CS2104 Problem Solving in Computer Science

Margaret Ellis,
Naren Ramakrishnan,
Sehrish Basir Nizamani

Introduction to Software Engineering

Software Engineering Learning Objectives

- explore concepts in software engineering
- recognize stages of software development
- construct software design given system requirements
- explore concepts in software engineering quality control
- recognize how LLMs can and cannot reliably assist in the software engineering process

Introduction to Software Engineering …

- Metaphors and Analogies
  - Multiple meanings: viruses, Trojan horses, worms, bugs, bombs, crashes, flames, and fatal errors
  - “A metaphor serves more as a heuristic than it does as an algorithm.”
  - “Because each program is conceptually unique, it's difficult or impossible to create a general set of directions that lead to a solution in every case. Thus, knowing how to approach problems in general is at least as valuable as knowing specific solutions for specific problems.”
  - Accretion, incremental development

Introduction to Software Engineering cont.

Building Software

- Building a dog house vs. a human house vs. a custom house
- Cost of labor vs cost of material
- Structural changes vs. independent changes
- Multiple meanings: architecture, scaffolding, construction, foundation classes, and tearing code apart

Adapted from Code Complete

Topics in Software Engineering

- Requirements Gathering
- Project Management
- Designing
- Programming
- Repository - version control
- Testing
- Security
- Debugging - stepping through
- Refactoring
- DevOps
- Maintenance

Topics in Software Engineering

Requirements Gathering
Project Management
Designing
Programming
Repository - version control
Testing
Security
Debugging - stepping through
Refactoring
DevOps
Maintenance

Requirements Gathering

Defining the Problem, Determining the Solution

- Iterative – Ask Many Questions!
  - What does the end result look like?
  - How does the end result work?
- Business Rules
  - Ask the right people the right questions
- Project Managers
- Testing
- Features?!
- SMART - specific, measurable, agreed upon, realistic and time-based

Searching for Requirements

- **How** will you use this feature?
- **How** might we meet this business need?
- **Where** does the process start?
- **Where** would the user be located physically when using this feature?
- **When** will this feature fail?
- **Who** will receive the outputs of the feature?
- **What** does this feature need to do?
- **What** are the pieces of this feature?
- **What** if…? Think of all the alternative scenarios and ask questions about what should happen if those scenarios are true
- **What** needs to be tracked?

https://www.bridging-the-gap.com/what-questions-do-i-ask-during-requirements-elicitation/

Use of LLMs in Requirements Gathering

- **Natural Language Processing (NLP) for Requirements**: LLMs can analyze and extract requirements from unstructured data such as emails, meeting notes, and user feedback. LLMs can help in translating natural language requirements into technical specifications and design documents.
- **Interview Structuring:** They can help structure interviews and surveys to gather requirements more effectively
- **Drafting Requirements Documents:** LLMs can generate initial drafts of Software Requirements Specifications (SRS) based on input from stakeholders
- LLMs can be utilized to **assess various characteristics of high-quality requirements**, such as:
  a. Unambiguity;
  b. Consistency;
  c. Traceability;
  d. Feasibility;
  e. Verifiability.

Generated with assistance of co-pilot Dec 2024 and resources:https://fitech101.aalto.fi/courses/software-engineering-with-large-language-models/part-4/2-requirements-gathering,https://arxiv.org/pdf/2404.17842,https://insights.sei.cmu.edu/blog/application-of-large-language-models-llms-in-software-engineering-overblown-hype-or-disruptive-change/,
https://www.techopedia.com/5-ways-llms-can-empower-software-engineering

Example: LLM helping with Requirements Gathering

Prompt: Provide an example of an LLM structuring and interview to gather requirements for a food delivery app

