Week 6-2
Agile Software Development

COMPSCI 2ME3 Introduction to Software Development

** Slides adapted from previous instructors of COMPSCI 2ME3/SFWREGN 2AA4*

McMaster University

Week 6 Goals

- ~~**Monday**~~
  - ~~Software Processes: Part 2~~

Chapter 2 in Software Engineering Textbook

- **Wednesday**
  - **Agile Software Development: Part 1**

Chapter 3 in Software Engineering Textbook

- **Friday**
  - Irene's Research Talk

McMaster University

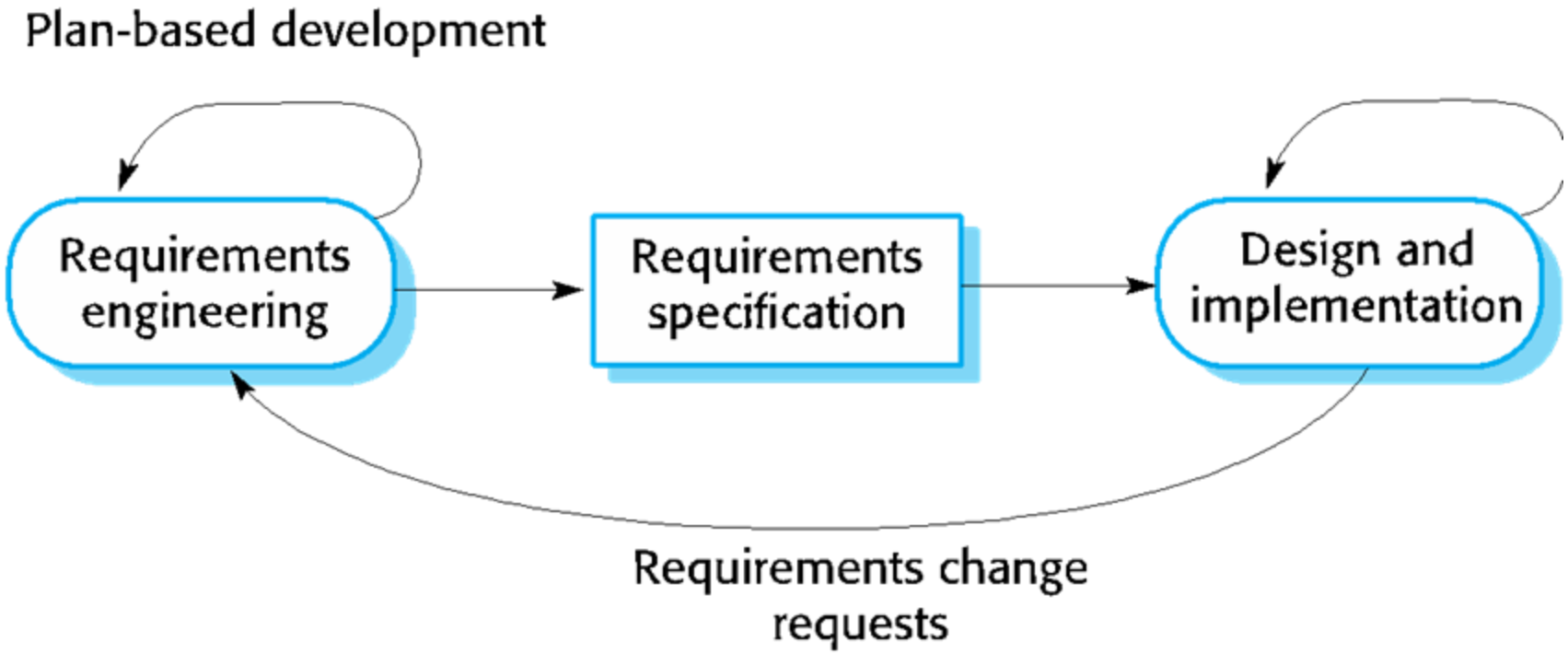
Plan-Driven Development Recap

Three traditional approaches we've studied:

- **Waterfall Model**
  - Sequential phases: Requirements, Design, Implementation, Testing, Deployment; Each phase must complete before the next begins
  - Best for stable, well-understood projects
- **Incremental Development**
  - System developed in increments, each increment adds new functionality
  - Best for projects where some requirements are known upfront
- **Integration and Configuration**
  - Reuse existing components and systems, focus on configuration and integration
  - Best for when suitable reusable components exist

Plan-Driven Development Recap

All three assume relatively stable requirements and sequential progression



Modern Software Dev Challenges

- A large number of software projects using traditional methods fail or are challenged
- Average time to market for enterprise software: 18-24 months
- By the time of delivery, almost half of the features are never actually used by customers

Why are nearly half of all built features never used?

