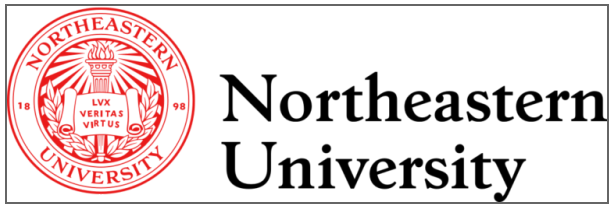
CS 7180: Agile/Scrum + Pair Workflow



John Alexis Guerra Gomez
jguerra at northeastern.edu
Class: **johnguerra.co/classes/aiCoding_spring_2026**
Slides: **johnguerra.co/lectures/ai_assisted_coding**

What We'll Cover Today

1. Where We Are -- Week 7 checkpoint
2. Agile/Scrum Crash Course
3. GitHub as Your Scrumboard
4. From PRD to Sprint Backlog
5. Branches, PRs, and Code Review
6. Pair Workflow with AI
7. Code Review Between Partners

Where We Are

> *Week 7 -- Process for working in pairs*

Recap: Week 6 Foundations

Last week you learned:

- How IDE AI tools work (context collection, indexing, embeddings)
- Code suggestions, inline edit (Cmd+K), chat panel
- Modes: Ask / Write / Agent / Plan
- Rules files and @ context references
- Tool comparison: Antigravity vs Copilot vs Cursor
- **P2 pair formation** -- you should have a partner and a Canvas group

You have the tools and a partner. Now you need the *process* to work effectively together.

This Week: Process for Pairs

Two themes today:

1. **Agile/Scrum** -- How professional teams organize work (you'll need this for P2 and P3)
2. **Pair workflow** -- How to split work, avoid conflicts, and review each other's code

Agile gives you the *structure*, your partner gives you *accountability*, and AI gives you *speed*.

Agile/Scrum Crash Course

> *The minimum you need to run effective sprints*

Why Agile for AI Projects?

AI changes the speed of coding, not the need for process.

- You can generate code faster than ever -- but *what* should you build?
- Without process, AI speed leads to building the wrong thing faster
- Agile gives you **short feedback loops** to course-correct
- Perfect fit: sprints align with project milestones in this course

P2 requires 2 documented sprints. P3 requires 4.

Scrum Roles

Role	Responsibility	In This Course
Product Owner	Decides *what* to build, prioritizes backlog	One partner takes lead (rotate per sprint)
Scrum Master	Facilitates process, removes blockers	The other partner (rotate per sprint)
Development Team	Builds the product	Both partners + your AI tools

In P2, you split roles between partners. In P3 teams, you'll have more flexibility.

The Sprint Cycle

```
Sprint Planning (start of sprint)
    ↓
Daily Work (build, test, commit)
    ↓
Sprint Review (demo what you built)
    ↓
Sprint Retrospective (what to improve)
    ↓
Next Sprint Planning...
```

Sprint length in this course: 1-2 weeks per sprint.

The key insight: you commit to a *small, achievable* set of work each sprint.

Scrum Ceremonies

Ceremony	When	Duration	Purpose
Sprint Planning	Start of sprint	30-60 min	Pick issues for this sprint
Daily Standup	Every day	5-15 min	What did I do? What will I do? Blockers?
Sprint Review	End of sprint	30 min	Demo working software
Retrospective	After review	15-30 min	What went well? What to improve?

For P2: Document your planning and retro in your README or project wiki. Partner standups count -- at least 3 per sprint.

GitHub as Your Scrumboard

> *Issues, Projects, and Labels -- everything you need*

GitHub Issues = Backlog Items

Each piece of work becomes a GitHub Issue:

- **Title:** Short, action-oriented ("Add user login page")
- **Description:** Acceptance criteria, design notes, links
- **Labels:** `feature`, `bug`, `chore`, `sprint-1`, `sprint-2`
- **Assignee:** Who's working on it
- **Milestone:** Which sprint or release

Issues are your single source of truth for what needs to be built.

