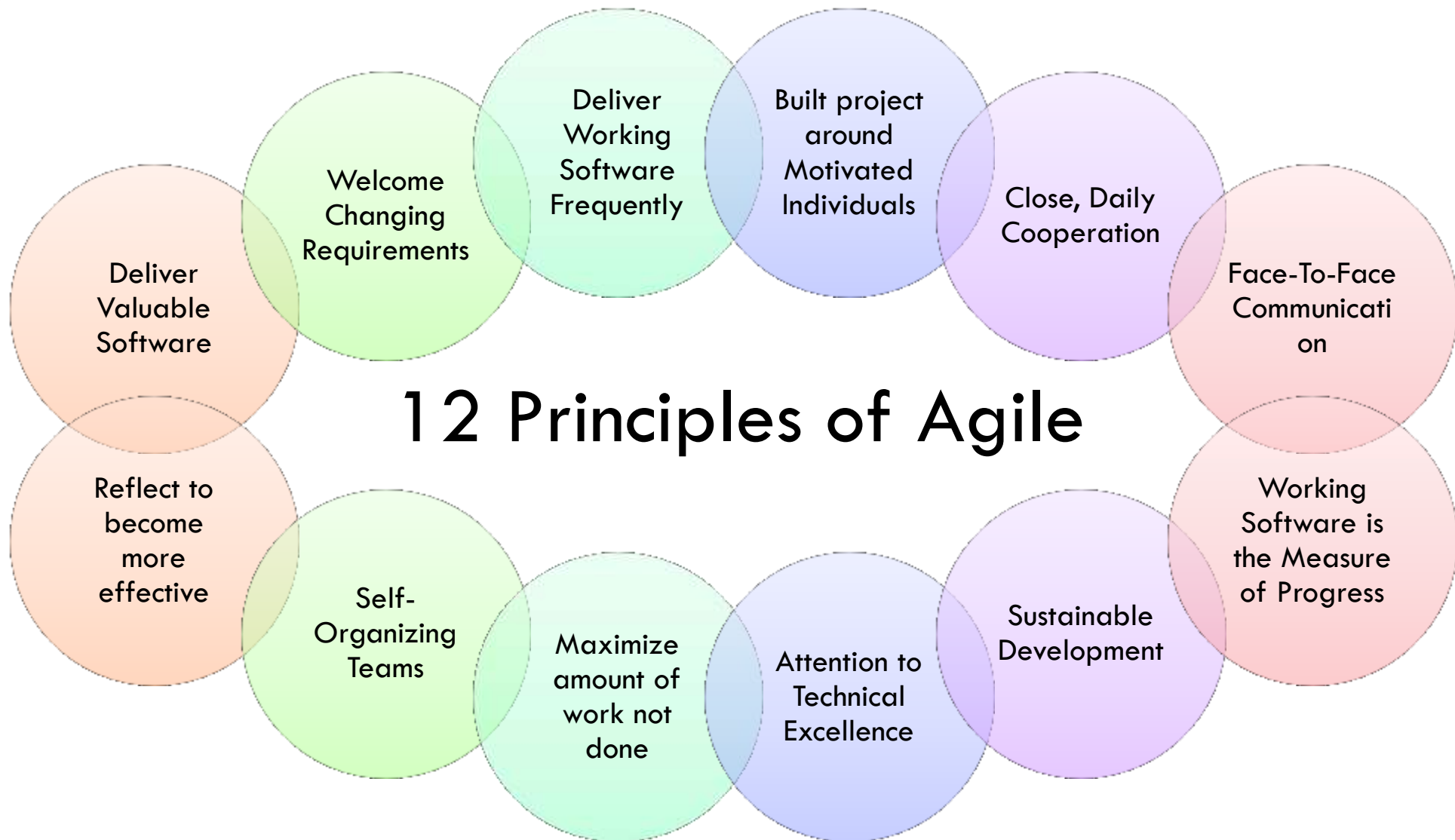


Value 4: Responding to Change over Following a Plan (2 of 2)

- **Better planning strategy** – make detailed plans for the next few weeks, very rough plans for the next few months, and extremely crude plans beyond that.
- Need to know what we will be working on the next few weeks; roughly for the next few months; a vague idea what system will do after a year.
- **Only invest in a detailed plan for immediate tasks;** once plan is made, difficult to change due to momentum and commitment.
- But rest of plan remains flexible. The lower resolution parts of the plan can be changed with relative ease.