See: Co-Pilot Generated Requirements Interview Script for Food Delivery App

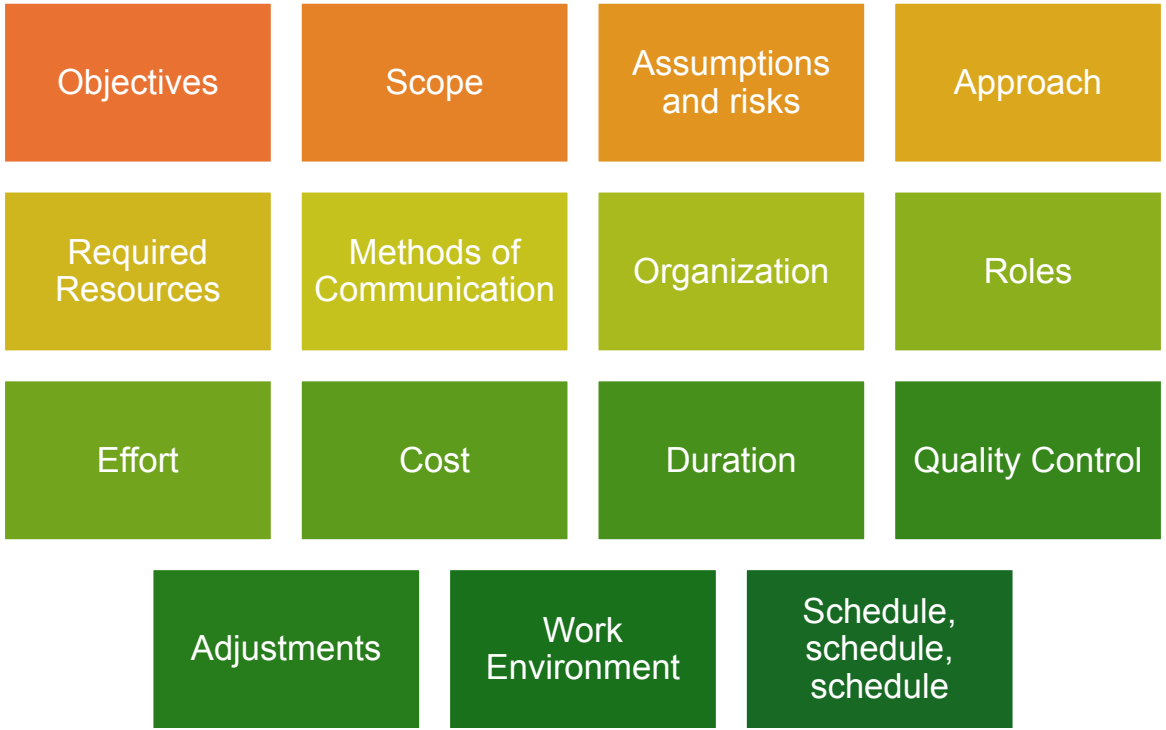
Topics in Software Engineering

Requirements Gathering
Project Management
Designing
Programming
Repository - version control
Testing
Security
Debugging - stepping through
Refactoring
DevOps
Maintenance

Project Management

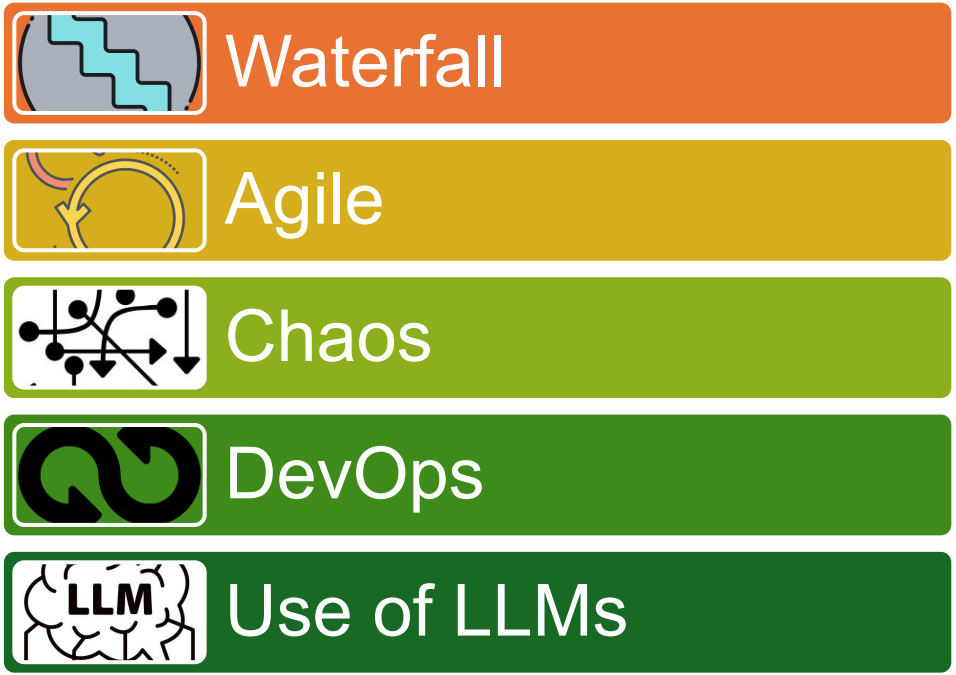
Adapted from 2011-12 Shaffer & McQuain CS 2104