GitHub Projects = Sprint Board

GitHub Projects (Board view) gives you a Kanban board:

Backlog	Sprint Todo	In Progress	In Review	Done
Future work	This sprint's work	Actively coding	PR open	Merged & deployed

Setup:

1. Create a Project in your repo (Board layout)
2. Add columns: Backlog, Todo, In Progress, Review, Done
3. Drag issues between columns as work progresses
4. Filter by sprint milestone to see current sprint only

Labels & Milestones for Sprint Tracking

Labels categorize work:

- `feature` / `bug` / `chore` / `docs`
- `priority: high` / `priority: low`
- `sprint-1` / `sprint-2`

Milestones group issues into time-boxed sprints:

- Create a milestone for each sprint with a due date
- Assign issues to milestones during sprint planning
- Track progress via the milestone's completion percentage

For P2: Create at least 2 milestones (Sprint 1, Sprint 2). Each should have 5+ issues.

From PRD to Sprint Backlog

> *Connecting what you learned in Weeks 3-4 to how you build in Weeks 7+*

Revisiting the PRD

Remember your PRD from Weeks 3-4? It contains:

- Problem statement and target users
- User stories ("As a ___, I want ___ so that ___")
- Acceptance criteria for each story
- Technical architecture decisions

Now it's time to turn that PRD into actionable work items.

Your PRD is the *what*. Your sprint backlog is the *when* and *how*.

Breaking PRD into GitHub Issues

Each user story becomes one or more GitHub Issues:

```
PRD User Story:
  "As a user, I want to log in with email
   so that my data is saved across sessions"

GitHub Issues:
  #1  Set up authentication library (chore)
  #2  Create login page UI (feature)
  #3  Implement email/password auth endpoint (feature)
  #4  Add session persistence (feature)
  #5  Write login flow tests (chore)
```

Rule of thumb: Each issue should be completable in 1-4 hours. If it's bigger, break it down further.

Sprint Planning: Picking Issues

In sprint planning, you and your partner:

1. Review the backlog together (all open issues)
2. Estimate effort (small / medium / large)
3. Pick issues that fit the sprint's capacity
4. **Assign each issue to one partner** -- clear ownership
5. Assign them to the sprint milestone

For Sprint 1 of P2, aim for:

- Core data model and API setup
- 1-2 key features (login, main CRUD operation)
- Basic test suite setup
- Shared rules file established

Branches, PRs, and Code Review

> *Professional workflow with AI assistance*

Branch-per-Issue Workflow

Every issue gets its own branch:

```
# Create branch from issue number
git checkout -b feature/42-add-login-page

# Work on the feature...
git add .
git commit -m "Add login page UI (#42)"

# Push and create PR
git push -u origin feature/42-add-login-page
```

Naming convention: `type/issue#-short-description`

- `feature/42-add-login-page`
- `fix/57-null-avatar-crash`
- `chore/63-update-dependencies`

PRs and AI-Assisted Review

Pull Request best practices:

- **Link to issue:** "Closes #42" in the PR description auto-closes the issue on merge
- **Small PRs:** Easier to review, fewer conflicts, faster to merge
- **PR template:** Add a template in `.github/pull_request_template.md`
- **AI-assisted review:** Use your IDE AI to review diffs before pushing

AI-assisted self-review before creating PR:

```
"Review this diff for bugs, security issues, and style violations.
 Our project uses TypeScript, React, and follows the patterns in
 @.antigravityrules"
```

Pair Workflow with AI

> *Scrum for two -- agile process meets AI-assisted development*

The Pair + AI Workflow

Your pair runs scrum, AI assists both partners:

Activity	How It Works	AI's Role
Sprint Planning	Meet together, pick issues from backlog	Helps break down stories into tasks
Development	Each partner owns assigned issues	Real-time coding assistance per partner
Code Review	Every PR reviewed by your partner	Pre-review via AI, human catches intent
Standups	Async check-ins on progress & blockers	Summarize changes, draft standup notes

When to pair vs. split work:

- **Pair (synchronous):** Complex features, architecture decisions, debugging hard issues
- **Split (async):** Independent features with clear interfaces, tests, documentation

Your **design thinking** and **mom test** skills feed the backlog -- user feedback drives sprint priorities.

