GitHub in Software Engineering

CS450: Modern Software Engineering

Project Kickoff

What is Agile?

- **Agile** = a **software engineering methodology**
- Focuses on:
  - **Flexibility**
  - **Collaboration**
  - **Iterative progress**
- Response to rigid, heavyweight methods (e.g., **Waterfall**)

Agile Manifesto (2001)

4 Values:

1. **Individuals & interactions** > processes & tools
2. **Working software** > documentation
3. **Customer collaboration** > contracts
4. **Responding to change** > following a plan

12 Supporting Principles (Highlights)

- Deliver working software **frequently**
- Welcome **changing requirements**
- Maintain a **sustainable pace**
- Encourage **face-to-face communication**
- Regularly reflect & **adapt**

Key Concepts

- **Iterative development**: short cycles (sprints/iterations)
- **Incremental delivery**: shippable functionality each cycle
- **User stories**: features in user terms
- **Backlog**: prioritized work list
- **Velocity**: team’s work rate
- **Definition of Done (DoD)**: agreed completion criteria

Scrum Framework

- Timeboxed **sprints** (1–2 weeks)
- Roles: **Product Owner**, **Scrum Master**, **Dev Team**
- Events: Planning, **Daily Standup**, Review, Retrospective

Kanban

- **Visualizes work** with a board
- Continuous delivery (no sprints)

Agile vs. Waterfall

Agile	Waterfall
Iterative, incremental	Linear, sequential
Embraces change	Resists change
Software early & often	Software late
Collaboration-driven	Contract-driven
Adaptive planning	Predictive planning

Benefits of Agile

- **Flexibility** to adapt
- **Faster feedback loops**
- **Closer customer involvement**
- **Improved quality** via testing/integration
- **Team empowerment** through self-organization

> ## https://github.com/wwu-cs450

Public Repositories

If you want private use Gitlab

Project Journal

Setup: `journal.md`

Course Overview

GitHub features across 5 key phases:

1. **Requirements & Ideation**
2. **Design & Architecture**
3. **Implementation**
4. **Testing & QA**
5. **Deployment & Maintenance**

Phase 1: Requirements & Ideation

GitHub Issues for Requirements Management

Key Features:

- Issue templates
- Labels and milestones
- Projects (Kanban boards)
- Discussions

GitHub Issues Documentation

Creating Issue Templates

Setup: `.github/ISSUE_TEMPLATE/user-story.md`

```
---
name: User Story
about: Capture a user requirement
---

## User Story
As a [type of user], I want [goal] so that [benefit]

## Acceptance Criteria
- [ ] Criterion 1
- [ ] Criterion 2
```

Issue Templates Guide

GitHub Projects for Planning

```
┌────────────┬─────────┬───────────────┬─────────┐
│  Backlog   │  Todo   │  In Progress  │  Done   │
├────────────┼─────────┼───────────────┼─────────┤
│ Issue #12  │ #5      │ #3            │ #1      │
│ Issue #15  │ #7      │ #8            │ #2      │
│ Issue #18  │         │               │ #4      │
└────────────┴─────────┴───────────────┴─────────┘
```

GitHub Projects provide Kanban-style workflow management

GitHub Projects Documentation

GitHub Discussions for Ideation

Use **Discussions** for:

- Feature brainstorming
- Architecture decisions
- Q&A with stakeholders
- Knowledge base

```
Ideas → Discussion → Issue → Implementation
```

GitHub Discussions

Phase 2: Design & Architecture

Documentation in Repository

Key Practices:

- Architecture Decision Records (ADRs)
- Wiki for technical specs
- README-driven development
- Diagrams in Markdown

Architecture Decision Records

Structure: `docs/adr/0001-database-choice.md`

```markdown
# ADR 1: Use PostgreSQL for Primary Database

## Status
Accepted

## Context
We need a relational database for transaction support...

## Decision
We will use PostgreSQL 15+

## Consequences
+ Strong ACID compliance
- Additional deployment complexity
```