Project Management Considerations



http://www.techrepublic.com/blog/10-things/10-best-practices-for-successful-project-management/

Project Management Strategies



http://www.techrepublic.com/blog/10-things/10-best-practices-for-successful-project-management/

Agile Problem Solving Approach (Agile Manifesto)

- Welcome change
- Small cycles, regular feedback
- Reflect, recap what you know
- Ask questions
- Keep it simple
- Iterate

Scheduling

- Managing large-scale projects involves significant efforts to plan and schedule activities
  - It is human nature to work better toward intermediate milestones.
- The same concepts can/should be applied to mid-sized projects encountered in class.
  - For any project that needs more than a week of active work to complete, break into parts and design a schedule with milestones and deliverables.
    - Track your progress
    - Set goals
    - Seek feedback

Use of LLMs in Project Management

- **Task Automation:** LLMs can automate routine tasks such as scheduling meetings, summarizing project updates, and generating reports
- **Risk Management:** They can analyze historical data to predict potential risks and suggest mitigation strategies
- **Communication:** LLMs can facilitate clearer communication by summarizing complex documents and translating languages.

Generated with assistance of co-pilot Dec 2024 and resources: https://pmsymposium.umd.edu/pm2024/session/demystifying-llms-polaski-and-beall/,https://integranxt.com/blog/large-language-models-project-management/

Example of Risk Assessment

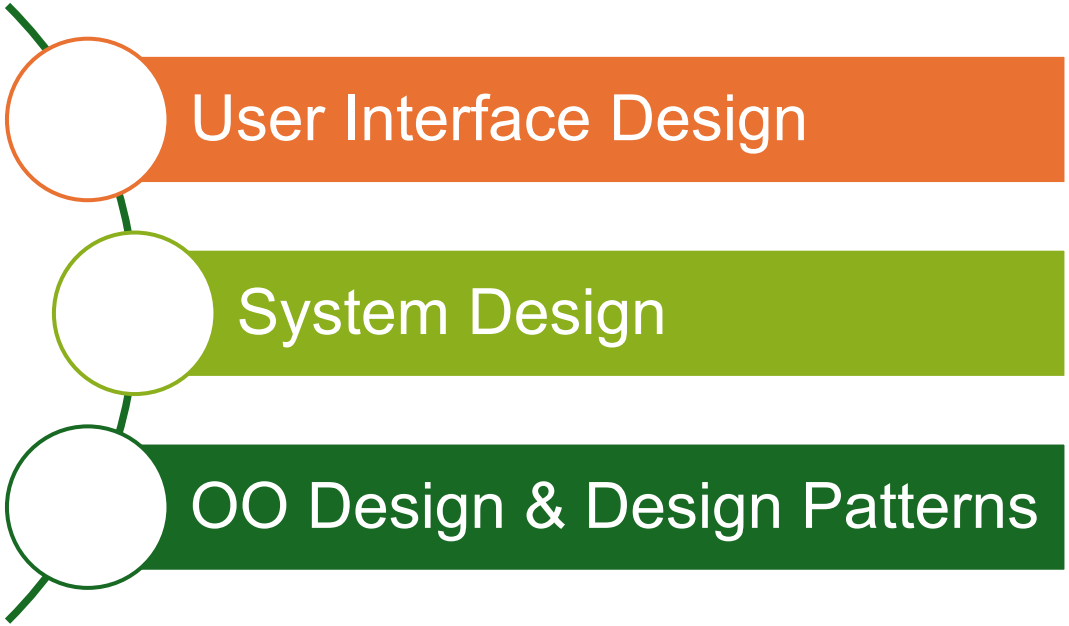
Example Case Study: A company used an LLM to analyze historical project data from their project management system. The LLM identified that projects with frequent scope changes and high team turnover were more likely to experience significant delays. Based on these insights, the company implemented stricter change management processes and focused on improving team stability, which led to a reduction in project delays.

https://dev.to/delamorareseach/project-management-and-large-language-models-llm-a-synergistic-future-3ei

