Software Engineering 2025–26

Lecture 6: Software Engineering in the AI Era

Disclaimer: AI is moving very fast and this lecture is probably already out of date!

Prof. Robert Harle

Department of Computer Science and Technology
University of Cambridge

Easter Term

The "State of the Art" Disclaimer

WARNING: HIGH VELOCITY AREA

- AI in Software Engineering is moving at a pace unprecedented in CS history.
- Breakthroughs occur weekly (new models, new agentic frameworks).
- What we call "best practice" today might be an anti-pattern by Michaelmas.
- **Goal:** Focus on the enduring engineering principles, not just the specific buttons to click.

Part 1: AI as a Coder

From Autocomplete to Co-pilot

Autocomplete vs. Semantic Understanding

Beyond Lexical Completion

- **Traditional Autocomplete (IntelliSense):** Uses Language Server Protocols (LSP). It knows that `user.` should be followed by a member of the `User` class.
- **Semantic Understanding (LLMs):** Understands the *intent* across multiple files.
- It doesn't just complete the word; it completes the **logic**.
- **Example:** If you write a comment `// Calculate the Haversine distance`, the AI understands the mathematical intent, not just the syntax.

Semantic Awareness in Action

Context is King

- Modern AI tools use "Project Context" (RAG for code).
- They "read" your `package.json`, your existing utility functions, and your naming conventions.
- **Result:** It doesn't suggest a generic sorting algorithm; it suggests one that uses your specific `Logger` and `ErrorHandler` classes.

Handling Boilerplate Code

Eliminating the "Plumbing"

- 40-60% of professional code is "boilerplate":
  - Setting up Express/FastAPI routes.
  - Database schemas and migrations.
  - Data transfer objects (DTOs) and Mappers.
  - Unit test setup code.
- AI reduces the "Time to Hello World" from hours to seconds.

Building Full Prototypes

Prompt to MVP

- **Scenario:** "Build a React frontend with a Python backend that stores user notes in PostgreSQL."
- AI can generate the directory structure, the Dockerfile, the DB schema, and the API endpoints in one pass.
- **Value:** Rapid hypothesis testing. You can see if a product idea works before committing a sprint to it.
- *Warning:* Prototypes are often "happy path" only.

Language Polyglot: The Universal Translator

Changing Between Languages

- AI handles the "Syntactic Tax" of switching languages.
- **Example:** "Convert this Java Stream logic into idiomatic Rust using iterators."
- **Why switch?**
  - **Performance:** Python $\rightarrow$ C++/Rust for bottlenecks.
  - **Safety:** C $\rightarrow$ Rust for memory safety.
  - **Frontend:** Backend logic $\rightarrow$ TypeScript for client-side execution.

AI in Testing: Unit Test Generation

The End of the "Lazy Tester"

- Engineers notoriously hate writing unit tests.
- AI can analyze a function and generate a comprehensive suite:
  - Happy path.
  - Edge cases (nulls, empty strings, max integers).
  - Mocking external dependencies (DBs, APIs).
- *Catch:* It tests what the code *does*, not necessarily what it *should* do.

AI in Testing: Intelligent Fuzzing

Breaking the Unbreakable

- **Fuzzing:** Providing random data to find crashes.
- **AI Fuzzing:** Generating *plausible but malformed* data.
- **Example:** Instead of random bytes, the AI generates a JSON payload with a circular reference or an unexpectedly large nested array designed to trigger a stack overflow.

AI in the CI/CD Pipeline

The Self-Healing Build

- **Fail → Analyze → Fix.**
- If a test fails in GitHub Actions, an AI agent can:
  1. Read the error log and stack trace.
  2. Identify the failing line.
  3. Propose a fix in a new commit.
- *Role:* The "Night Watchman" for your codebase.

Debugging: Root Cause Analysis

Finding the "Why"

- **Old way:** Adding `print()` statements or stepping through a debugger for 3 hours.
- **AI way:** Pasting the stack trace and state variables.
- "The crash is caused because Service A expects an ISO-8601 date, but Service B is sending a Unix timestamp."
- AI excels at correlating disparate pieces of information.