Agile Principle





Principle 1: Our Highest Priority is to Satisfy the Customer through Early and Continuous Delivery of Valuable Software


Number of practices have significant impact upon quality of final system:

- **1. Strong correlation between quality and early delivery of a partially functioning system.**
 - The less functional the initial delivery, the higher the quality of the final delivery.
- **2. Another strong correlation exists between final quality and frequently deliveries of increasing functionality.**
 - The more frequent the deliveries, the higher the final quality.
- **Agile processes deliver early and often.**
 - Rudimentary system first followed by systems of increasing functionality every few weeks.
 - Customers may use these systems in production, or
 - May choose to review existing functionality and report on changes to be made.
 - Regardless, they must provide meaningful feedback.




Principle 2: Welcome Changing Requirements, even late in Development. Agile Processes harness change for the Customer's Competitive Advantage.

- This is a statement of **attitude**.
- Participants in an agile process are **not afraid** of change.
 - Requirement changes are good;
 - Mean team has learned more about what it will take to satisfy the market.
- Agile teams work to keep the **software structure flexible**, so requirement change impact is minimal.
- More so, the **principles of object oriented** design help us to maintain this kind of flexibility.




Principle 3: Deliver Working Software Frequently (From a couple of weeks to a couple of months with a preference to the shorter time scale.)

- **We deliver working software.**
 - Deliver early and often.
 - Be not content with delivering bundles of documents, or plans.
 - Don't count those as true deliverables.
- **The goal of delivering software that satisfies the customer's needs.**




Principle 4: Business People and Developers Must Work Together Daily throughout the Project.

- For agile projects, there must be **significant** and **frequent interaction** between the
 - customers,
 - developers, and
 - stakeholders.
- An agile project must be continuously guided.



Principle 5: Build Projects around Motivated Individuals. (Give them the environment and support they need, and trust them to get the job done.)

- An agile project has people the most important factor of success.
- All other factors, process, environment, management, etc., are considered to be second order effects, and are subject to change if they are having an adverse effect upon the people.
- Example: if the office environment is an obstacle to the team, change the office environment.
- If certain process steps are obstacles to the team, change the process steps.




Principle 6: The Most Efficient and Effective Method of Conveying Information to and within a Development Team is face-to-face Communications.

- In agile projects, developers talk to each other.
- The primary mode of communication is conversation.
- Documents may be created, but there is no attempt to capture all project information in writing.
- An agile project team does not demand written specs, written plans, or written designs.
- They may create them if they perceive an immediate and significant need, but they are not the default.
- The default is conversation.



Principle 7: Working Software is the Primary Measure of Progress

- Agile projects measure their progress by measuring the amount of working software.
- Progress not measured by phase we are in, or
- by the volume of produced documentation or
- by the amount of code they have created.
- Agile teams are 30% done when 30% of the necessary functionality is working.



Principle 8: Agile Processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

- An agile project is not run like a 50 yard dash; it is run like a marathon.
- The team does not take off at full speed and try to maintain that speed for the duration.
- Rather they run at a fast, but sustainable, pace.
- Running too fast leads to burnout, shortcuts, and debacle.
- Agile teams pace themselves.
- They don't allow themselves to get too tired.
- They don't borrow tomorrow's energy to get a bit more done today.
- They work at a rate that allows them to maintain the highest quality standards for the duration of the project.



Principle 9: Continuous Attention to Technical Excellence and Good Design enhances Agility.

- High quality is the key to high speed.
 - The way to go fast is to keep the software as clean and robust as possible.
 - Thus, all agile team-members are committed to producing only the highest quality code they can.
 - They do not make messes and then tell themselves they'll clean it up when they have more time.
 - Do it right the **first** time!




