# Software Engineering

CS305, Autumn 2020

Week 15

# Class Progress… (last week)

## Software Quality

- What is quality? General and software-specific definition.
- Metric for judging quality (COQ)
- Why improve quality?
- Approaches and Implementation guidelines for continuous improvement of quality: TQM, ISO, and CMM

## Project Management

- Steps/activities in project management
  - Effort estimation and techniques – FP, COCOMO

# Class This Week..

- Agile Methodologies
- Revision

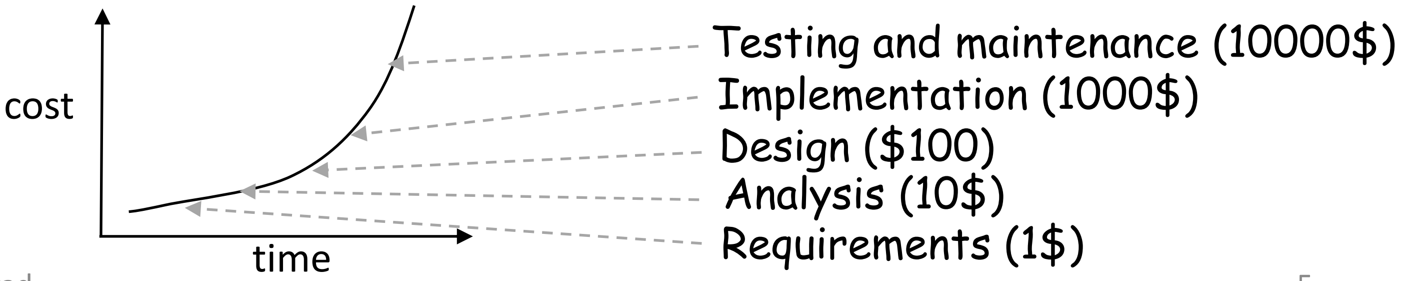
# Agile Development Methodology

- Another type of software development methodology heavily based on testing.
- Also called Test Driven Development (TDD)
    - Recall PA1 that briefly introduced you to TDD:
        - Developed test specs based on SRS.
        - Implemented test specs (test cases and test suites) – Functional Testing (Black-Box testing)
- A group of software developers published the manifesto for Agile Software Development in 2001.
    - They had met to discuss lightweight software development processes

# Why Lightweight Software Development?

- Recall waterfall model:
  - A phase in the process started only after the previous phase ended. Phases: `Requirements -> Design -> Implementation -> Testing -> Maintenance`
  - Very old (70s, some concepts date back to 50s), Not flexible w.r.t changing requirements and design
  - Good at catching errors early, which is important considering Boehm's observation of the cost of change:

*Cost grows exponentially with time*



Testing and maintenance (10000$)

Implementation (1000$)

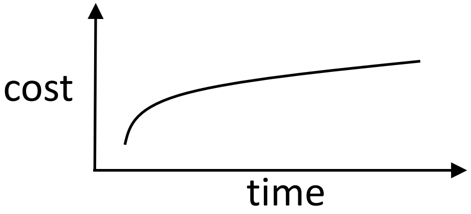
Design ($100)

Analysis (10$)

Requirements (1$)

# Why Lightweight Software Development Method ? (Contd..)

- *What if the cost remained flat?*
  - Possible because of improvements in technology and tools:
    - punch cards for inputs and batch processing in job submission vs. faster compilation and execution times
    - assembly vs. high-level programming languages
    - slow vs. fast hardware
    - IDEs, Cloud, many more…



- Because of the shorter turnaround time, you can let time answer questions and resolve uncertainties inherent in software development. What this means….

# Agile Methodology

- Delay investing in resources / plans that might never be used / realized. Ambiguity and volatility are inevitable

  *There is value in waiting*

- Implement upfront

  *Focus on code rather than the design*

  *Deliver working software quickly and adapt quickly*

- Get feedback and iterate

  *Prioritize People over Processes (esp. customer)*

- Focus on Simplicity (of design, implementation..)