Two fundamental problems that plan-driven models weren't designed to handle.

Challenge #1: Changing Requirements

Traditional models **assume we can plan and set requirements early,** where requirements constantly evolve (and that's actually a good thing)

- Instagram: Started as location check-in app "Burbn," completely pivoted based on user behavior
- Twitter: Began as podcasting platform "Odeo," reinvented when iTunes dominated podcasts

Challenge #1: Changing Requirements

Traditional models **assume we can plan and set requirements early,** where requirements constantly evolve (and that's actually a good thing)

- Instagram: Started as location check-in app "Burbn," completely pivoted based on user behavior
- Twitter: Began as podcasting platform "Odeo," reinvented when iTunes dominated podcasts

The Problem with Plan-Driven Models:

- Changes require returning to requirements phase
- Extensive documentation needs updating
- Previous work may be scrapped
- Change viewed as failure, not learning

Challenge #2: Speed Matters

How Software Development Cycles Have Accelerated:

- **1990s:** 2-3 year development cycles were acceptable
- **2000s:** 12-18 months to stay competitive
- **2010s:** 6-month cycles became the expectation
- **2020s:** Continuous delivery—daily or weekly updates

Challenge #2: Speed Matters

How Software Development Cycles Have Accelerated:

- **1990s:** 2-3 year development cycles were acceptable
- **2000s:** 12-18 months to stay competitive
- **2010s:** 6-month cycles became the expectation
- **2020s:** Continuous delivery—daily or weekly updates

What Happens If You Take 2 Years Today?

- Competitors beat you to market
- Technology becomes outdated
- User needs evolve beyond your solution
- Investor patience evaporates
- Market opportunity closes

Example: healthcare.gov

The Project:
US government healthcare enrollment website

- Budget: $1.7 billion
- Timeline: 3.5 years of development
- Approach: Traditional waterfall methodology
- Expectation: 50,000 simultaneous users

Launch Day, October 1, 2013:

- 250,000 people tried to access the site
- Only 6 people successfully enrolled on day one
- Website crashed repeatedly
- Error messages everywhere
- National embarrassment and political crisis

Example: healthcare.gov

1. Requirements Locked Too Early

- Specifications finalized years before launch
- No mechanism to incorporate learning
- Assumptions about usage patterns completely wrong

2. No User Testing Until Too Late

- First real user interaction at launch
- No opportunity to discover problems early
- No iterative refinement

3. Technology Changed

- Development started in 2010, launched 2013
- Web technologies evolved significantly
- Original choices became outdated

4. No Adaptive Capacity

- Built for 50K users, needed to handle 250K
- Couldn't scale or adjust when reality hit
- No working version to test until the very end

Modern Software Dev Challenges

Challenge #1: Requirements Always Change

- User needs evolve
- Markets shift
- Technology advances
- Competitors innovate
- Learning happens through building

Challenge #2: Speed Matters More Than Ever

- First to market wins
- Continuous delivery expected
- Feedback loops must be fast
- Slow = irrelevant

Modern Software Dev Challenges

Challenge #1: Requirements Always Change

- User needs evolve
- Markets shift
- Technology advances
- Competitors innovate
- Learning happens through building

Challenge #2: Speed Matters More Than Ever

- First to market wins
- Continuous delivery expected
- Feedback loops must be fast
- Slow = irrelevant

We Need:

- Embrace and accommodate change
- Deliver value quickly and frequently
- Continuous customer feedback
- Adapt based on learning