Debugging: Log Analysis at Scale

The Needle in the Haystack

- Enterprise systems generate terabytes of logs.
- AI can perform **Anomaly Detection**: "Usually, the login endpoint takes 50ms. In the last hour, 5% of requests took 2000ms. Here are the 3 logs representing that delay."
- Summarizing unstructured text into actionable insights.

Refactoring: Paying Down Debt

Cleaning the Kitchen

- "This function has a cyclomatic complexity of 15. Refactor it into smaller, testable units."
- "Rename all variables in this file to follow the Google Style Guide."
- "Extract the hardcoded configuration into an environment variable handler."

Understanding a Large Codebase

The "Onboarding" Superpower

- New job? 10 million lines of code?
- **Semantic Search:** "Show me every place where we calculate VAT, and explain the differences between the implementations."
- AI maps the mental model of the codebase for you.

Rubber Duckie Reinvented

Pair Programming 2.0

- The "Rubber Duck" method involves explaining your code to a toy to find bugs.
- **AI Rubber Duck:** The duck now talks back.
- "I'm thinking of using a Bloom Filter here to save memory. Does that make sense given our false positive tolerance?"
- It provides a sounding board for architectural brainstorming.

The Risks of AI Coding

Trust, but Verify (Aggressively)

Risk: Not All Models are Equal

Size and Fine-tuning Matter

- A small, local model (Llama-3-8B) might be fast but prone to logic errors.
- A massive model (GPT-4o, Claude 3.5) has better reasoning but higher latency/cost.
- **The Trap:** Assuming that because the AI was right 10 times, it will be right the 11th time.

Risk: The Technical Debt of Ignorance

“It works, but I don’t know why”

- If you submit code you don’t understand, you cannot maintain it.
- AI often uses clever (or “hacky”) tricks that pass tests but violate long-term architectural goals.
- **Rule:** You are the owner of every line you commit.

Risk: Hallucinations

Confident Incorrectness

- AI will invent libraries (`import magic_security_fix`) or API methods that don't exist.
- It will confidently suggest insecure patterns if they were common in its training data (e.g., old versions of libraries with known CVEs).

Mitigation: LLMs to check LLMs

The "Critic" Architecture

- **Generator Model:** Writes the code.
- **Verifier Model:** Tries to find bugs, security flaws, or style violations in the generated code.
- This "Adversarial" approach significantly reduces hallucinations.

Risk: Algorithmic Bias

The Internet's Mirror

- AI reflects the biases of its training data (GitHub, StackOverflow).
- This can include biased logic in social-technical systems (e.g., loan approval logic) or simply a bias toward "popular" but "sub-optimal" solutions.

Part 2: AI throughout the Workflow

Beyond the Code Editor

AI in the Design Process

Brainstorming the Product

- "What features are missing from current task-management apps for students?"
- "Identify the potential 'Jobs to be Done' for this user persona."
- AI acts as a creative partner for Product Management.

Brainstorming System Design

Architectural Trade-offs

- "Create a trade-off table between using AWS Lambda vs. Kubernetes for a periodic data scraping service."
- "How should we handle eventual consistency in our proposed microservices architecture?"
- AI can generate Mermaid.js or PlantUML diagrams from descriptions.

Team Co-ordination

Cutting through the Noise

- **Focus Management:** AI summaries of long Slack threads or Jira comments.
- **Distraction Minimization:** "Summarize my mentions from the last 4 hours and highlight anything requiring a code change."
- AI agents can act as "Scrum Assistants," identifying blocked tasks based on chat sentiment.

Automated Code Review

Semantic Understanding > Style Checking

- **Linter:** "You missed a semicolon."
- **AI Reviewer:** "This function is called in a loop, but it performs a synchronous DB query. This will cause a performance bottleneck as the user base grows."
- AI catches **architectural violations**.

Documentation Generation

Living Docs

- Auto-generating docstrings, READMEs, and API references (Swagger).
- **The Best Part:** Keeping them in sync. If the code changes, the AI flags the documentation as out of date or updates it automatically.

The Rise of AI Agents

From Chatting to Doing

- **Tool Use:** Models can now call functions, run shell commands, and read files.
- **Agency:** The ability to break a complex goal ("Fix this bug") into multiple steps and execute them autonomously.