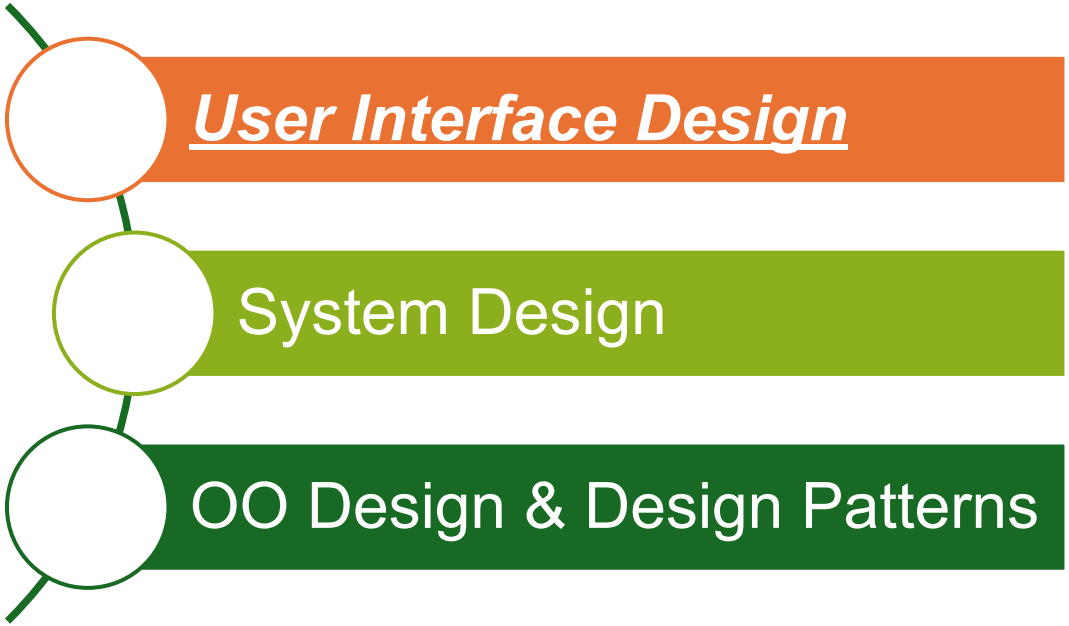
Topics in Software Engineering

Requirements Gathering
Project Management
Designing
Programming
Repository - version control
Testing
Security
Debugging - stepping through
Refactoring
DevOps
Maintenance

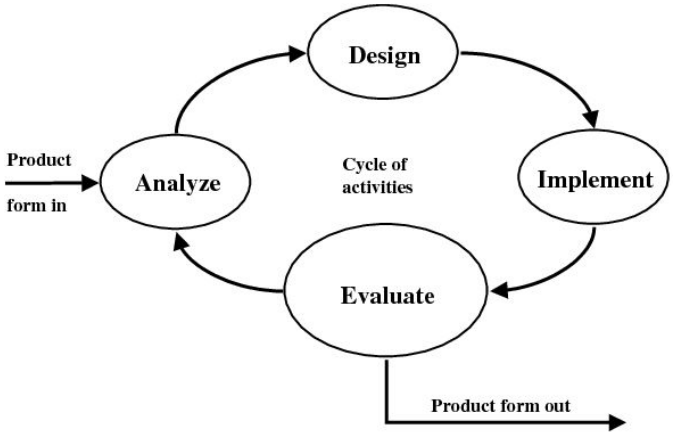
Design



Design

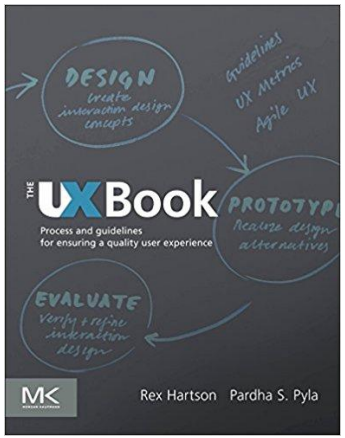


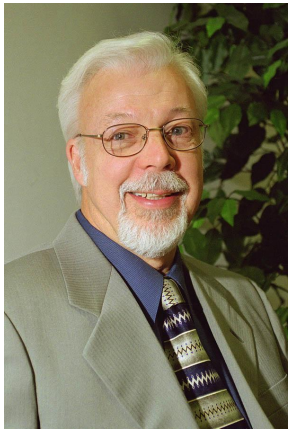
User Interface Design



User Interface Design

- HCI
- Usability and Accessibility
- Prototyping
- Use Cases





Requirements Discovery & Verification:
Efficiency techniques and strategies

Cross business representation

No devices

Keep the team small

Set solo work

Set inescapable deadlines

A bit of competition

Independent decision making

Short, energetic bursts

Divide and swarm

Prototype – answer questions by producing prototypes (not PowerPoint presentations)

Finish with a test – put your final prototype in front of real customers

Design Sprint Video

Chatbot Boom

- What are examples of chatbots you use?
- Chatbots are everywhere
  - Universities
  - Airlines
  - Online shopping
- Chatbots can be built based on a specific content area
- Querying a LLM can be inefficient and costly
  - ***Uses 10x more power than a websearch!***

https://www.msn.com/en-us/money/other/the-cost-of-an-ai-query-chatbot-boom-linked-to-massive-energy-drain/ar-AA1tubfL

Deciding whether to include GenAI in a product?