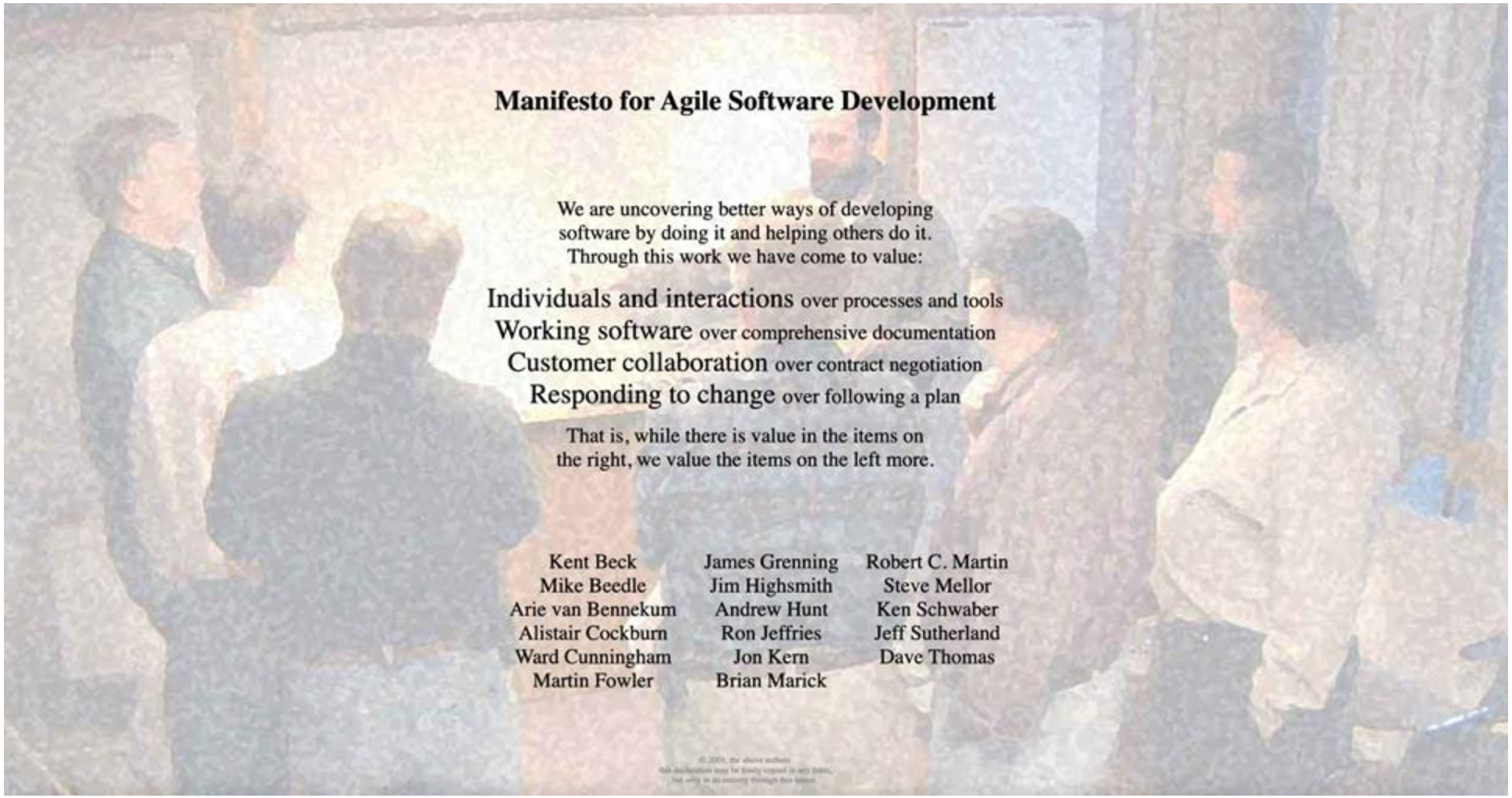
Agile Software Development

17 software developers all frustrated with heavy-weight methodologies, met and developed the manifesto for Agile Software Development

The Impact of Agile:

- Fundamentally changed how software is built
- Spread rapidly across the industry
- Now the dominant approach for most software development
- Spawned entire ecosystem of practices, tools, and certifications

Agile Development



https://agilemanifesto.org/

Four Agile Manifesto Values

1. **Individuals and Interactions Over Processes and Tools:** People and communication matter most; adapt process to the team, not team to process
2. **Working Software Over Comprehensive Documentation**: Build working software first; document what's necessary, not everything possible
3. **Customer Collaboration Over Contract Negotiation:** Ongoing partnership throughout development; adapt to learning together
4. **Responding to Change Over Following a Plan:** Plan, but expect plans to evolve; change means we're learning

Agile is Not Chaos

While there is value in the items on the left, we value the items on the tight **more**.

