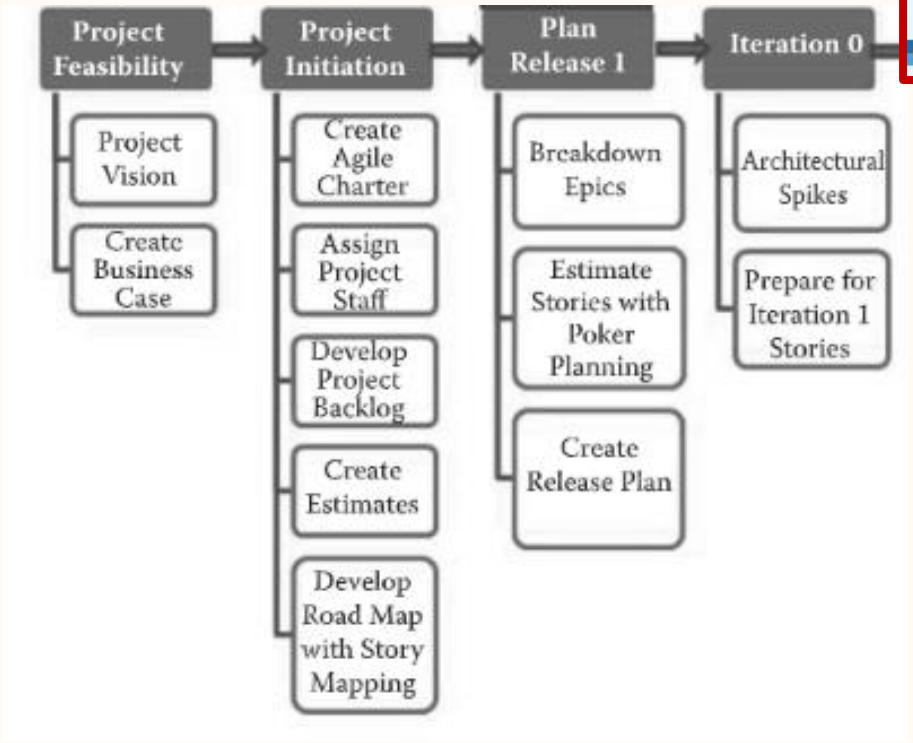
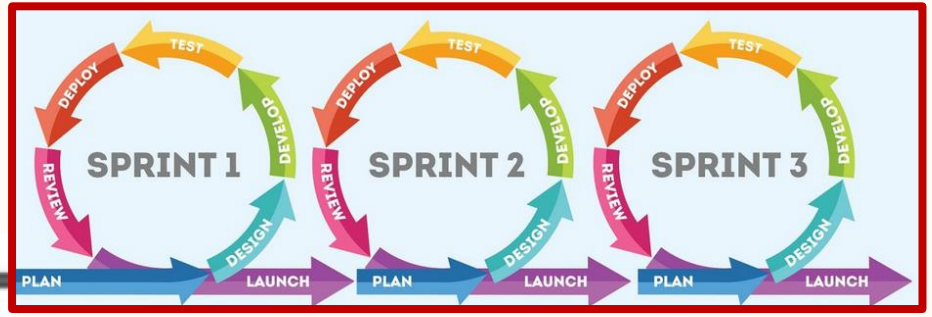
Development

Coding, testing, & deploying

Context





- Planned project at high level
- Initial designs: stories, architecture, interactions
- Considered solution stacks

- Today's focus: What does development entail?