[Design Sprint for Adding GenAI]

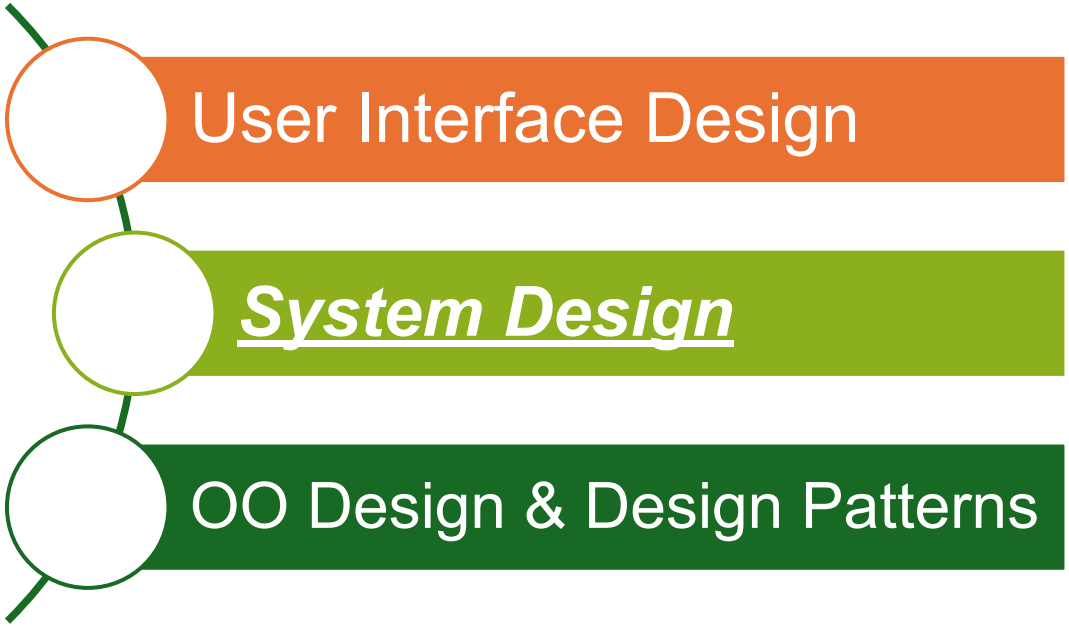
1. Problem Definition with content experts
2. Ideation
3. Rapid Prototyping
4. Testing with users

GenAI Strengths	GenAI Weaknesses
• **Generating creative text:** generating different creative text formats, including such as documents, code. • **Translation:** translating languages with impressive accuracy, often outperforming traditional rule-based systems. • **Summarization:** condense large amounts of information into concise summaries, highlighting the key points. • **Question-answering:** retrieve information from vast data sources and provide answers in a conversational manner. • **Content Creation:** create engaging marketing copy, product descriptions, social media posts.	• **Factual inconsi ncy:** confidently present incorrect inform n as facts. It's crucial to always verify th ut, especially with sensitive topics • **Bias:** trained rld data, GenAI will mirror societa esent in the data. This can lead to di or harmful outputs. • **Lack** real- nua • **Acti** any reli • **Hall** gaps in they don't know something.