Agile Is NOT:

- No planning
- No documentation
- No process
- No discipline
- Just "winging it"
- Excuses for poor quality

Agile IS:

- Balanced approach
- Appropriate documentation
- Lightweight process
- High discipline
- Deliberate adaptation
- Quality built in

Twelve Principles of Agile

1. Our highest priority is to satisfy the customer through **early and continuous delivery** of valuable software.
2. **Welcome changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from **a couple of weeks to a couple of months**, with a preference to the shorter timescale.
4. **Business people and developers must work together daily** throughout the project.
5. Build projects around **motivated individuals**. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is **face-to-face conversation**.

1. **Working software** is the primary measure of progress.
2. Agile processes promote **sustainable development**. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
3. Continuous attention to **technical excellence** and good design enhances agility.
4. **Simplicity**--the art of maximizing the amount of work not done--is essential.
5. The best architectures, requirements, and designs emerge from **self-organizing teams**.
6. At regular intervals, the team reflects on how to become more effective, then **tunes and adjusts its behavior accordingly**.

Key Principles of Agile

Principle	Description
Customer Focus	Customers **should be closely involved throughout the development process**. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Frequent Delivery	The software is developed in increments **with the customer specifying the requirements** to be included in each increment.
Collaboration & Communication	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace Change	Expect the **system requirements to change** and so design the system to accommodate these changes.
Sustainable Pace & Simplicity	**Focus on simplicity** in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

The Fundamental Mindset Shift

Follow the plan → **Adapt to change**
Big delivery → **Iterative releases**
Documentation-heavy → **Working software**
Isolated teams → **Collaborative culture**
Hero developers → **Sustainable pace**
Change is failure → **Change is learning**
Process over people → **People over process**
Contract protection → **Customer partnership**
Comprehensive documentation → **Just enough documentation**
Predict and control → **Inspect and adapt**

Discussion #1

What Does an Agile Developer's Day Look Like?

Imagine you're a software developer on an agile team building a mobile food delivery app

Think about:
What time do you start?
When do you actually write code?
What meetings might you have?
How do you know what to work on?
Who do you talk to during the day?

Discussion #1

A Traditional Developer's Day

9:00am - Check email, attend status meeting
10:00am - Wait for requirements clarification
11:00am - Code (alone at desk)
12:00pm - Lunch
1:00pm - More solo coding
3:00pm - Weekly team meeting to discuss blockers
4:00pm - Write documentation
5:00pm - Go home

Discussion #1

A Traditional Developer's Day

9:00am - Check email, attend status meeting
10:00am - Wait for requirements clarification
11:00am - Code (alone at desk)
12:00pm - Lunch
1:00pm - More solo coding
3:00pm - Weekly team meeting to discuss blockers
4:00pm - Write documentation
5:00pm - Go home

An Agile Developer's Day

9:00am - Daily Standup (15 minutes) 9:15am - Pair programming on user story 11:00am - Quick code review with teammate
12:00pm - Lunch
1:00pm - Continue development (solo or paired)
2:00pm - Brief chat with Product Owner to clarify story
3:00pm - Write automated tests
4:00pm - Demo completed feature to team
4:30pm - Backlog grooming meeting (2x per week)
5:00pm - Sprint retrospective (every 2 weeks)

Agile Practices: Daily Standup

What It Is:

- Entire team gathers (stand in circle or video call)
- Exactly 15 minutes, every day
- Same time, same place

The Three Questions: Each person answers:

- What did I do yesterday?
- What will I do today?
- What's blocking me?

Purpose:

- Coordination and transparency
- Quick identification of problems
- Everyone knows what everyone else is doing
- Blockers surface immediately

Agile Practices: Pair Programming

How It Works:
Two developers sit together
- **Driver:** Types and implements
- **Navigator:** Reviews, thinks ahead, suggests

Switch roles frequently (every 30-60 min)

Actually Effective:
- **Immediate bug detection:** Navigator catches errors as they happen
- **Better design:** Two brains thinking through problems
- **Knowledge sharing:** Both learn the codebase
- **Natural mentoring:** Senior helps junior
- **Fewer interruptions:** Working together maintains focus
- **Higher code quality:** Built-in code review

Agile Practices: Agile Ceremonies

Backlog Grooming (1-2x per week, 1-2 hours):

- Team + Product Owner
- Review upcoming work
- Break down large features into tasks
- Estimate effort required
- Clarify requirements

Sprint Review (Every 2-4 weeks, 1-2 hours):

