CSC301

Survey of Additional Agile Processes: XP & Kanban

Software Processes

- Previously, we discussed the need for *software processes*
  - Conventions, rules and/or tools used for organizing a team
  - Explicitly define roles, events and artifacts
- We briefly described *Waterfall*
- We finished the discussion talking the Agile manifesto
  - A set of rules/values/guidelines that software teams should follow
  - The focus is on adapting processes to the reality of the business
  - Some of the highlights are: Transparency, incremental improvements, accepting changing requirements, collecting user feedback, delivering code as frequently as possible, etc.
- We introduced Scrum
  - Roles: Scrum Master, Product Owner, Development Team
  - Activities: Sprint Planning, Daily Scrums, Sprint Retrospective, Sprint Review

Team Project

- Deliverable 1 was due on Friday
- Now, you start building your product
- You will define your own (agile) process
    - Different from standard Agile processes used in the industry
        - Different situations (with different constraints) require different processes.
    - Define your roles and communication process clearly
        - How are you communicating? How frequently?
        - Who is responsible for what? What are expectations?
    - Be precise when you describe your Git/GitHub workflow
        - Do you use pull-requests to merge code? Who is responsible for merging?
        - How are you keeping track of issues? Using GitHub or something else?
        - How do you search, organize and/or prioritize issues?

Team Project

- **Everybody** is expected to **code**!
    - **No exceptions**
    - Writing Docs ≠ coding
      You are expected to write actual code
- You are expected to **contribute valuable work at least twice a week**
    - We want you to work continuously so you can make incremental improvements.
- Your TA's will evaluate your individual contribution based on the graphs of your team repo
    - If you see that your commits are not associated with your account (and, as a result, are missing from the graph), read GitHub docs explaining how to fix that (by changing the Git configuration on your local machine).
    - Email your TA if the graph does not show your work

Software Processes

- Today, we’ll describe two other Agile processes
  - XP & Kanban
  - Both have been very popular in the industry
  - Give you an example of the level of details people use to describe processes.
- Even within Agile processes, you will see a trend
  - Processes become less prescriptive
  - Processes become lighter

XP

XP - Extreme Programming

- An agile process
  - In *Java* terms one would think of *Agile* as an interface and *XP* as one of its implementations.
- A lot of hype in the late 90’s and early 2000’s.
- Prescriptive - Consists of many rules and practices.
- XP: A Gentle Introduction

XP - Extreme Programming

- Some highlights of XP:
  - Iterative incremental model
  - *Pair programming*
  - *Tests are written before the actual code*
  - Customer’s decisions drive the project
  - *Dev team works directly with a domain expert*
  - Accept changing requirements (even near the deadline)
  - Focus on delivering working software, instead of documentation
- Some are general Agile values, some are specific to XP

XP - Extreme Programming

- XP is a very detailed and fairly rigid
    - Open work space and daily stand-up meetings are prescribed
        - Doesn’t really work for remote teams
    - Pair programming is prescribed
        - Pair programming = Two people working on one machine
    - TDD is prescribed
        - Old concept, rediscovered by Kent Beck
        - The idea simple: Write a failing test, write the code to pass it, then repeat.
- In practice, a team may choose to adopt only a subset of XP’s rules.

Since we already mentioned TDD ...

Test Driven Development

- Some arguments for it:
    - Helps focus the development efforts (on the important features).
    - Thinking about how the code will be used, before writing it, leads to better design.
    - Tests serve as clear specifications.
    - Regression tests allow you to change your code and be confident that you did not break anything that was working before.
    - Increases confidence in functionality and minimizes technical debt increase

Test Driven Development

- Some arguments against it
  - Not always cost-effective - Testing infrastructure/scaffolding can be very expensive and/or complex to build
  - Too many unit tests, not enough system tests.
  - Sometimes it’s hard to tell whether the side effects of your code are correct.
    - Your code depends on 3rd party API’s.
    - Your code is dependent on a specific condition (e.g., time/geography)
  - Strict test-first approach is not for everybody (and not for every project).

Pair Programming

- As mentioned, pair programming is also a part of “the XP prescription”
  - Two people working on one machine
  - One drives (i.e., with their hands on the keyboard) and one navigates/observes
  - Usually, swap roles every now and then
- Can you think of arguments for pair programming?
- Can you think of arguments against it?