  *Does not mean create inadequate software.*
  *"Look for the simplest thing <u>that works</u>"*

# Xtreme Programming (XP)

*XP is a lightweight methodology for small to medium sized teams developing software in the face of vague or rapidly changing requirements*
***-Kent Beck***

- 4 Attributes: lightweight, humanistic, disciplined, software development
- Guidelines and *Principles*:
    1. Write tests *( to get feedback)*
    2. Restructure code often (*to simplify, to show courage*)
    3. Talk to fellow programmers and customers often *(communicate)*

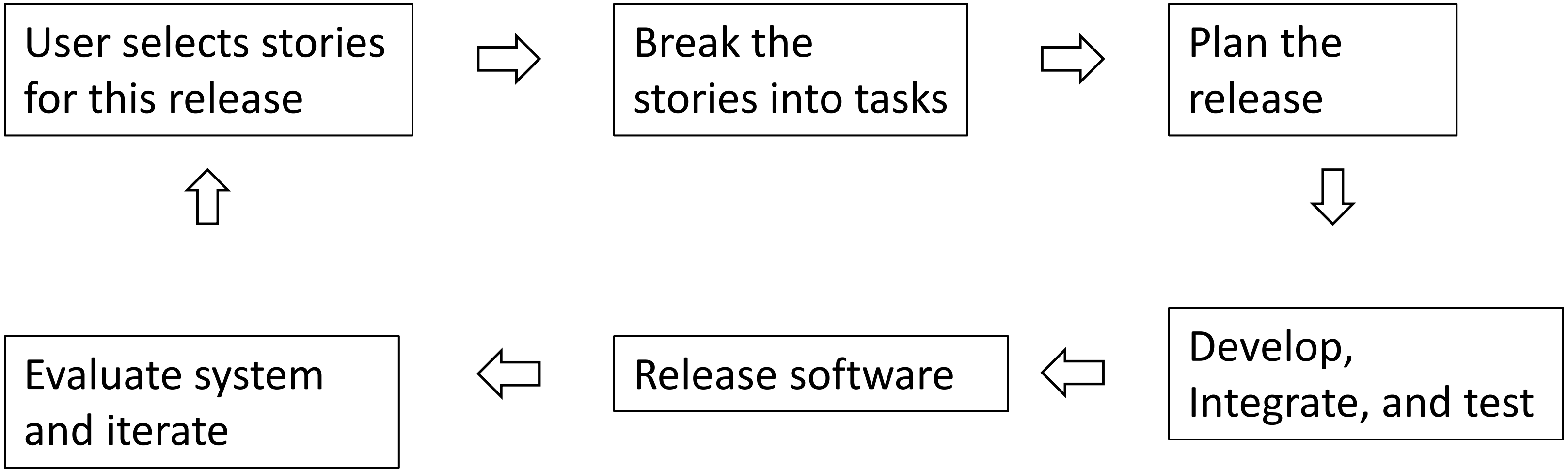
When you have to throw away code that doesn't work/unnecessary

# XP in Practice

- Incremental planning
- Small releases
- Simple design
- Test first
- Refactoring
- Pair programming
- Continuous Integration
- On-site customer

# Incremental Planning

- Assumes that the requirements are recorded on story cards, use cases, or scenarios.
- First, pick story (stories) for this release


User selects stories for this release

Break the stories into tasks

Plan the release

Develop, Integrate, and test

Release software

Evaluate system and iterate

# Small Releases

- Rather than focusing on a big release consisting of a lot of stories, focus on small releases
  - Helps deliver business value faster => builds customer confidence
  - Gives rapid feedback and hence, adapt quickly to changing requirements
  - Reduces risks and gives a sense of accomplishment to developers

# Simple Design

- Simple enough to just meet the requirements
  - No duplicated functionality
  - Fewest possible classes and methods
    - So adapting / changing is easier

# Test-First Development

- If there is a feature, write test case for the feature and test before writing the feature itself
    - Do this for unit tests as well
    - You see that test fail initially (obviously). As you add more functionality, tests start passing.