Caution: LLMs use a lot of power

© Neil Shah, 2024

Design



System Design

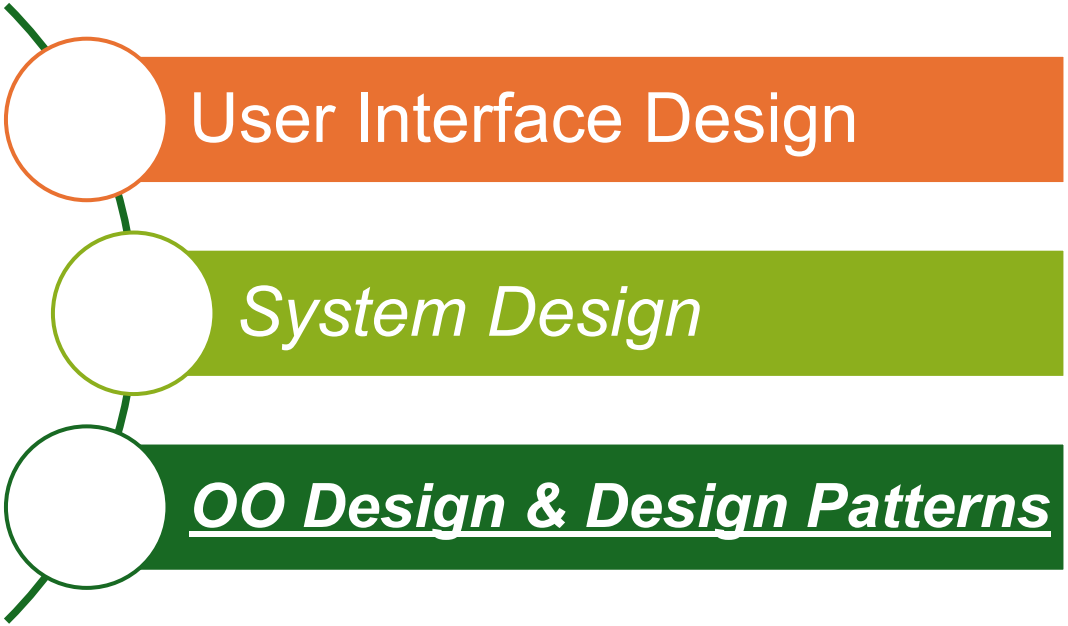
Strategies

- [ ] Spreadsheets
- [ ] Workflow Diagram
- [ ] Algorithm Flowcharts
- [ ] Algorithm Pseudocode

Considerations

- [ ] System Storage
- [ ] Time Constraints
- [ ] Telecommunication
- [ ] Users
- [ ] Platforms
- [ ] Type of data
- [ ] Environment
- [ ] Security
- [ ] Toolbox
- [ ] Budget!

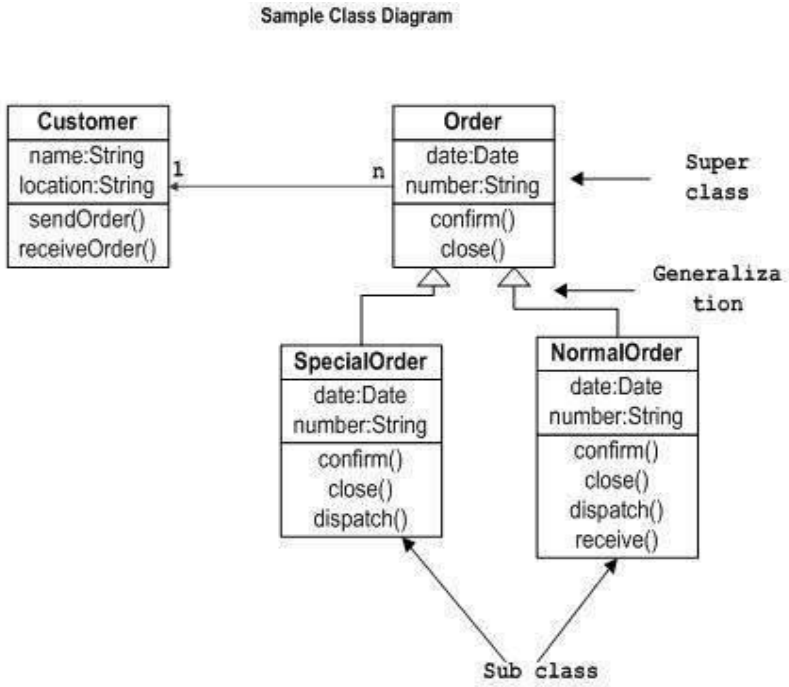
Design



Object Oriented Design

Use UML diagrams to represent this information

- What are your classes?
- What tasks are they responsible for?
- What information do they contain?
- What kind of interaction do they have with other classes?
- Are there hierarchies? Nested classes?
- Code reuse?



Object Oriented Design

GOALS:

- Manage Complexity
- Ease of Maintenance
- Reuse
- Portability
- Extensibility

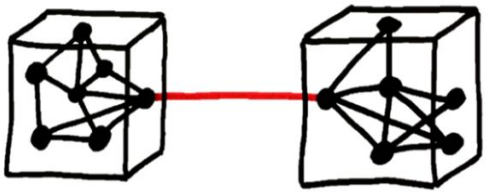
When I am working on a problem I never think about beauty. I think only how to solve the problem. But when I have finished, if the solution is not beautiful, I know it is wrong.