Principle 10: Simplicity – the art of maximizing the amount of work not done – is essential.

- Agile teams take the simplest path that is consistent with their goals.
- They don't anticipate tomorrow's problems and try to defend against them today.
- Rather they do the simplest and highest quality work today, confident that it will be easy to change if and when tomorrow's problems arise.



Principle 11: The Best Architectures, Requirements, and Designs emerge from Self-Organizing Teams

- An agile team is a self organizing team.
 - Responsibilities are not handed to individual team members from the outside.
 - Responsibilities are communicated to the team as a whole, and the team determines the best way to fulfill them.
- Agile team members work together on all project aspects.
 - Each is allowed input into the whole.
 - No single team member is responsible for the architecture, or the requirements, or the tests, etc.
 - The team shares those responsibilities and each team member has influence over them.



Principle 12: At regular Intervals, the Team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

- **An agile team continually adjusts its organization, rules, conventions, relationships, etc.**
- **An agile team knows that its environment is continuously changing, and knows that they must change with that environment to remain agile.**



Agile Frameworks (Scrum/Kanban/XP)

- **Scrum**
- **Extreme Programming (XP)**
- **Kanban**
- **Capgemini Agile Framework (CAF) Hybrid of Scrum & XP**
- **Feature Driven Design (FDD)**
- **Dynamic System Development Method (DSDM)**
- **Crystal**



Incremental and Iterative Development

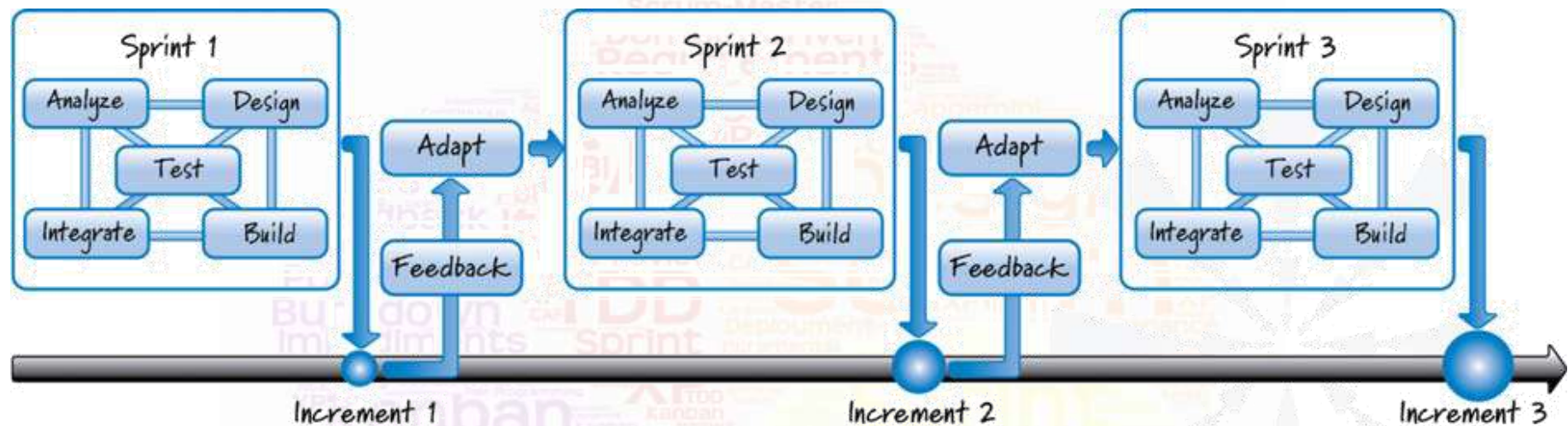
Cyclic software development process

Starts with initial planning and ends with deployment with cyclic interactions in between.