# Refactoring

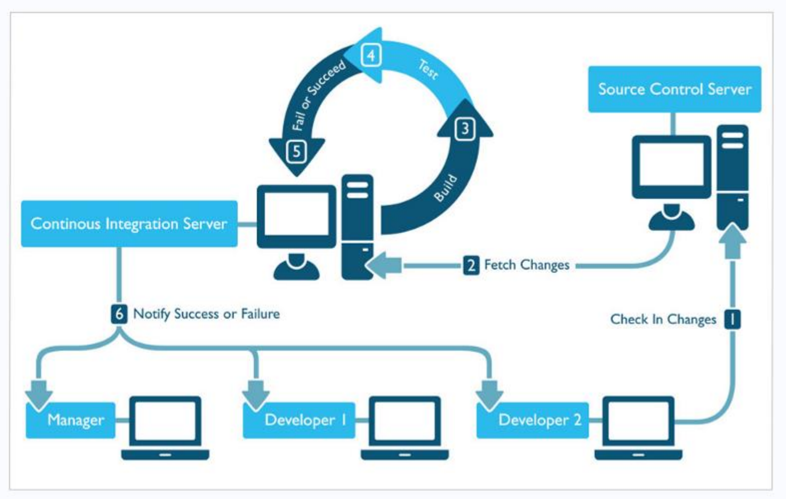
- Recall software refactoring from topics in software construction:
  - Transforming code to make it easier to read, maintain, and improve
- Refactoring is an important XP practice
- Done on-demand and not speculatively

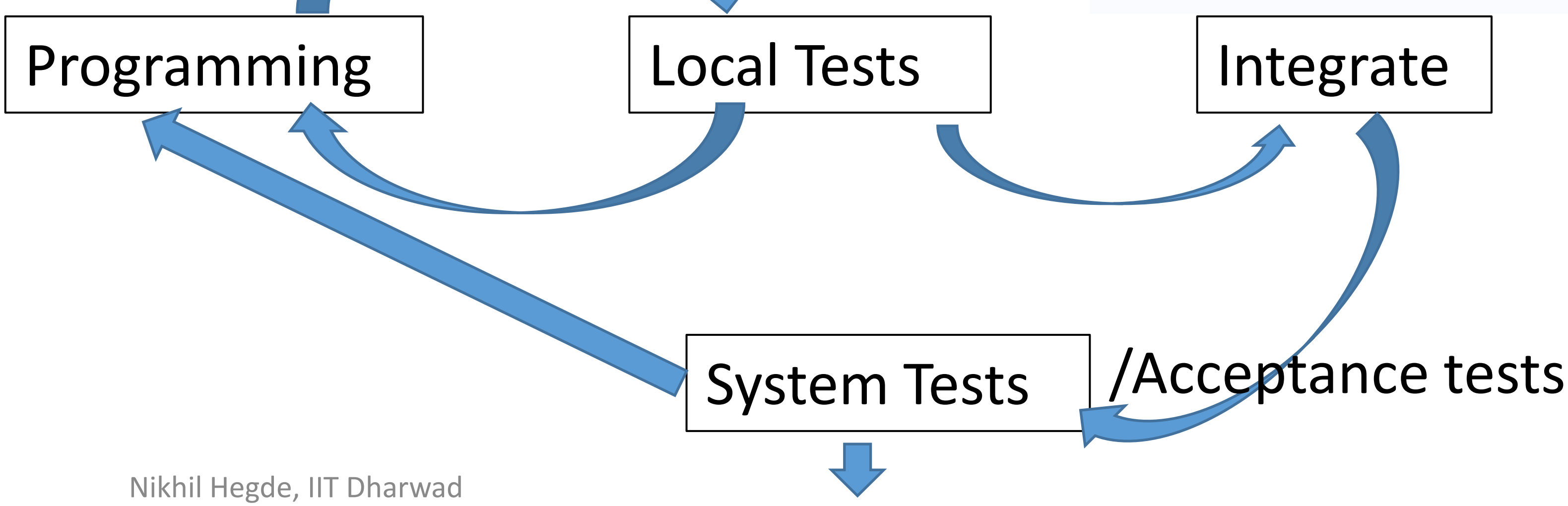
# Pair Programming

- All production code is written by two people looking at one machine (with one keyboard and one mouse)
- Study shows that productivity is equal to / better than two independent developers working
- Programmers play dual roles: programmer and strategizer (provider of out-of-context perspective)

# Continuous Integration

- Recall from Week13:
  - Ongoing monitoring from integration to testing to deployment