Outline

- Backlog grooming & Sprint planning
- QA/QC
- Testing & test-driven development
- Coding: environments and practices
- Scrum meetings, deployment

Iteration planning (reprise)

- Recall: Product owner (on customer behalf) is responsible for **backlog grooming**
  - Includes prioritizing the product backlog
- At start of each sprint
  - Agile team selects set of high-priority stories they feel confident they can complete during sprint
  - These go into the **sprint backlog** where they can be further broken down into tasks

Prioritization in product backlog

- How high is the risk?
- How much value for customer, business?

story	**priority / order**	**risk**
adding and viewing records*	high / first	low
updating and deleting records	high / later	low
(etc... you will choose your own features)	X	Y

** "As a media consumer, I want to add basic records (e.g. text) about media I consume, so that..."*

Steps to writing expanded stories

- Create user personas
  - Identify who these stories are for
- List ordered steps
  - Consider what steps user and system take in story
- Outline tasks and subtasks
  - Plan specific development steps, who does what
- Define “done”
  - Define criteria for success in user story, process

Example

- Persona(s): David, media consumer
- User story: *As a media consumer, I want to be able to add basic information about the media I consume (e.g. title, media type, author, etc.), so that I can keep a record of what I've consumed*
- Steps / interactions
  - David starts application
    - show home screen
  - David clicks "add record"
    - show form to add information
  - David types in information and clicks "submit"
    - propagate data to data service / DB
    - retrieve all records
    - redirect to records listing

Note: if you're starting from scratch, this may be an epic!

- Tasks
  - UX designs / user testing
  - backend / data management
  - frontend / UI development
  - feature / integration testing
- Definition of done
  - a user can add records to the database and view a list of all records.
  - all tasks have been completed (developed, tested, reviewed, and validated)

Decomposing high priority stories

- Given a story like this…
  - As a media consumer, I want to be able to add basic information about the media I consume (e.g. title, media type, author, etc.), so that I can keep a record of what I've consumed
- How might I break up to prioritize further?
  - “As a <user> I want to **manage records** so I can persist data for media consumers”
  - “As a <user> I want to **access** the app through a web interface so I can use it from any device”

Backend stories

- User of API or other backend services could be a developer (via other software)
  - They are the consumer of the API
- Backend stories might look like this:
  - "***As a <user> I want to <CRUD database records> so that I can keep persistent data for my app***"
    - "As a <user> I want to create records to persist data"
    - "As a <user> I want to read records to use data"
    - "As a <user> I want to update records to persist data"…

Outlining tasks

- Story: "…I want to **create|read** records so…"
- Associated developer ***tasks*** might include:
  - Design REST API for basic read/write operations
  - Develop operations in PyMango
  - Develop data manager service using Azure functions
  - Deploy data manager service

Definitions of “done” for backend

- Might look something like this for larger task:
- Design and develop REST API / data service using Azure functions
  - Endpoints for CRUD operations are defined
  - Service is developed and tested with API tests
  - API is Documented
  - Service is Deployed

Front-end tasks

- Note that back-end can be developed without front-end involvement, vice-versa
- Some front-end tasks related to back-end
  - Like, developing views/interactions for CRUD
- Definition of done here involve the user
  - User can add records via a web interface
  - User can view all records via a web interface
  - User testing is complete

Defining “done” for main story

- Story: “…I want to **create|read** records so…”
- Definition of “done” for story might include:
  - User can add a new record to the database
  - User can retrieve all records from the database

Front end

  - Data manager service is well tested (by someone in addition to developer)
  - Data manager service is well documented
  - Data manager service is deployed, reachable

Back end

Outline

- Backlog grooming & Sprint planning
- QA/QC
- Testing & test-driven development
- Coding: environments and practices
- Scrum meetings, deployment

QA and QC

- Primary focus of each
  - **Quality Assurance** (QA)
    - Processes/procedures that improve quality
    - Includes training, documentation, monitoring, audits
  - **Quality Control** (QC)
    - Product meets stakeholder expectations
    - Looking for defects that remain after development
- Often conflated, used together: **QA/QC**

Agile and QA/QC

Agile/Scrum designed to help with QA/QC

- Visibility, accountability, focus: working software
  - Use of tools like GitLab for **planning & tracking work**
  - Scrum **standup meetings**
- Regular interaction with stakeholders
  - Client is closely involved, **validates work of team**
- Continuous improvement
  - Sprint reviews focus on the **product**
  - Retrospectives focus on the **processes**

QC: verification and validation

- Did you build the system right?
  - **Verification**:

    process of determining if software is designed and developed ***according to the requirements*** or not
- Did you build the right system?
  - **Validation**:

    Process of checking if the software ***meets the needs and expectations of the client*** or not

Outline

- Backlog grooming & Sprint planning
- QA/QC
- Testing & test-driven development
- Coding: environments and practices
- Scrum meetings, deployment

Some types of testing

- Functional testing
  - Testing overall **functionality** (finding bugs)
- User testing
  - Test w/ end-users for **usability**
- Performance & load testing
  - Test response-, processing-time; **scalability**
- Security testing
  - Test **integrity**, protecting user data, etc.