Terminal Integration: Gemini CLI

Command-Line Superpowers

- **Example:** `gemini ‘‘find all unused exports in this project and delete them’’`
- **Value:** Removing the friction of manual file navigation.
- The terminal becomes a natural language interface to the file system.

Claude Code & GitHub Copilot Workspace

End-to-End Task Completion

- "Implement a new 'Dark Mode' across the entire application."
- The agent:
  1. Identifies all CSS/Theme files.
  2. Modifies them.
  3. Updates the toggle component.
  4. Runs the tests to ensure no regressions.

Part 3: Building FOR AI

Architecting the Future

Building FOR vs. WITH AI

A Fundamental Shift

- **Building WITH AI:** Using Copilot to write a Java app. The final binary contains no AI logic.
- **Building FOR AI:** Building an app where an LLM is a core component of the runtime (e.g., an automated legal assistant).

Determinism vs. Probabilistic

The Reliability Paradox

- **Traditional Code:** $f(x) \rightarrow y$ (Always).
- **AI Code:** $f(x) \rightarrow y$ (Probably).
- How do you build a reliable system on top of a "black box" that might behave differently tomorrow?

Why AI is so good for coding...

The Feedback Loop of Truth

- Code has a **Ground Truth**: Does it compile? Does it pass tests?
- AI can learn from its own mistakes by trying to run the code it just wrote.
- This is why coding AI is advancing faster than creative writing AI.

The Challenge of Subjective AI

The Virtual Doctor Problem

- There is no “compiler” for medical advice or legal opinions.
- **Confidence and Safety:**
  - Building multi-layered “Guardrail” systems.
  - **Evals:** Massive datasets of “correct” answers graded by human experts.
  - Using LLMs to check LLMs for factual consistency.

Architectural Challenges: Latency

Speed is a Feature

- A database query takes 10ms.
- An LLM generation takes 2000ms - 10,000ms.
- **UX Impact:** We must move to **Streaming Architectures** (showing text as it's generated) and asynchronous patterns.

Architectural Challenges: Cost and Scalability

Tokens are Money

- Traditional APIs are “free” to call once hosted.
- AI APIs charge per word (token).
- A viral app can go bankrupt in a week if not architected with cost-caching and rate-limiting.

Provider Lock-in?

The New Monopoly

- If you build around OpenAI's specific "GPT-4o Vision" API, how hard is it to switch to Google's Gemini?
- **Abstraction Layers:** Software engineering best practice suggests building wrappers around AI providers to remain "Model Agnostic."

Part 4: The New Engineer

Survival and Growth

The Junior Developer Gap

The Training Ladder is Breaking

- Junior tasks (fixing small bugs, writing boilerplate) are exactly what AI is best at.
- If we don't hire juniors to do "easy" work, how do they become senior architects?
- **New Junior Skill:** Orchestrating AI to do 10x more work, while maintaining high quality.

Jevons Paradox

Efficiency $\rightarrow$ Demand

- **The Paradox:** Making something more efficient usually *increases* its total consumption.
- Making software 10x cheaper to build won't mean we need 1/10th the engineers; it means companies will want 100x more software.

The Shifting Skillset

Domain Knowledge > Raw Syntax

- Knowing where the semicolon goes is a commodity.
- **What is valuable?**
  - **Domain Knowledge:** Understanding the business/science problem.
  - **System Design:** How the parts fit together.
  - **Human Communication:** Defining the *right* thing to build.

Lifelong Learning

Adaptability is the only Security

- In the AI era, a tech stack's lifespan is 2-3 years.
- You are not a "React Developer" or a "Python Developer."
- You are a **Problem Solver** who uses the best tools available.

Course Review: The Core Principles

AI changes the TOOLS, not the GOALS

- **Reliability:** Is the code tested and robust?
- **Maintainability:** Can another human (or AI) understand it in 2 years?
- **Security:** Is the user's data protected?
- **Value:** Does it actually solve a real human problem?

"AI will not replace software engineers.
Engineers who use AI will replace engineers who don't."