Programming

Local Tests

Integrate

System Tests

/Acceptance tests

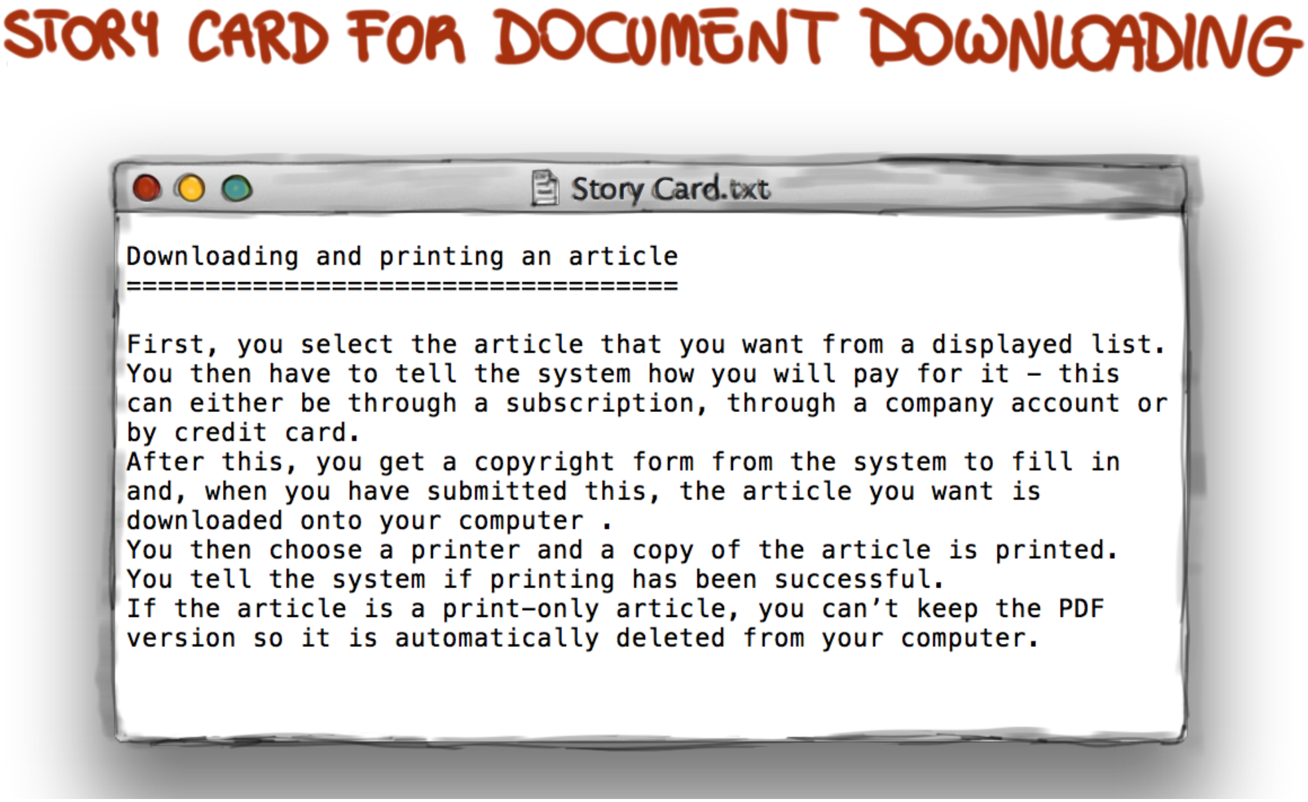
# On Site Customer

- Customer is part of the team
  - Brings the requirements
  - Sits with the team

*“If the system is not worth the time of one customer then it may not be worth building”*

