AI-Assisted Coding

Week 2 - Advanced Programming

Prof. Agon Bajgora

Today’s Agenda

- Introduction: AI in Software Development
- GitHub Copilot & AI Pair Programming
- Benefits & Limitations of AI Coding Tools
- Research on AI Coding Productivity
- Hands-on: Using GitHub Copilot
- Coding Exercise: AI-assisted Implementation
- Refactoring with AI

What is AI-Assisted Coding?

- AI coding assistants act as "pair programmers"
- Suggest code snippets, functions, and methods
- Autocomplete based on natural language comments
- Help debug and explain code
- Tools include GitHub Copilot, ChatGPT, Tabnine, Amazon CodeWhisperer

Traditional Coding:

- Manual code writing
- Documentation lookup
- StackOverflow searches

AI-Assisted Coding:

- Contextual suggestions
- Natural language to code
- In-editor intelligence

How AI Coding Assistants Work

- Trained on billions of lines of public code repositories
- Use large language models (LLMs) to understand patterns
- Analyze your code, comments, and context
- Predict what code you’re likely to write next
- Improve over time through feedback

Important: AI coding assistants don't "understand" code the way humans do - they're making statistical predictions about what code patterns should follow your current code.

GitHub Copilot

- Developed by GitHub and OpenAI
- Built on OpenAI Codex (GPT model fine-tuned on code)
- Integrates directly into code editors like VS Code
- Suggests entire functions and code blocks
- Works across dozens of programming languages

Example Functionality:

```
# Function to calculate the mean of
# a list of numbers
def calculate_mean(numbers):
    # Copilot might suggest:
    total = sum(numbers)
    return total / len(numbers)
```

Copilot can generate entire function bodies based on just a comment!

Research: Impact of AI Coding Assistants

Based on GitHub & Accenture research on Copilot:

Productivity Boost

- 55% of developers completed tasks faster with Copilot
- 88% report feeling more productive
- 96% report faster completion of repetitive tasks

Developer Experience

- 77% spend less time searching for information
- 87% maintain flow state longer
- 73% report less mental fatigue

Source: Research by GitHub & Accenture, 2023 - "Quantifying GitHub Copilot's impact in the enterprise"

AI Coding Assistants: Benefits

Speed & Efficiency

- Write code faster
- Reduce boilerplate work
- Automate repetitive tasks

Learning & Discovery

- Explore new patterns
- Learn libraries & frameworks
- See alternative approaches

Reduced Context Switching

- Less documentation lookup
- Fewer web searches
- Stay in your IDE

Accessibility

- Lowers barriers to coding
- Help with syntax recall
- Reduces typing strain

Focus on Logic

- Focus on problems, not syntax
- Think at a higher level
- More time for architecture

Experimentation

- Try ideas quickly
- Rapid prototyping
- Explore alternatives

AI Coding Assistants: Limitations

Quality Concerns

- Can generate incorrect code
- May introduce subtle bugs
- Sometimes produces inefficient solutions
- May suggest outdated approaches

Understanding Gaps

- No true understanding of business logic
- Can't replace domain knowledge
- Missing context beyond the codebase
- No awareness of project goals

Security Risks

- Potential to introduce vulnerabilities
- Might use insecure patterns
- Could expose sensitive information

Overreliance Danger

- Skill atrophy concerns
- Critical thinking still required
- May accept solutions without verification

Effective Prompting for AI Coding Tools

Best Practices:

- Be specific and detailed in comments
- Include expected inputs and outputs
- Specify language/framework preferences
- Break complex problems into smaller steps
- Provide context about your application

Remember: The quality of suggestions directly correlates with the quality of your prompts!

Example - Poor Prompt:

```
# Sort function
```

Example - Better Prompt:

```
# Function to sort a list of student
objects by GPA (descending) # Each
student has properties: name (string),
id (int), and gpa (float) # Should
handle empty lists and return a new
sorted list
```

Code Refactoring with AI Tools

- AI can help improve existing code
- Identify patterns for optimization
- Suggest cleaner, more maintainable alternatives
- Apply best practices and design patterns

Before Refactoring:

```
def process(d):
  res = []
  for i in range(len(d)):
    if d[i] > 0:
      res.append(d[i] * 2)
    else:
      res.append(0)
  return res
```

After AI-Suggested Refactoring:

```
def process_numbers(data):
  """Double positive numbers, replace negatives with ze
  return [num * 2 if num > 0 else 0 for num in data]
```

Hands-On Exercise: GitHub Copilot

Let’s explore AI pair programming together!

Exercise: Fibonacci Implementation with AI

Task:

1. Open VS Code with GitHub Copilot enabled
2. Create a new file: `fibonacci.py`
3. Write a detailed comment for a Fibonacci function
4. Let Copilot suggest an implementation
5. Test the function with different inputs
6. Try to improve the implementation with Copilot

Extension: How would you optimize this for large numbers? Ask Copilot to help!

Example Comment:

```
# Function to calculate the nth Fibonacci number # The Fibonacci sequence starts with 0 and 1 # Each subsequent number is the sum of the two preceding ones # Example: 0, 1, 1, 2, 3, 5, 8, 13, ... # Parameter: n (int) - the position in the sequence (0-indexed) # Returns: the nth Fibonacci number
```

Evaluation Points:

- Correctness of implementation
- Code readability and documentation
- Handling of edge cases

Exercise: Refactoring with AI

Task:

1. Take the provided code snippet
2. Identify issues with readability, efficiency, or style
3. Use AI assistance to suggest improvements
4. Apply and evaluate the refactored code
5. Compare before and after

Evaluation Points:

- Code readability improvement
- Better function and variable names
- Improved algorithm efficiency
- Added appropriate error handling
- Better documentation

Code to Refactor:

```
def f(l, t):
    r = []
    for i in range(len(l)):
        if l[i]["t"] == t:
            r.append(l[i])
    return r

# Usage example:
items = [
    {"id": 1, "t": "book", "price": 20},
    {"id": 2, "t": "food", "price": 10},
    {"id": 3, "t": "book", "price": 15},
    {"id": 4, "t": "food", "price": 5}
]
books = f(items, "book")
```

Questions?

Let’s discuss AI-assisted coding!

Homework Assignment

AI-Assisted Development Portfolio Task:

1. Choose one algorithm from the provided list
2. Implement it with GitHub Copilot or another AI assistant
3. Document your process - include:
   - The prompts/comments you used
   - Screenshots of AI suggestions
   - Your modifications to the AI code
   - Testing strategy and results
4. Submit your code and documentation to GitHub
5. Be prepared to discuss your experience in the next class

Due Date: Before our next session

Next Class Preview

Session 2 Topics:

- Applied AI pair programming
- Working on mini-project
- Code review best practices
- Debug and refine code with AI

Preparation:

- Complete the Fibonacci exercise
- Start on your homework assignment
- Bring questions about your AI coding experience
- Make sure GitHub Copilot is working in your environment

Remember: AI is a tool to enhance your programming skills, not replace them. The goal is to become a more effective developer by leveraging AI capabilities.

Thank You!

Contact Information

Professor: Agon Bajgora

Email: agon.bajgora@umib.net

Office Hours: Mondays 10:00-11:00 (K1)

Course Materials

All slides and code examples available on the course GitHub repository