- Demo working software to stakeholders
- Show what was completed this sprint
- Get feedback
- Discuss what to build next

Agile Practices: Agile Ceremonies

Sprint Retrospective (Every 2-4 weeks, 1-2 hours):

- Team only (no managers ideally)
- What went well this sprint?
- What didn't go well?
- What will we improve next sprint?
- Focus on process improvement

Sprint Planning (Start of each sprint, 2-4 hours):

- Team commits to work for next 2-4 weeks
- Break down user stories into tasks
- Discuss how to implement

Agile Development Practices

More Collaboration:

- Daily standup with entire team
- Pair programming sessions
- Quick chats with Product Owner
- Code reviews with teammates
- Regular demos and retrospectives

Shorter Feedback Loops:

- Know immediately if blocked (daily standup)
- Bugs caught in minutes (pair programming)
- Features reviewed within days (sprint review)
- Process improved every 2-4 weeks (retrospective)

Less Waiting:

- Product Owner available for questions
- Team can make decisions quickly
- No long approval chains
- Self-organizing authority

Continuous Communication:

- Face-to-face preferred
- Informal and frequent
- Rich, synchronous interaction
- Build relationships and trust

Discussion #2

What Roles Are Needed on an Agile Team?

You're forming an agile team to build that food delivery app**.**

Think about:
Who decides WHAT features to build?
Who actually writes the code?
Who manages the team?
Who talks to customers?
Any other roles needed?

Discussion #2

Agile teams, especially using Scrum, have three core roles:

- **Product Owner** - The "WHAT" person
- **Scrum Master** - The "HOW" person
- **Development Team** - The "BUILD" people

Key Characteristics:

- Flat structure, not hierarchical
- Each role has specific authority
- Shared accountability for results
- Collaboration over command-and-control

Agile Roles: Product Owner

Primary Responsibility: Decide WHAT features to build and in what order

Key Duties:

- Manage the product backlog (prioritized list of work)
- Write and define user stories (requirements)
- Prioritize features based on business value
- Accept or reject completed work
- Represent customer and stakeholder needs
- Make trade-off decisions
- Ensure team builds the right thing

Who Is This Person?

- Business/domain expert, NOT a developer
- Deep understanding of customers and market
- Decision-making authority
- Available to team daily

Agile Roles: Scrum Master

Primary Responsibility: Ensure team follows agile practices and remove obstacles

Key Duties:

- Facilitate all agile meetings (standup, retrospective, planning, review)
- Remove blockers that slow team down
- Shield team from distractions and interruptions
- Coach team on agile practices
- Ensure process is working well
- Servant leader to the team

Who Is This Person?

- Agile expert and facilitator
- NOT a traditional project manager
- No authority to assign tasks
- Focused on how team works together

Agile Roles: Development Team

Primary Responsibility: Build working software
Typical Composition (3-9 people):

- Frontend developers
- Backend developers
- QA/Testers
- UX/UI designers (sometimes)
- Database specialists
- DevOps engineers

Key Characteristics:

- **Cross-Functional:** All skills needed to deliver working software; No dependencies on people outside the team
- **Self-Organizing:** Team decides HOW to do the work; No task assignment from above; Members volunteer for tasks based on skills, interest, capacity Collective ownership of code
- **Collaborative:** Not individual heroes, but team success; Share knowledge and help each other; Collectively responsible for outcomes

Agile Roles

Stakeholders (Customers, Executives, Users)

Product Owner  Scrum Master (WHAT to build) (HOW to work)

Development Team (BUILD it)
Frontend Devs
Backend Devs
QA/Testers
Designers

Scenario 1: A manager from another department keeps interrupting developers during the sprint with urgent requests that aren't in the sprint backlog.
Who should hanlde this?

Scenario 2: The team needs to decide whether to use React or Angular for the frontend framework.
Who should handle this?

Agile Summary

Overall, agile development roles and practices are set up to **emphasize**:

- Constant collaboration over isolated work
- Face-to-face communication over documentation
- Quick decisions over long approval chains
- Trust and empowerment over command-and-control

Week 6 Goals

- ~~**Monday**~~
  - ~~Software Processes: Part 2~~

Chapter 2 in Software Engineering Textbook

- ~~Wednesday~~
  - ~~Agile Software Development: Part 1~~

Chapter 3 in Software Engineering Textbook

- **Friday**
  - **Irene's HCI Research Talk**

McMaster University