XP, Summary

- Started in the late 90’s
- Probably the first popular Agile process
- In today’s standards, XP is considered fairly “heavy”
  - Prescriptive process with many rules
  - Many assumptions that don’t apply to many teams (e.g., having an open workspace, a customer that is always available, a team that can meet daily for a stand-up meeting, etc.)

Kanban

Kanban

- Kanban is another agile process that is gaining popularity
    - Designed to improve (and maintain a high) level of production
    - Lighter and less-prescriptive than Scrum
    - Gaining popularity amongst software teams

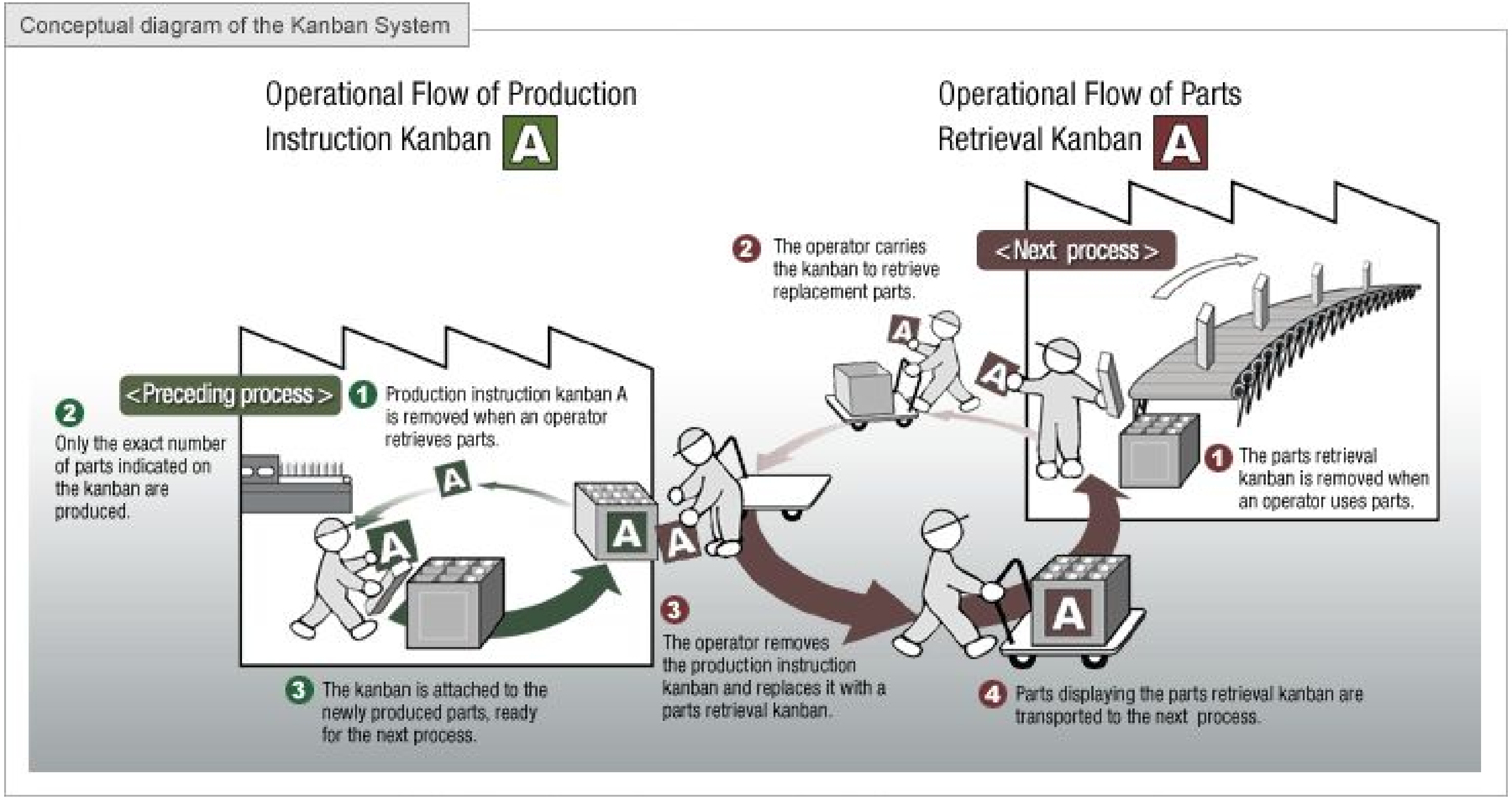
Origins

- Kanban originates from Toyota’s efforts to perform Just In Time manufacturing (also called Toyota Production System, TPS)
- Inventory is stored in bins
    - *Kanban card*, with product details, sitting at bottom of each bin
- “Pull” more inventory when running low
    - When bin on floor runs out, fetch bin from storage
    - When storage depleted, fetch bin from supplier
    - When supplier inventory runs out, make more