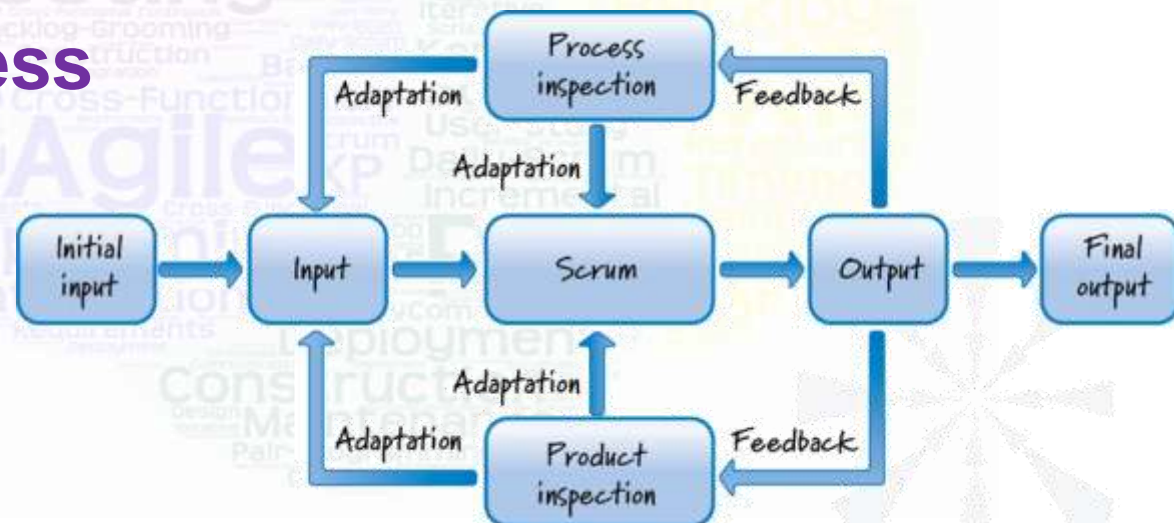
Essential parts of the Scrum, Rational Unified Process, Extreme Programming and various agile software development frameworks.

Follows a process similar to “plan-do-check-act” cycle of business process improvement

Incremental & Iterative Development



Emperical process







Heads Up Golf (End-line)

The objective of the game is to highlight the benefits of iterative and incremental development.

Instructions:

1. Listen to the instructions carefully when asked to move from point x to point y
2. Start only when the instructor says GO
3. Pause if the instructor says PAUSE
4. Stop when instructor says STOP
5. Start from any point on the dotted rectangle for the designated flag.
6. There will be a total of two rounds in this endline activity
7. When moving from point x to Point y, draw dotted line for the path like this - - -
8. Wandering into Water Hazard (WH) would mean a penalty 5 points (-5)
9. Touching the Bunker (B) will invite a penalty of 3 points (-3)
10. Reaching the target fetches 10 points.
11. There are no winners or losers. Only learners.



Kanban



Problems with current system

- Burnout
- Frequent bugs on Production
- Complaints about productivity
- Low throughput
- Leads to vague sprint planning
- Too much work stuffed into one sprint
- Unidentified bottlenecks





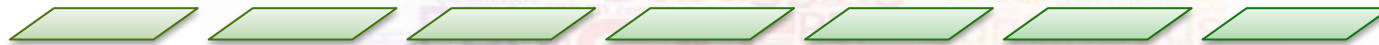
History of Kanban

- Developed by Taichi Ohno at Toyota in 1940's
- Designed after the shelf-stocking techniques used by supermarkets
- Demand controlled system where replenishment happens based on market conditions
- Based on a pull based system rather than a push based one
- Use of visual signals was essential to the system



Principles of Kanban

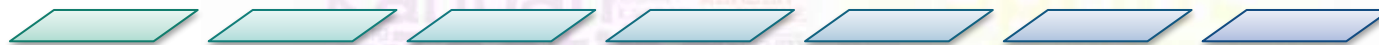
Visualization



Limiting work in progress



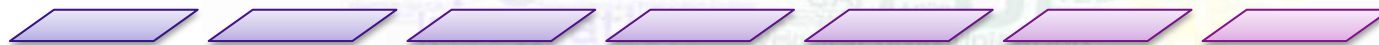
Flow management



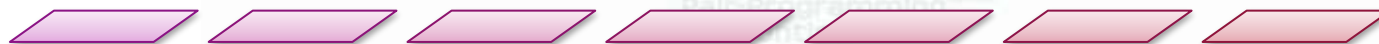
Making policies explicit



Using feedback loops



Collaborative or experimental evolution.