# Requirements Engineering in XP

- Customer writes the requirements as story cards



STORY CARD FOR DOCUMENT DOWNLOADING

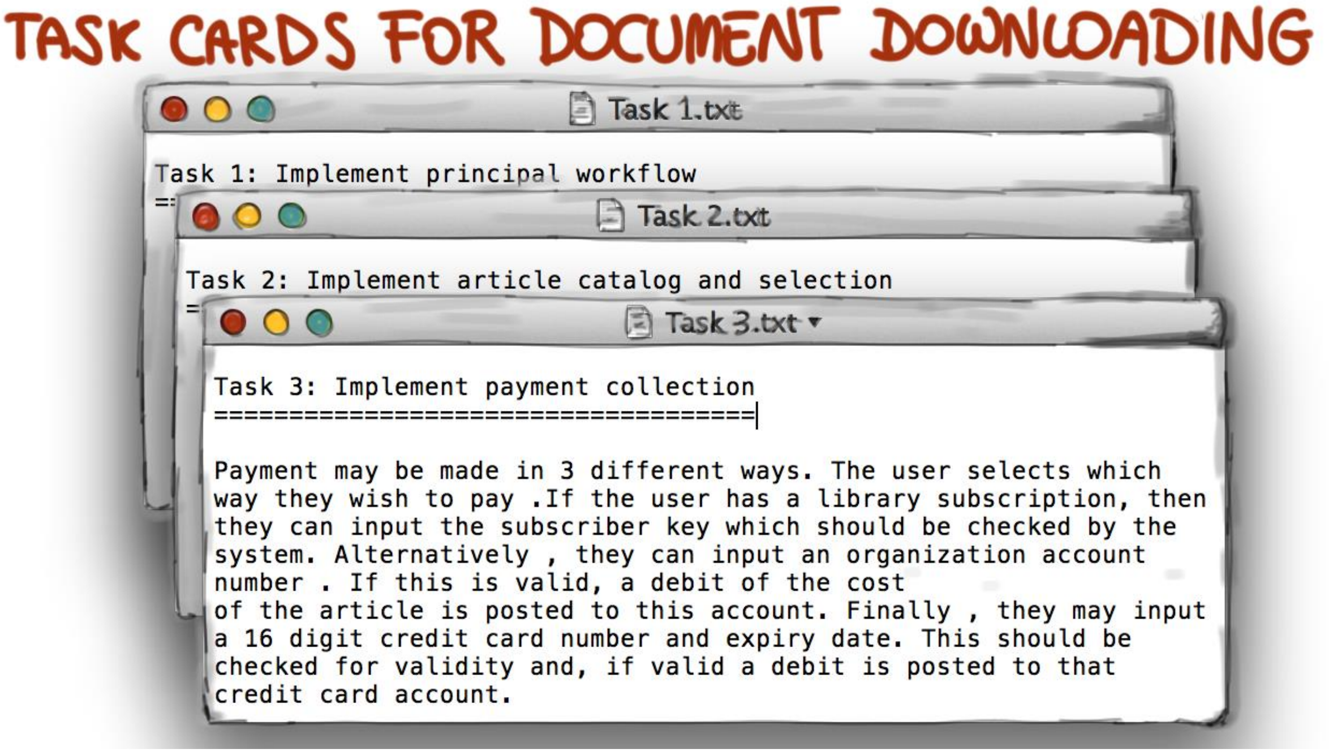
```
Downloading and printing an article
===================================

First, you select the article that you want from a displayed list.
You then have to tell the system how you will pay for it - this
can either be through a subscription, through a company account or
by credit card.
After this, you get a copyright form from the system to fill in
and, when you have submitted this, the article you want is
downloaded onto your computer .
You then choose a printer and a copy of the article is printed.
You tell the system if printing has been successful.
If the article is a print-only article, you can't keep the PDF
version so it is automatically deleted from your computer.
```

Source: Alex Orso, CS6300

# Requirements Engineering in XP

- The story cards are broken down into tasks and some tasks (story cards) are picked for next release



```
Task 1: Implement principal workflow
==
```

```
Task 2: Implement article catalog and selection
=
```

```
Task 3: Implement payment collection
====================================

Payment may be made in 3 different ways. The user selects which
way they wish to pay .If the user has a library subscription, then
they can input the subscriber key which should be checked by the
system. Alternatively , they can input an organization account
number . If this is valid, a debit of the cost
of the article is posted to this account. Finally , they may input
a 16 digit credit card number and expiry date. This should be
checked for validity and, if valid a debit is posted to that
credit card account.
```

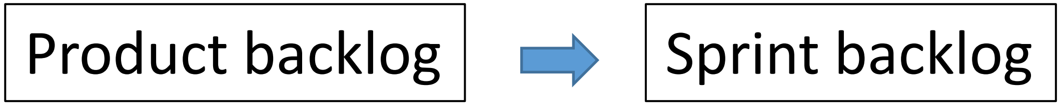
Source: Alex Orso, CS6300

# Scrum

- Extremely popular Agile methodology used in the industry
- 3 Actors:
    - Product Owner (is the Customer): responsible for listing and prioritizing backlogs (aka stories in XP)
    - Team: responsible for making software releases.
    - Scrum master: responsible for facilitating (meetings), removing obstacles.

# Scrum Process

Step 1:



Product backlog

Sprint backlog

via. sprint planning meet

Scrum master schedules a Sprint planning meeting involving product owner.

