Survey Common Agile Processes: XP, Scrum & 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*
  ○ Older process, was the standard for many teams (for many years)
  ○ Divide a full project into a sequence of 7 stages
  ○ More suitable for construction & manufacturing than for (most) software organizations

● 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.
Team Project

● Deliverable 1 is due this Friday.
● In other words, after Friday, you start building your product
● You will define your own (agile) process
  ○ Different than standard Agile processes used in the industry
    ■ Different situations (with different constraints) require different processes.
  ○ Be precise when you describe your Git/GitHub workflow
    ■ Do you use pull-requests to merge code? Who is responsible for merging?
    ■ Are you using forks, remote branches or a combination of the two?
    ■ How are you keeping track of issues? Using GitHub or something else?
    ■ How do you search, organize and/or prioritize issues?
    ■ Who is responsible for assigning issues?
    ■ etc.
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](https://docs.github.com/en/authentication/changing-your-github-login#changing-the-github-username-on-your-local-machine) (by changing the Git configuration on your local machine).
Software Processes

● Today, we’ll describe three Agile processes
  ○ XP, Scrum & Kanban
  ○ All are/were 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 - 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.
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. Ex: Your code depends on 3rd party API’s.

○ 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 the one machine
  - One drives (i.e. with their hands on the keyboard) and one navigates
  - 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.)
Scrum
Today’s Tour - Scrum

- **Scrum**
  - An agile process
  - Well ... More framework than process
  - Extremely popular in the last 5 - 10 years
  - Fairly prescriptive
  - Empirical - Make predictions based on past experience
    "Only what has happened may be used for forward-looking decision-making"
Let’s meet the players and some of the basic rules of the game ...
Scrum - Roles

- **Product Owner**
  - Focused on value to stakeholders
  - Ultimately, responsible for the success of the product

- **Scrum Master**
  - Facilitator, ensures that everything is running smoothly

- **Development Team**
  - Get stuff done!
  - Engineers, designers, creatives, QA, etc.
  - Traditionally, 3 - 9 people
Scrum - Events

- **Sprint**
  - Scrum terminology for iteration
  - Fixed-length. Usually, 1 - 4 weeks.

- **Each sprint has the following events:**
  - Planning meeting, at the beginning of the sprint.
  - Daily (standup) meetings, during the sprint.
  - Review and retrospective meeting, at the end of the sprint.
Scrum - Artifacts

- **Product Backlog**
  - Ordered list of high-level requirements (e.g. user stories)
  - For each requirement,
    - Business value is assessed by product owner
    - Cost is assessed by the development team

- **Sprint Backlog**
  - Backlog items to be completed during a sprint
  - Broken down to more granular, concrete tasks

- **Product Increment**
  - Completed product backlog items
  - Commonly visualized by burn-down charts
    - In different resolutions, sprint or release
Scrum Artifacts

On the right, you can see an example of how a team might organize their artifacts on a Scrum board.

Notice that details may change from one team/project to the other.

- Physical vs. software.
- Categorizing the sprint backlog (e.g. To do, In progress, testing).
- Keeping the product and sprint backlogs on the same board.
- etc.
Let’s see how the game is played ...
Scrum - Sprint Planning

● Decide what to do
  ○ Select items from the product backlog
  ○ All players participate

● Decide how to do it
  ○ Prepare the sprint backlog
    ■ Break product backlog items into concrete tasks
    ■ Plan how to complete the tasks
      Ex: Design diagrams, task ordering, etc.
  ○ Only the development team participates
Scrum - Sprint Planning

● How to decide what to do?
  ○ Product owner sets goals & priorities
    ■ Understands the stakeholders and business value
  ○ Development team decides what can get done
    ■ Understands the actual required work
    ■ Has a good sense of the team’s ability
    ■ Knows about any special circumstances
      Ex: Someone is sick or away for a conference
  ○ Work together towards the same goal - Deliver value!
Scrum - Sprint Planning