Just in Time manufacturing

- Work is done in a pipeline
- Each stage of pipeline needs inventory
- As inventory runs out, signal preceding stages to get more
- Consider lead time

Toyota illustration of kanban


Conceptual diagram of the Kanban System
Operational Flow of Production Instruction Kanban A
Operational Flow of Parts Retrieval Kanban A
<Preceding process>
2 Only the exact number of parts indicated on the kanban are produced.
1 Production instruction kanban A is removed when an operator retrieves parts.
3 The kanban is attached to the newly produced parts, ready for the next process.
2 The operator carries the kanban to retrieve replacement parts.
<Next process>
1 The parts retrieval kanban is removed when an operator uses parts.
3 The operator removes the production instruction kanban and replaces it with a parts retrieval kanban.
4 Parts displaying the parts retrieval kanban are transported to the next process.
A

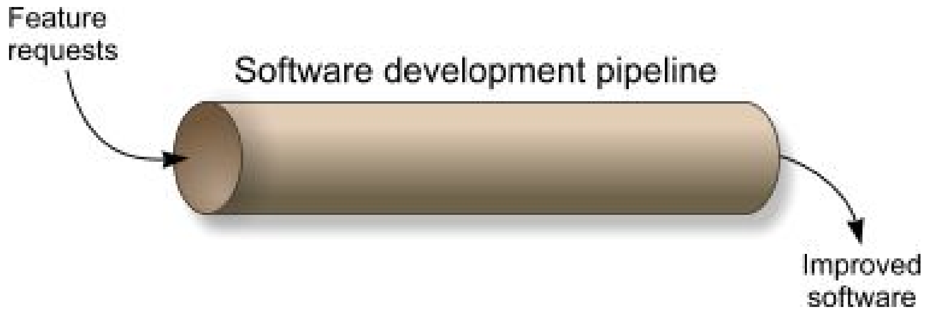
Kanban for software

- No prescribed roles
- No prescribed meetings
- One artifact (the Kanban board) and one simple concept ...