Implementing Kanban

Step 1: Visualize your work

- Break down the flow of work from the moment you start it to when it's finished into distinctive steps and draw a column for each
- Get some stickies, write down each task on a separate sticky note. Use different colors for different types of work.
- Put them on the whiteboard. Each task will move from left to right until it's done and leaves the workflow

Step 2: Limit work in progress

- Kanban is all about maintaining flow and eliminating waste.
- Start by putting limits on columns in which work is being performed.
- WIP Limits aim to enforce a high, smooth flow of work and eliminate different kinds of waste.



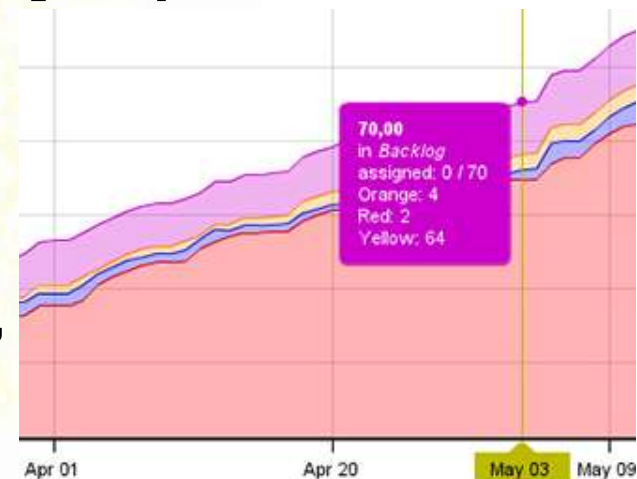
Implementing Kanban

Step 3: Don't push too hard. Pull!

- It's easy to get friction between different teams, especially when one is performing better and pushes more work than another one can actually handle. A solution to this is a pull system, where next team pulls work only when they are ready for it.
- You can implement pull system by adding a limited capacity buffer between teams.

Step 4: Use, monitor, adapt and improve

- The best tool to measure Kanban performance is Cumulative Flow Chart. Each day, for each column, mark how many tasks are in it or somewhere further down the workflow.
- This will produce a mountain-like looking chart, which gives insight into the process, shows past performance and allows to predict future results.





Lead Time and Cycle Time

Lead Time

The **Lead time** is the time from the moment when the request was made by a client and placed on a board to when all work on this item is completed and the request was delivered to the client. So it's the total time the client is waiting for an item to be delivered.

Cycle Time

The **Cycle time** is the amount of time, that the team spent actually working on this item (without the time that the task spent waiting on the board). Therefore, the Cycle time should start being measured, when the item task enters the "working" column, not earlier.



Kanban in Action



kanbanboardsimu
1-0-1-13041815214



Q n A

MAAKE
TERMA KASIH
GRAZIE
MERC
CHOKRANE
MATUR NUWUN
MATONDO
CHOKRANE
DANK JE
RAIBH MATH AGAT
SPASIBO
MAAKE
OBRIGADO
WELALIN
SPASIBO
ARIGATO
MOCHCHAKKERAM
OBRIGADO
KIITOS
DANKON
NIRRINGRAZZIAK
MOCHCHAKKERAM
JUSPAXAR
OBRIGADO
MATONDO
KIA ORA
MULTUMESC
CHOKRANE
SALAMAT
CAM ON BAN
MERC
OBRIGADO
MOCHCHAKKERAM
UA TSAUG RAU KOJ
KIITOS
OBRIGADO
DANK JE
MAMANA
NIRRINGRAZZIAK
MULTUMESC
VINAKA

THANK YOU