● What exactly is the sprint backlog?
  ○ Product backlog items + a plan how to complete them
  ○ Product backlog items become concrete tasks / user-stories
    ■ Ideally short (< day)
    ■ Clearly define the notion of “done”
    ■ Estimated size (in hours or points)
    ■ More immediate task ⇒ More details
Scrum - During The Sprint

● Development team members pull tasks

● Daily Scrum meeting
  ○ Regular time & place
  ○ Very short (e.g. 15 min), usually standing up
  ○ Development team members only
    ■ Scrum master facilitates
  ○ Synchronize, inspect progress, bring up issues, etc.
  ○ Everybody is put on the spot
Scrum - End Of The Sprint

- Review meeting
  - All players + stakeholders
  - Demo work that was done
  - Mention work that wasn’t done
  - Provoke a discussion
    - What went well, what didn’t
    - Updates to the product backlog
    - Ideas for next sprint
  - Recommended dosage: 1 hour per week of work
Scrum - End Of The Sprint

- Retrospective
  - Usually, immediately after the review meeting
  - All players participate, but no stakeholders
  - The team suggests improvements to its process
  - Based on the review meeting
  - Focused on Process Quality Improvement
    - Productivity
    - Accuracy of estimations
    - etc.
Some observations ...
Scrum, Observations

● **Division of responsibility**
  ○ Product owner focused on business value
  ○ Development team on building
  ○ Scrum master on facilitating

● **Collaboration and reality checks**
  ○ Product owner ensures focus on the important tasks
  ○ Dev team ensures realistic expectations
Scrum, Observations

- Accurate estimations are important
  - Product owner’s credibility
    - Deliver on promises to stakeholders
  - Dev team’s credibility
    - By committing to manageable amount of work
  - Everything is easier when your team is predictable
    - Budget, hiring, etc.
Scrum, Observations

● Further into the future ⇒ Use less details
  ○ Avoid “science fiction” by being less precise
    ■ Tomorrow’s plan is more precise than next week’s, which is more precise than next month’s
  ○ Don’t plan too far into the future:
    ■ Requirements may change
    ■ Your understanding of the problem may change
    ■ Harder to predict
One more important artifact ...
Burndown Chart
Sprint Burndown Chart

● The basics:
  ○ X-axis: Timeline, granularity is days
  ○ Y-axis: Remaining work, in hours/point
  ○ On day zero, all work is remaining
  ○ On the last day, all work should be done

● The information:
  ○ Team’s productivity and/or
  ○ Ability to accurately estimate effort

● Let’s see a few examples ...
What Does It Say About Your Team?

- Took a little while before tasks started to get done
- Could be many reasons:
  - Spent the first half of the sprint learning new technologies.
  - Didn’t have granular enough tasks, and progress was only noticed towards the end of the sprint
  - Our team only works well under the pressure of a deadline
What Does It Say About Your Team?

- You finished the work halfway through the sprint.
- Could be many reasons:
  - The team had a superstar sprint
  - The team’s initial estimation was off
  - You intended to finish early. Maybe there’s a holiday vacation, so you wrapped up the sprint early
  - Unexpected extra help became available
What’s The Point?

- Data Visualization reveals information
- Improved traceability
  - Keep historical data of the team’s performance
    - Velocity is a common metric for a given team
  - Easier to notice trends
  - Easier to evaluate decisions
    - Ex: Did switching programming languages make our team more productive?
Scrum, Summary

- Scrum gives our team some structure
  - Roles, events and artifacts
  - In order to benefit, everybody has to play by the rules
- Team decides on “implementation details”
  - Length of sprints
  - Scale used for evaluating difficulty/cost, and how to assign points to issues.
  - How to do the actual work (e.g. some teams use pair-programming with Scrum)
  - etc.
- Aims to promote
  - High productivity
  - Transparency & traceability
  - Professional responsibility
  - Collaborative culture
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
- 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
Toyota illustration of kanban
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

Goal: Maximize throughput

http://kanbanblog.com/explained/
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
  - Ex: 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
  - 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
End of the tour
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)