Kanban

Think of the development process as a pipeline

- Feature requests come in
- Improved software comes out



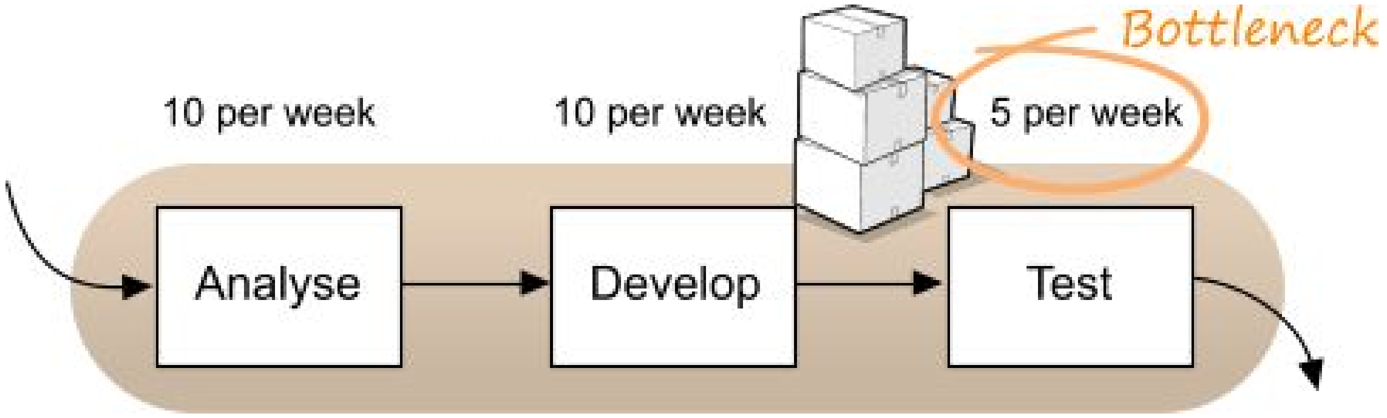
http://kanbanblog.com/explained/

Goal: Maximize throughput

Kanban

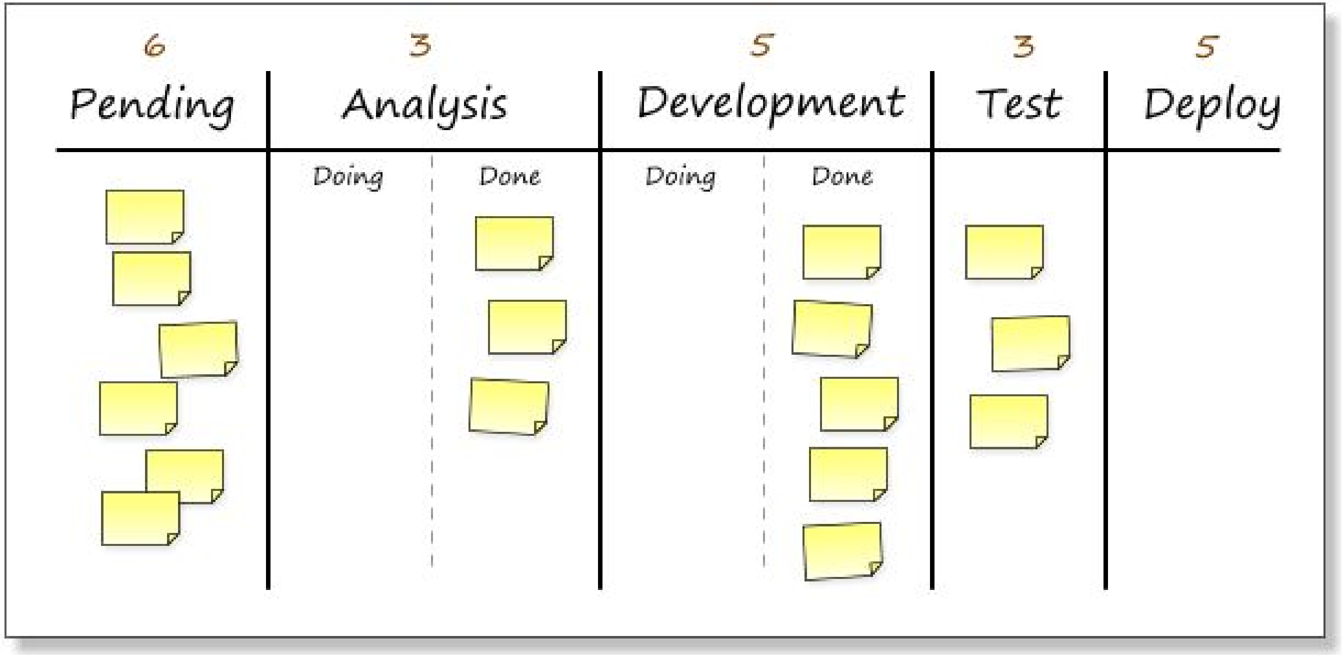
Items go through the pipeline in steps

- The number of steps and their names, can differ between teams and/or projects
- Throughput of the pipe is limited by the bottleneck



