GitHub Copilot Integration in Programming Courses

## Downloads

* [Download as Word Document (DOCX)](/downloads/teaching/programming-course-github-copilot.docx)

# Case Study: Integrating GitHub Copilot in an Engineering Programming Course

## Course Context

**Course:** Introduction to Programming for Engineers (CE/ME 2114)
**Level:** Sophomore
**Enrollment:** 60 students
**Prior Format:** Traditional programming instruction with weekly labs and projects
**Programming Language:** Python for engineering applications
**Faculty:** Dr. Z, Associate Professor of Civil Engineering

## Implementation Goals

Dr. Z identified several challenges in her programming course that GitHub Copilot might address:

1. **Skill Disparity:** Wide range of prior programming experience among engineering students
2. **Debugging Bottlenecks:** Limited TA resources for helping students debug code during labs
3. **Focus on Concepts:** Students spending too much time on syntax rather than engineering concepts
4. **Real-World Alignment:** Need to prepare students for industry environments where AI coding tools are becoming standard

## Implementation Process

### Phase 1: Faculty Preparation (2 weeks before semester)

1. **Tool Exploration:**
	* Requested institutional GitHub Education access
	* Completed GitHub Copilot tutorial course
	* Practiced integrating Copilot into VS Code
	* Experimented with prompting techniques
2. **Course Material Adaptation:**
	* Reviewed all programming assignments
	* Identified which assignments would allow/restrict Copilot
	* Created “Copilot-resistant” assessment components
	* Developed Copilot-specific prompting guides
3. **Policy Development:**
	* Created clear guidelines on appropriate Copilot use
	* Developed documentation requirements for AI assistance
	* Established grading criteria that valued process understanding

### Phase 2: Student Onboarding (First two weeks of semester)

1. **Introduction Session:**
	* 50-minute workshop on GitHub Copilot setup
	* Demonstration of basic functionality
	* Discussion of benefits and limitations
2. **Guided Practice:**
	* Structured lab exercise specifically for learning Copilot
	* Paired programming with one student driving Copilot
	* Reflection on what Copilot could/couldn’t do well
3. **Expectation Setting:**
	* Clear communication about when Copilot use was permitted
	* Documentation requirements explained
	* Emphasis on understanding generated code

### Phase 3: Progressive Integration (Throughout semester)

1. **Scaffolded Approach:**
	* Early assignments: Limited Copilot use with specific sections marked for AI assistance
	* Mid-semester: Expanded use with documentation requirements
	* Late semester: Full integration with critical evaluation components
2. **Documentation Practice:**
	* Students maintained “AI logs” documenting:
		+ Prompts used with Copilot
		+ Code generated vs. code modified
		+ Understanding of how the code worked
		+ Limitations encountered
3. **Metacognitive Development:**
	* Regular reflection assignments on:
		+ When AI assistance was most/least helpful
		+ How they evaluated AI-generated code
		+ What they learned about their own coding process

## Assessment Transformation

### Before: Traditional Programming Assignment

# Original Assignment Instruction
# Write a program that processes structural beam load data from the given CSV file.
# The program should:
# 1. Calculate the maximum bending moment for each beam
# 2. Determine if the beam strength is adequate based on allowable stress
# 3. Output a summary report with recommendations for each beam

### After: AI-Integrated Assignment

# Revised Assignment Instruction
# PART A - AI-Assisted Development (60% of grade)
# Using GitHub Copilot, develop a program that processes structural beam load data.
# Your program should:
# 1. Calculate the maximum bending moment for each beam
# 2. Determine if the beam strength is adequate based on allowable stress
# 3. Output a summary report with recommendations for each beam
#
# Documentation Requirements:
# - Include your initial prompts to Copilot
# - Highlight sections written by you vs. generated by Copilot
# - Explain key engineering concepts implemented in the code
# - Describe any modifications you made to the generated code and why
#
# PART B - Critical Analysis (40% of grade)
# Without using Copilot, complete the following:
# 1. Identify potential limitations in the generated code
# 2. Explain one alternative approach to solving the problem
# 3. Describe a scenario where the current implementation might fail
# 4. Propose a specific enhancement to make the code more robust

## Resources Included

1. **Syllabus Language:** Clear policy statements about AI use
2. **Assignment Templates:** Examples of AI-integrated assignments
3. **Grading Rubrics:** Assessment criteria balancing AI use and individual understanding
4. **Student Guide:** Instructions for GitHub Copilot setup and use
5. **Reflection Prompts:** Questions to guide student metacognition

## Implementation Advice

### For Faculty Considering Similar Integration:

1. **Start Small:** Begin with one or two assignments rather than whole-course transformation
2. **Cultivate Curiosity:** Position yourself as a co-learner with students
3. **Emphasize Process:** Develop assessment approaches that value understanding over output
4. **Expect Iteration:** Be prepared to adjust policies and practices mid-semester
5. **Leverage Community:** Connect with other educators implementing similar approaches

### Technical Prerequisites:

1. GitHub Education account for faculty and students
2. Appropriate IDE setup (VS Code with GitHub Copilot extension)
3. Clear documentation templates
4. Version control system to track changes
5. Alternative access options for students without compatible devices

*This case study was developed as part of the “Strategies for Integrating Generative AI in Engineering Education” workshop materials in collaboration with Claude-3.7 Sonnet.*