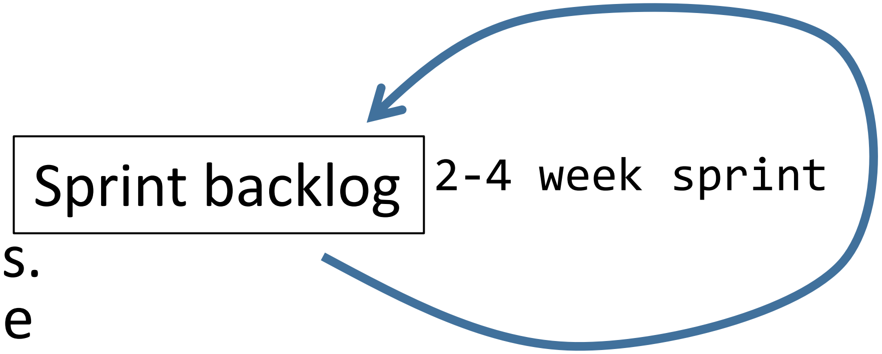
Decision on:

- Product backlogs that make it to Sprint backlog
- Breakdown of backlogs into tasks

# Scrum Process

## Step 2: Sprint

Sprint is an iteration in the process. It is the main activity that ships the release



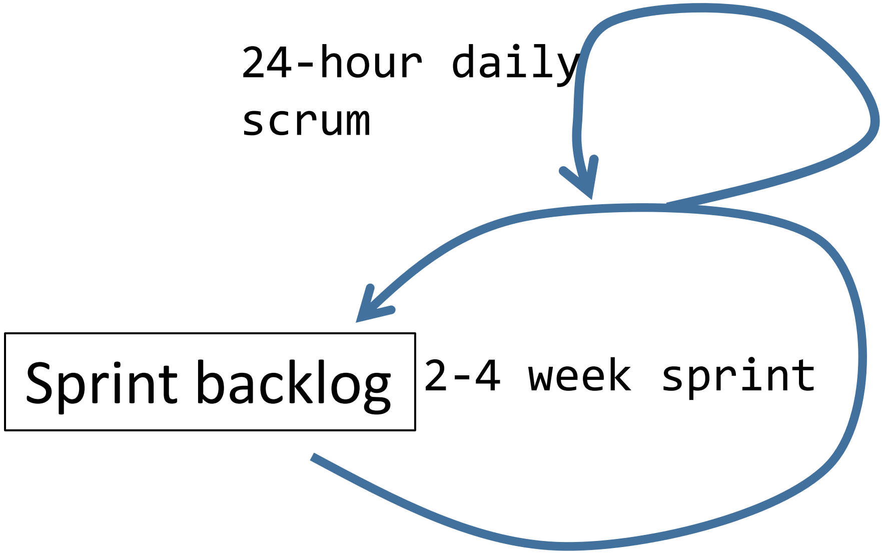
Sprint backlog

2-4 week sprint

## Step 2.1: Daily scrums:

15-mins team sync meeting:

- Tasks completed since last meeting
- TODO list for the next meeting
- Obstacles analysis