Acceptance tests

- Acceptance test driven development (ATDD)
  - Create tests before any implementation occurs
  - Tests features, behaviors observable by user
  - Focus on business requirements, not code
- Goal: ensure product meets customer needs
  - Acceptance tests closely related to user stories
  - Customer or their representative performs tests

Acceptance testing

Perspectives are key

- Team members with different perspectives collaborate to write acceptance tests
- Three perspectives:
  - Customer (What problem are we trying to solve?)
  - Development (How might we solve this problem?)
  - Testing (What about…?)

Acceptance testing process

- Development team and customer discuss requirements
- Test cases entered into testing tool
- Development team develops code
- Dev team and/or QA team execute tests
- Dev team demo’s software (using automated acceptance tests where applicable)

Test-driven development (TDD)

- Usually lower-level form of functional testing
  - Unit testing—tests individual methods or units of functionality in isolation from rest of software
  - Integration (or feature) testing—tests units when combined with each other, as a functional group
  - System (or release) testing
- Develop tests first
  - Automated test is a dynamic specification—“a specification in executable form”

Automated tests

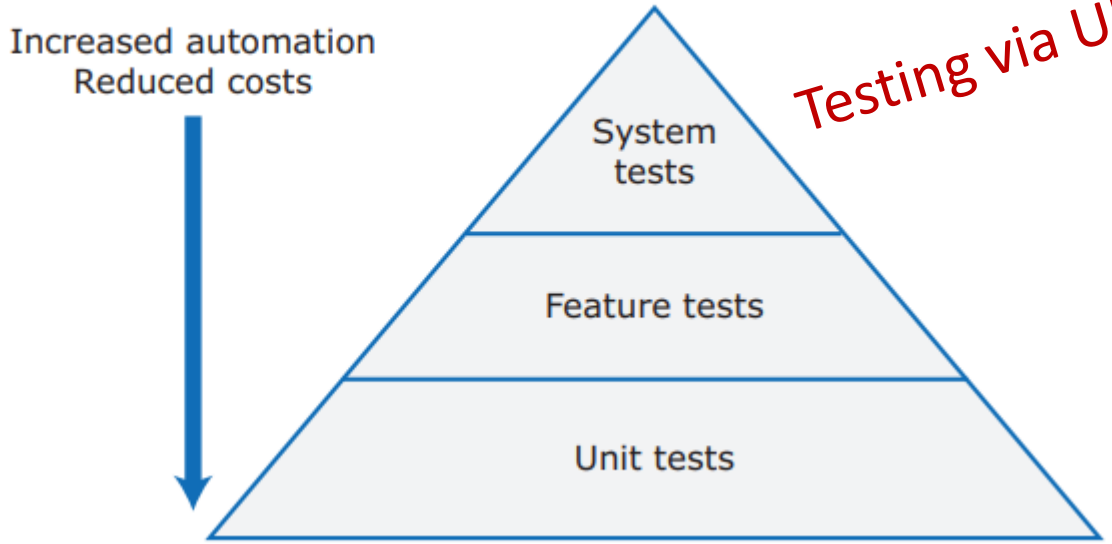
- Based on idea: Tests should be executable
- Each test consists of
  - Input data to unit being tested
  - Check that unit has expected behavior
    - Could test a resulting value
    - Could test for expected state of member variables
    - Could test that exception is thrown
    - Could verify order of calls to external code is correct
    - …

Regression testing

- Process of re-running previously existing tests when change is made to system
  - Checks for unexpected side-effects
    - Code change may have broken existing code
    - May reveal bugs undetected in prior testing runs
  - When automated, these take very little time
- A “regression” is a step backwards…
  - Bug in part of system that was working before