— R. Buckminster Fuller

https://www.tutorialspoint.com/uml/uml_class_diagram.htm

Design Considerations for Maintainability and Reuse

- **Loosely Coupled** – want to be able to make changes easily, replace one class with another
- **Encapsulated** – protect the class' data and functionality, information only available through accessors(getters) and mutators(setters)
- **Highly Cohesive** – a class should be a cohesive unit, only contain relevant data, anything extraneous should be in its own class



https://www.planetgeek.ch/2011/07/08/presentation-agile-code-design-how-to-keep-your-code-flexible/

How LLMS can aid in Design

- **Generative Design:** LLMs can convert text-based prompts into design specifications and generate multiple design variations
- **Design Optimization:** They can evaluate and optimize designs based on performance criteria
- **Documentation:** LLMs can generate detailed design documentation, making it easier to understand and maintain design decisions
- **Technology Considerations:**
  a. LLMs can provide valuable assistance to software development teams in selecting implementation tools and frameworks.
  b. Based on the system specifications and design models, LLMs can recommend appropriate programming languages, libraries, and implementation frameworks

Generated with assistance of co-pilot Dec 2024 and resources:https://hdsr.mitpress.mit.edu/pub/15nqmdzl/release/2,https://blog.burnsmcd.com/revolutionizing-engineering-design-unleashing-the-power-of-large-language-models,
https://www.techopedia.com/5-ways-llms-can-empower-software-engineering

Topics in Software Engineering

Requirements Gathering
Project Management
Designing
Programming
Repository - version control
Testing
Security
Debugging - stepping through
Refactoring
DevOps
Maintenance

Programming

- Unit test as you go
- Reuse code
- Readability
- Less is more
- Documentation
- Algorithms
- Efficiency

How LLMs can assist in programming

- **Code Generation and Completion**: LLMs can assist in writing code by generating boilerplate code, suggesting code completions, and even writing entire functions based on natural language descriptions.
- **Code Review and Quality Assurance**: LLMs can help in reviewing code by identifying potential bugs, suggesting improvements, and ensuring adherence to coding standards.
- **Documentation and Commenting**: LLMs can generate documentation and comments for code, making it easier for developers to understand and maintain the codebase.

Generated with assistance of co-pilot Dec 2024 and resources: https://www.techradar.com/computing/artificial-intelligence/best-large-language-mo https://www.techopedia.com/5-ways-llms-can-empower-software-engineering, https://nomadicsoft.io/choosing-the-best-llm-for-coding-a-comprehensive-guide-to-top-llms-for-code-generation/, https://www.scribbledata.io/blog/the-top-llms-for-code-generation-2024-edition/, https://insights.sei.cmu.edu/blog/application-of-large-language-models-llms-in-software-engineering-overblown-hype-or-disruptive-change/

Suggestions when using LLMs for Coding

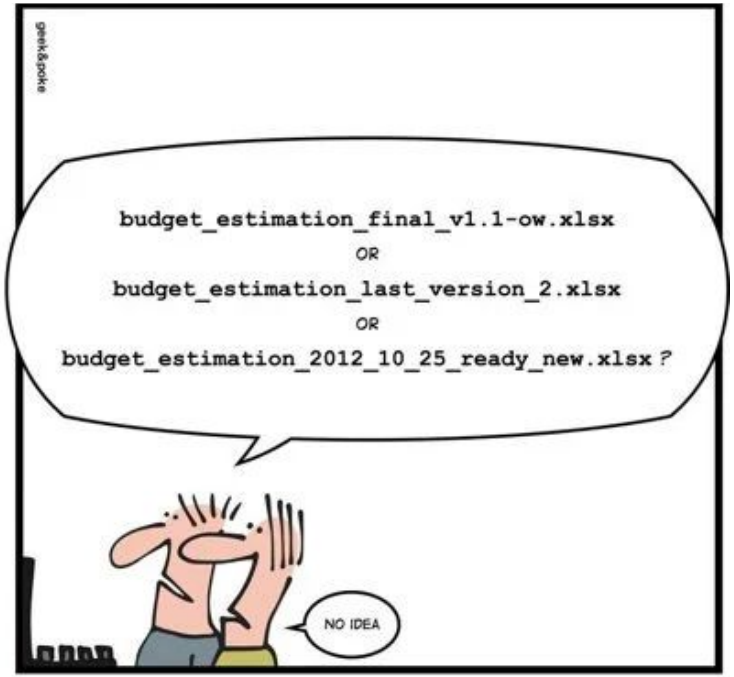
Continue to use incremental test-driven development