24-hour daily scrum

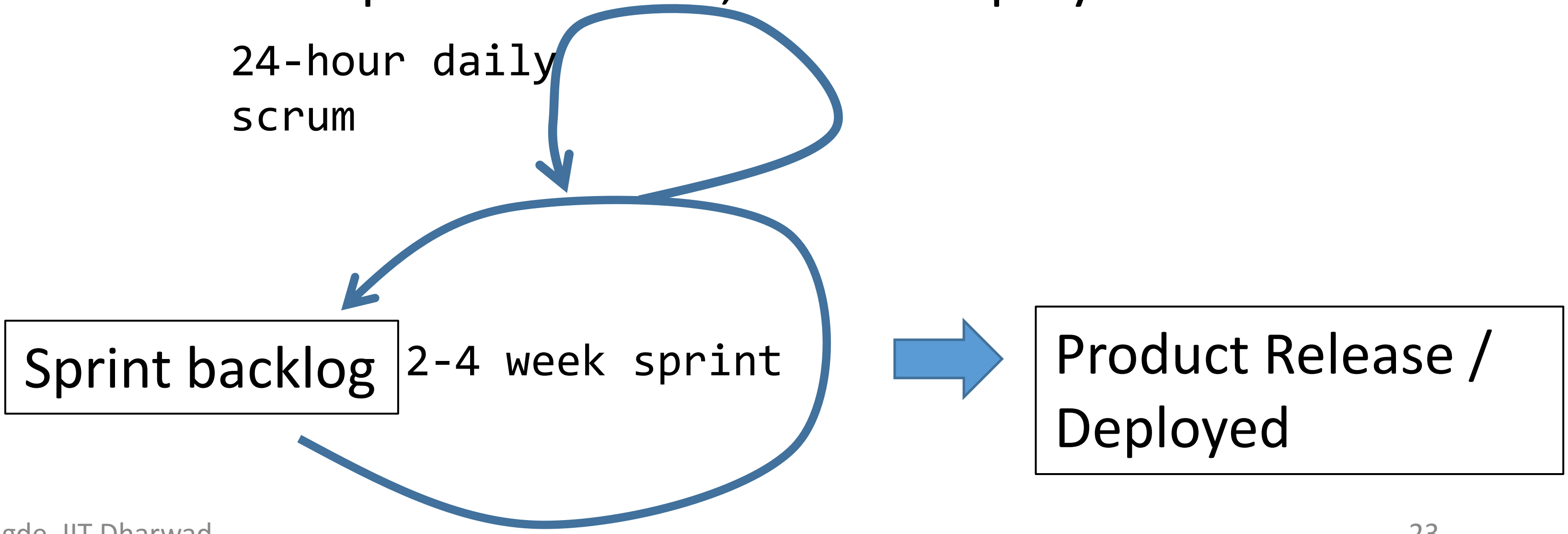
Sprint backlog

2-4 week sprint

# Scrum Process

## Step 3: Sprint Review and Retrospection

- Typically 3-4 hr meeting.
- Product owner assesses the accomplishments. Typically a demo is involved.
- If meets the acceptance criteria, can be deployed.



24-hour daily scrum

Sprint backlog

2-4 week sprint

Product Release / Deployed

# Now some (more) interesting stuff..

# Software Engineering for Machine Learning

- What do the software engineering processes look like while developing AI-based applications?

  1. Recall 'essential' and 'accidental' challenges in "*no silver bullet*" paper. What do the essential challenges look like while developing large-scale AI applications?
  2. What are three aspects of AI-based applications that make them different from traditional application domains?

  Amershi et. al. provide answers to the above questions in the paper: `Software Engineering for Machine Learning: A Case Study`, *appeared in* `ACM/IEEE ICSE-SEIP 2019.`

# AI-Apps vs. Traditional Apps

- What do the essential challenges look like while developing large-scale AI applications?

# AI-Apps vs. Traditional Apps

- What are three aspects of AI-based applications that make them different from traditional application domains?
  - Data centric: AI-based applications are all about data
  - Customizability and extendibility requires both SE and deep knowledge of ML skills
  - High coupling among modules

# Data Centric

- Traditional software apps (primarily) is about shipping code. For AI-based apps, data powers the models.
- Data is context-centric, voluminous, heterogeneous, and difficult to describe
  - How can version controlling be done for data?
  - Software APIs have specifications. Data rarely has metadata info associated (schemas, statistical distributions etc.)
  - Data changes rapidly (arrives ‘live’ sometimes)

