CSE 115A - Introduction to Software Engineering

Elijah Hantman

Note:

Today is the day the projects are supposed to start, ideally we would have a sprint plan and TA meetings/stand up times ready to go, but it should be fine as long as we get it done within the week.

This should be the first sprint, but since we ran a little late, we should assume less time.

We also need to make presentations.

Scrum Best Practices

Review

Roles

- Product Owner, communication between customer and team, generally responsible for prioritizing user stories and deciding what exactly each release will look like.
- Scrum Master, organize and fascilitate Scrum practices among the team. They also serve as an advocate for the team to management.
- The Team, the actual experts and people who are responsible for creating the final product and managing what exactly happens in each sprint.

Definitions of Done

- Agile emphasis
  - deliver actual functionality
  - avoid waste
- Strict definition of progress
  - user stories
  - tasks
- Strict definition of completion
  - user stories
  - sprint tasks

Each team creates their own definitions which **must** be applied consistently

Example

- Code checked into repo
- Code reviewed for style and standards
- Code reviewed by team member
- External/public API documentation
- Unit tests
- Non-functional tests (perf, memory, usability, etc.)
- Regression tests
- static code analysis
- test coverage

Acceptance Criteria

- Is the thing what was requested?
- Criteria
  - objective criteria for user story completion
  - Basis for functional testing
  - Basis for testing before release
  - Basis for BDD/ATDD

        * BDD: Behavior Driven Development
        * ATDD: Acceptance Test Driven Development
- User stories in Product Backlog
    - User story start of conversation
    - Product backlog contains all possible things anyone thought to ask for
- User stories in Sprint Backlog
    - Commitment to actually create
    - should be known what is required to complete and fulfill the user story.

Team Working Agreements

- Logistics
    - Dev environments
    - Coding Style/standards
- Work Patterns/Process
    - Definitions of done
    - collaboration
- Product Design Patterns
    - UX/UI look and feel
    - architecture
    - Error handling.

Avoiding Waste

- Unresolved problems grow
    - defect build up
    - Technical Debt: steep interest rate

    > As a side note: This is where a preliminary design would be helpful. It can allow for us to discover the 'correct' way to do something without necessarily taking as much time as just trying to build the right thing from the start.

- Minimize time between defect insertion and correction
    - Goal is to make choices which allow for things to be tested and finalized as soon as possible so that fixes are small and contained to single tasks or only a couple sprints.

Increment Strategies

- Conventional Approach
    - Layer by layer
    - Good when there are hard dependencies, if one layer depends on the layer below existing in its entirety.
      Also usable when the system itself cannot be easily broken apart into functional units, like some hardware problems.
    - Example
        * Start with Frontend
        * Then Server
        * Then Backend

- Alternative Approach
  - Slice by Slice
  - Example
    - Start with Feature 1
    - Then Feature 2
    - Then Feature 3
  - Each slice includes the full tech stack required to make it work. For a web app the Front end, and Backend would be developed at the same time.
  - Benefits to slice by slice is that it creates a testable feature at each step, and it mirrors the way the customer and product owner interact with the software
- Considerations

  Try and minimize the number of stories which are in progress. Only fully completed stories are actually useful to the Product Owners and Customers.

Splitting User Stories

- User stories need to fit into a sprint
- Guideline
  - Size of user story should be ¡50% of a sprint

How can we break apart large stories?

- Split by workflow

  Example: create an account in Banking App
  - Checking
  - Savings
  - Line of Credit

  Ideally split and then prioritize based on usefulness and the frequency of usage.
- Split by Lifecycle

  Example: create an account in Banking App
  - Create Account
  - Make Deposit
  - Withdraw Money

  General Patter: CRUD
  - Create
  - Read
  - Update
  - Delete
- Split by Convenience Level

  Example: create an account in Banking App
  - Enter data without autofill
  - Type in Data vs select from menu
  - Basic vs cool UI
  - Manual vs Automatic

  Start with basic tools and add the convenience and polish on top.
- Split by User Typicality

Example: create an account in Banking App

- Novice, occasional, expert
- single account owner vs multiple

Add context to the user story to make it more specific.

What **Not** to do

- Do not split by architecture
  Not helpful to the user, leaves features which are not complete for demoing and testing.
- Do not split by function or critical qualities
  - cannot split off security
  - Data protection.

  Leaves features which are broken and or actively harmful to users. Can also make adding the critical feature later more costly and complicated.

Misc Tips

- Defined Room
- Stand up same time and place every time
- Enforce working agreements
- Post Three Questions, should not need to be reiterated every time

1. What did you do
2. What are you going to do
3. What is stopping you

- Use a physical Scrum board in addition to whatever other tools.

Not entirely sure why for this one? I mean a scrum visualizer isn't all that bad, and you shouldn't be updating the scrum board in the meeting anyways.
Maybe put the scrum board in a public place.

- Use Synchonous Communication

Not for everything, but it forces an immediacy to the communication which enhances the speed of communication and increases accountability.

Agile Myths

- Understanding Problems
  - What can go wrong
  - How to identify and remove obstacles
- Agile means no plan

Planning is still required, and happens on multiple levels
The main characteristic is that the plan is flexible and constantly is updated with the new knowledge

- Agile means any process is fine

Agile still requires well defined processes and discipline

- Agile means rushing products out of the door

  > While agile emphasizes functional intermediate builds, it does not require poor engineering or doing the fastest least effort thing always.
  >
  > Agile relies on good feedback which means robust and fast processes for communication and revision.

Scrum Smells

- Zero or more than 1 product owner.

  > Product owner has the vison and controls the direction. Zero or multiple either lack or have conflicting visions and direction.

- The Scrum Task Master

  > A scrum master who micro manages and directly commands team members

- Commitment Phobia

  > Avoidance on committing to compeleting tasks or to actually making decisions

- Self unmanaged teams

  > Teams that aren't managed externally and have little internal control.

- Burn up/down charts which stagnate

  > Stagnation shows that something else may be wrong. Either everyone is blocked, or people aren't updating the chart, or even that something is wrong with the tasks.
  >
  > Sign to begin investigation into what is happening exactly on the team.

- Urgent things crowd out important things
  - Important vs Unimportant

    > Whether a task contributes to the goals of a system

  - Urgen vs Not Urgent

    > Requires immediate attention

  It is a problem if the unimportant urgent things cause the important non urgent things to not be done. A good Scrum process ensures that all important things are done regardless of urgency.

  It is also a benefit if things can be taken care of or forseen before they become urgent.

Scrum-but

- Scrum without key features
  - no stand ups
  - stand ups don't respect time
  - No permanent Scrum master or product owner
  - Sprint length varies
  - Fix bugs in stabalization sprint
  - No sprint review
  - No sprint retrospective

Ask about alternatives, replacement systems.

Technical Practicies

Borrowed from XP (eXtreme Programming)

- Done Criteria
- Peer review / Pair Programming (lmao)
- Clean Code (yuck)
- TFD/TDD (mixed)
- Continuous Integration
- Version Control
- Test Coverage criteria
- Static Analysis Tools