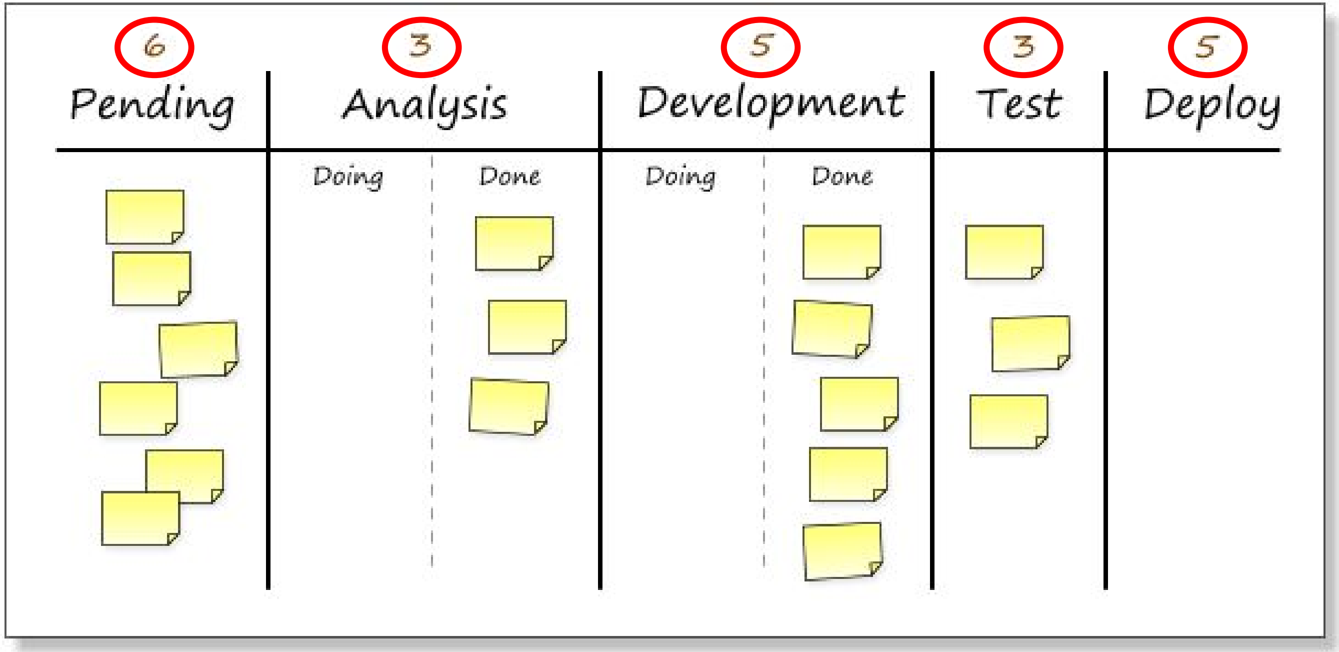
The Kanban Board

- Columns represent pipeline steps
- Sticky notes represent items



The Kanban Board

- Items flow from left to right
- Limit the number of items in each column



The Kanban Board

- Limit amount of work-in-progress (WIP) that may accumulate at each step.
- When items pile up in a specific step
  - There exists a bottleneck downstream
  - Get other team members to help
- We measure *lead time* - The time it takes for an item to make it through the board

The Kanban Board

- Use whiteboard + sticky notes
- Or one of the many available software tools
- Or even something as simple as GitHub issues
  - Labels can indicate columns
  - Search to see which issues are in a column
    - At a given point in time,
    - Or within a time range
  - Developers should respect the WIP limit(s)

Kanban vs. Scrum

- Similarities
  - Self-organizing team
  - Break work into tasks
  - Developers pull tasks
  - Transparent processes
  - Frequent delivery
- Both are agile processes

Kanban vs. Scrum

- Kanban doesn’t have prescribed roles
  - What about product owner’s responsibilities?
  - Who facilitates the process?
- Up to the team to divide responsibilities
  - E.g.: Prioritizing tasks and decide which items go on the Kanban board first

Kanban vs. Scrum

- Scrum uses fixed-length sprints
  - Start with planning meeting (Scrum board is reset)
  - Continue with daily meetings
  - End with review & retrospective meeting
- Kanban is an ongoing process
  - Board is never reset (you may do “releases” to clean up the board)
  - No meetings prescribed
  - The team decides on the frequency, duration and nature of its meetings

Kanban vs. Scrum

- Estimating task size
  - Prescribed in Scrum, optional in Kanban
- Charts
  - Burndown in Scrum, no prescribed chart in Kanban
- Metric
  - Velocity in Scrum, lead-time in Kanban
- Limit Work-In-Progress
  - Implicit in Scrum, explicit in Kanban

Kanban vs. Scrum

- Kanban is lighter & less prescriptive
  - Less prescribed meetings and/or artifacts
  - Less facilitation required
- Kanban fits well with continuous deployment
  - Release when an item makes it to the last column
  - No need to wait until the end of the sprint
- Kanban gives more freedom (and leaves more decisions) to the team

Software Processes, Summary

- Historic trend
  - From Waterfall to XP
  - From XP to Scrum
  - From Scrum to Kanban
- Process are
  - Becoming lighter
  - Less prescriptive
  - Supporting more frequent delivery
- For your team project, you should define a process that works for you
  - Articulate decisions explicitly
  - Reflect on them when preparing the deliverable(s)