GitHub Wiki for Architecture

```
Project Wiki Structure:
├── Home
├── System Architecture
│   ├── Component Diagram
│   └── Data Flow
├── API Documentation
└── Deployment Guide
```

Benefits: Version controlled, searchable, collaborative

GitHub Wiki Guide

Mermaid Diagrams in Markdown

```
graph LR
    A[Client] --> B[API Gateway]
    B --> C[Auth Service]
    B --> D[Business Logic]
    D --> E[(Database)]
```

GitHub renders Mermaid diagrams natively!

Mermaid Documentation

Phase 3: Implementation

Branching & Pull Requests

Key Concepts:

- Git Flow / GitHub Flow
- Protected branches
- Code review process
- Branch policies

GitHub Flow

```
main ───────────────────────────────────────
     \            \           \
      feature-A    fix-bug     feature-B
           │         │             │
           └─────────┴─────────────┘
              Pull Requests
```

Simple workflow:

1. Create branch from `main`
2. Make commits
3. Open Pull Request
4. Review & discuss
5. Merge to `main`

Pull Request Template

Setup: `.github/PULL_REQUEST_TEMPLATE.md`

```
## Changes
Brief description of changes

## Related Issues
Closes #123

## Testing
- [ ] Unit tests pass
- [ ] Manual testing completed

## Screenshots
(if applicable)
```

PR Templates

Branch Protection Rules

Configure on GitHub:

- Require PR reviews (2+ approvers)
- Require status checks
- Require conversation resolution
- No force pushes
- Require linear history

Branch Protection

GitHub Copilot & Code Suggestions

AI-Powered Development:

- Code completion
- Test generation
- Documentation writing
- Code explanation

GitHub Copilot

Phase 4: Testing & QA

GitHub Actions for CI/CD

Continuous Integration:

- Automated testing
- Linting & formatting
- Security scanning
- Build verification

GitHub Actions

Basic CI Workflow

File: `.github/workflows/ci.yml`

```yaml
name: CI
on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Setup Node
        uses: actions/setup-node@v4
      - run: npm test
      - run: npm run lint
```

Workflow Syntax

Status Checks on PRs

```
Pull Request #42: Add user authentication
├─ ✓ Build (2m 34s)
├─ ✓ Unit Tests (1m 12s)
├─ ✓ Lint (34s)
├─ ✗ Code Coverage (below 80%)
└─ ⏳ Security Scan (running...)
```

Block merging until all checks pass

Status Checks

Security Scanning

GitHub Security Features:

- Dependabot (dependency updates)
- Secret scanning

GitHub Security

Phase 5: Deployment & Maintenance

Releases and Versioning

Key Features:

- Semantic versioning
- Release notes
- Asset distribution
- Changelog generation

Releases

Creating Releases

Use GitHub's Releases UI or automate with Actions

Best Practices Summary

Do:

- Write clear commit messages
- Keep PRs small and focused
- Review code thoroughly
- Document decisions (ADRs)
- Automate repetitive tasks
- Use branch protection

Best Practices Summary

Don't:

- Commit directly to main
- Merge without reviews
- Ignore CI failures
- Leave issues unorganized

Workflow Integration

```
Requirements (Issues)
    ↓
Design (Wiki/ADRs)
    ↓
Implementation (Branches/PRs)
    ↓
Testing (GitHub Actions)
    ↓
Deployment (Environments)
    ↓
Maintenance (Insights/Issues)
```

GitHub provides tools for every phase

Additional Resources

- GitHub Skills - Interactive learning
- GitHub Docs - Complete documentation
- GitHub Blog - New features & best practices

Team Project Assignment

CS450 Project:

1. Create organization for your team
2. Set up repository with templates
3. Establish branching strategy
4. Configure CI/CD pipeline
5. Create first release!