Splitting Work Without Conflicts

How to avoid stepping on each other's toes:

1. **Assign issues to one person** -- never both working on the same file
2. **Define interfaces first** -- agree on API shapes, component props, data models before splitting
3. **Use feature branches** -- one branch per issue, never commit directly to main
4. **Merge frequently** -- don't let branches diverge for days
5. **Communicate blockers** -- async standups keep both partners aware

Anti-pattern: Both partners editing the same file in parallel. This leads to merge conflicts and wasted time.

Async Standups for Pairs

Not everyone can meet daily. Use async standups:

Post in your shared channel (Slack DM, GitHub Discussion, or project wiki):

```
🗓️ Date: [today]
✅ Yesterday: Completed #42 login UI, started #43 auth endpoint
🎯 Today: Finish auth endpoint, write tests for login flow
🚧 Blockers: Waiting on partner's DB schema PR (#40) to merge
```

For P2: At least 3 standups per sprint from each partner. These can be async messages.

Code Review Between Partners

> *Every PR gets a human review*

Partner Code Review Workflow

Every PR reviewed by your partner before merge:

```
1. Developer creates PR (links to issue)
2. Partner reviews the code
   - Read the diff
   - Check against acceptance criteria
   - Run locally if needed
   - Leave comments (questions, suggestions, approvals)
3. Developer addresses feedback
4. Partner approves → Merge
```

Minimum 5 PR reviews per partner across the project (visible in GitHub).

What to Look For in Review

When reviewing your partner's PR:

- **Does it match the issue?** -- Check acceptance criteria
- **Code quality** -- Naming, structure, readability
- **Tests included?** -- New feature should have tests
- **Rules file followed?** -- Does it match your shared conventions?
- **No dead code** -- Remove console.logs, commented-out code

Use AI to assist your review but don't skip reading the code yourself. AI catches patterns; humans catch intent.

What to Remember

1. **Agile isn't bureaucracy** -- it's short feedback loops to build the right thing faster.
2. **GitHub Issues + Projects = your scrumboard** -- no extra tools needed.
3. **PRD maps to issues** -- every user story becomes actionable GitHub Issues.
4. **Branch-per-issue** keeps your work organized and reviewable.
5. **Every PR gets reviewed by your partner** -- this is how professionals work.
6. **Pair workflow + AI** = scrum process, design thinking, and AI working together.

Looking Ahead

Next Week: Advanced IDE AI Features

Week 8 -- Power features for your P2 sprint

- **Agent memory** -- Persistent project knowledge via `.antigravityrules` / `CLAUDE.md`
- **MCP servers** -- Connecting your AI to external tools and data
- **Browser mode & mockup-to-code** -- AI that sees your running app
- **Debugging with AI** -- Error analysis, stack trace reading, rubber duck debugging
- **Shared rules files for pairs** -- Evolving conventions as a team

HW3 (Context Engineering) is due Week 8. Create your P2 rules file and Scrum board. Start now.

Resources

Required Reading

Resource	URL
The Scrum Guide (official)	**scrumguides.org**
GitHub Projects Documentation	**docs.github.com/issues/planning-and-tracking-with-projects**
GitHub Issues Documentation	**docs.github.com/issues**

Recommended Reading

Resource	URL
Scrum by Jeff Sutherland (course textbook)	Required book for the course
Atlassian Agile Coach	**atlassian.com/agile**
GitHub Flow Guide	**docs.github.com/get-started/using-github/github-flow**

Speaker notes