# Customizability and Extendibility

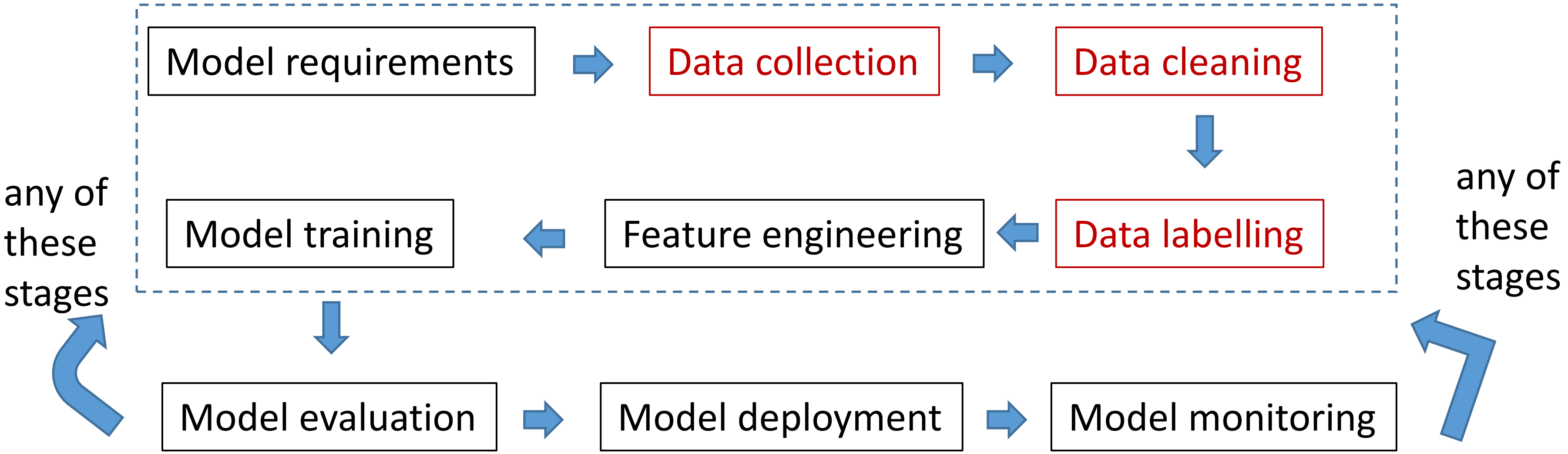
- Traditional Apps: customizing and reusing code components is made simple through elegant, modular, and simple design. *Is well understood.*
- AI-apps: have model (SVM, neural nets), parameters that control the model (support vectors, weights) and that are learned during training, and specific versions of data sets used for training
  - What if you want to use a model in a different domain and different data formats?

# Modularity

- Traditional apps: modules are separated and isolated so that development and maintaining is easy. APIs control the interaction among modules. APIs serve two purposes: keep interaction minimum, help maintain the separation
- AI-apps:
  - models are not extensible. E.g. you have a model identifying different kind of pizzas (from images). You also have a model for ordering pizzas (based on transcripts). Can't combine them.
  - Component entanglement – changes to one model in a part of the system affects other models in non-obvious ways

# Best Practices

- 9-stage Software Development Methodology



# Best Practices and Challenges

- End-to-end pipeline support for automation
  - Data pipelines for curating data
  - IDEs for development
- Data availability, collection, cleaning, and management
  - Blending data management tools with ML frameworks to avoid fragmenting data and model management activites
- Education and training
  - Engineers with tradition SE background need ML skills to work alongside ML specialists.

# Best Practices and Challenges

- Model debugging and interpretability
  - When and how models fail to make accurate predictions?
  - Develop more interpretable models and visualization techniques
- Model evolution, evaluation, and deployment
  - Employ rigorous and agile techniques to face the rapid changes that ML models go through
- Compliance
  - Fairness, accountability, transparency, and ethics

# Best Practices and Challenges

- Varied perceptions
  - What is a challenge depends on the amount of experience one has in building / integrating AI-capabilities

# TQM for Orgs. Building AI-apps

- Process Maturity Model for assessing large-scale AI-based app development capabilities
  - Hybrid of CMM and Six Sigma
    - S1: My team has **goals defined** for what to accomplish with this activity.
    - S2: My team does this activity in a **consistent manner**.
    - S3: My team has largely **documented** the practices related to this activity.
    - S4: My team does this activity mostly in an **automated** way.
    - S5: My team **measures and tracks** how effective we are at completing this activity.
    - S6: My team **continuously improves** our practices related to this activity.

# Concluding Remarks

- How has the recent trend in integrating AI capabilities in traditional large-scale apps affected job roles?
    - Traditionally: Program Manager, Team-development, Team-test
    - Adoption of DevOps (building and maintaining applications (and enabling platforms) supporting cloud computing means
        - Tester / developer role distinction is now seamless
        - IT, operations, and diagnostics are all integrated
        - New role of a data scientist has emerged. *Poly-math role* – also have to know and develop the model, bug-fix, support on cloud.

# Concluding Remarks

- Software Engineering is here to stay. Encompasses a wide variety of skills – frontend, backend, security skills, network admin skills etc.
- The skills developed for software engineering helps you build stuff. Help solve real-word problems
- Machine learning complements existing product rather than being the product itself