Test pyramid

Figure 9.5 The test pyramid



Testing via UI at this level

~10%

~20%

~70%

Outline

- Backlog grooming & Sprint planning
- QA/QC
- Testing & test-driven development
- Coding: environments and practices
- Scrum meetings, deployment

Coding environments (*sandboxes*)

- Best practice: provide dev team **sandboxes** in which to develop/test code
  - Dev. environment with "well-defined scope"
- Types of environments
  - **Dev** (development)—only for dev team
  - **Integration**—for multiple teams to integrate work
  - **Demo**—for giving demos to client
  - **Pre-production/test**—staging
  - **Production**—"live" environment

Coding environments

- Canty assumes
  - Dev team collaborates closely on shared codebase
  - Each team will integrate work with other teams
- In this class
  - Members work independently on sprints 2 & 3
  - Each team is isolated entity
  - New sandbox: **Individual dev** sandbox

Project/directory sturcuture

- <home dir on local machine>
  - CS 518
    - **Group project** ← Root of your Git repo
      - Documentation
        - Charter
        - Stories
        - …
      - Data Service ← Push code here end of sprint 2
      - Web app ← Push code here end of sprint 3
    - **Individual work** ← **Not** in your Git repo
      - Data Service ← Individual sandbox for sprint 2
      - Web app ← Individual sandbox for sprint 3

Collaborative coding practices

- **Pair programming**
  - 2 developers on same code: driver & navigator
- **Side-by-side programming**
  - Multiple developers in close proximity
    - Physical proximity ideal
    - Synchronous-remote ok (such as on Discord)
  - Working on different parts to be integrated
- **Code reviews**
  - Share code to inform, get feedback, etc.

Outline

- Backlog grooming & Sprint planning
- QA/QC
- Testing & test-driven development
- Coding: environments and practices
- Scrum meetings, deployment

Weekly ~~Daily~~ Scrum

- As described in the Scrum Guide, the purpose of the Daily Scrum includes
  - Inspect progress toward the Sprint Goal
  - Adapt the Sprint Backlog as necessary (adjust the upcoming planned work)
- You should have ***weekly*** Scrum meetings on Monday during lab time
  - You're not working on project enough for daily

Daily scrum

- Scrum team usually discusses these things:
  - Progress on goals for individuals, groups since last meeting
  - Impediments to progress (“blockers”)
  - Goals for individuals, groups until next meeting
  - Confidence in meeting goals for the sprint
  - Adjustments
- Meeting intended to be short: <15 min.

Weekly scrum for this class

- For next 2 sprints, all team members working on same development tasks
  - In industry, you'd "divide and conquer"
- What you should discuss:
  - How is everyone doing with current lab/assign.?
  - Is anyone stuck? (Discuss blockers)
  - Discuss confidence in meeting this goal:
    - Goal: Everyone completes the lab on time
  - Adjustments—need to ask faculty→deadlines?

Meeting minutes

- Summary notes of meeting
- You should record minutes for all meetings
  - Put in live document (Google Docs, Office 365, …)
  - Add newest notes to the ***top***
- Topics
  - Meeting date, attendees
  - Topics of discussion (summarize key points)
  - Action items
    - Include Who? What? and By when?

Iteration R

- Purpose: deploy product to production env.
  - Typically done in parallel to normal sprint
    - So dev team can continue work toward *next* release
  - This class: deployment incorporated into sprints
- Iteration R activities:
  - Project documentation
  - Formal testing
  - Deployment
  - Celebration!

Sprint review

- Inspect outcome of Sprint
- Determine future adaptations
- Scrum Team presents to key stakeholders
  - Results of work (demo)
  - Progress toward product goal is discussed
- Stakeholders accept/reject work
  - often the Product Owner makes this decision

Sprint Retrospective

- Scrum Team inspects how the last Sprint went with regards to
  - Individuals
  - Interactions
  - Processes & tools
  - Definitions of “done”
- Plan ways to increase quality & effectiveness