- Instead of overwhelming the LLM, break the problem into small pieces
  - Go step-by-step
- Give a skeleton for the LLM to work with
  - Test the subsolution at each step
  - Give very specific guidance
  - Tell the LLM exactly where to write/update code
  - Tell the LLM what code not to change
- Give very specific guidance to the LLM
  - Try to fix only one bug at a time (LLMs do not retain long context)

Topics in Software Engineering

Requirements Gathering
Project Management
Designing
Programming
Repository - version control
Testing
Security
Debugging - stepping through
Refactoring
DevOps
Maintenance

Simply Explained

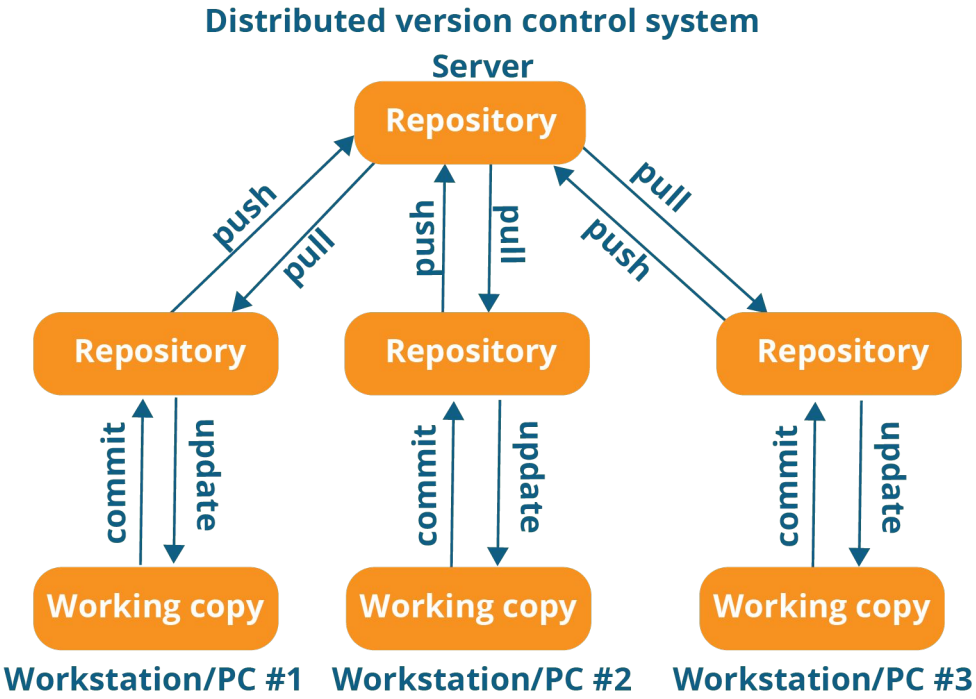


Why Version Control?

- Have you ever ….
  - Named things like my-project-current or my-project-UI_almostDone because you reached a milestone, and want to be able to get back to the current version if you screw things up later?
  - Have you ever wished that you *had* done that when you didn't?
  - Have you ever been working with other people, and had to send files back and forth, figure out how to merge each other's changes, figure out who had the best version, etc. when collaborating?
- A version control system such as GIT stores different versions of your files over time, allowing you to
  - go back to older revisions
  - see how the code developed over time
  - facilitate collaboration between people

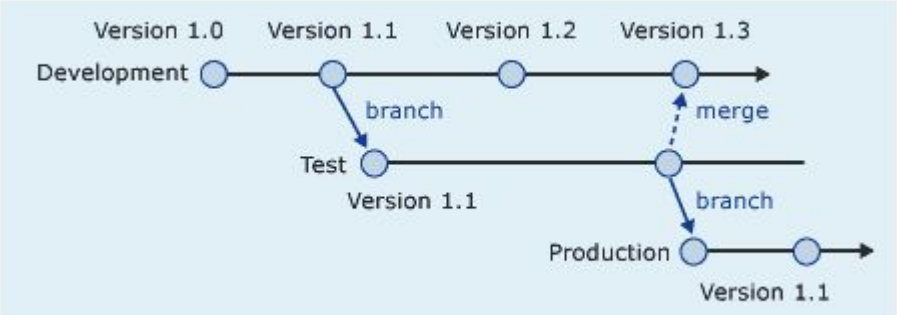
http://pages.cs.wisc.edu/~driscoll/software/vcs/

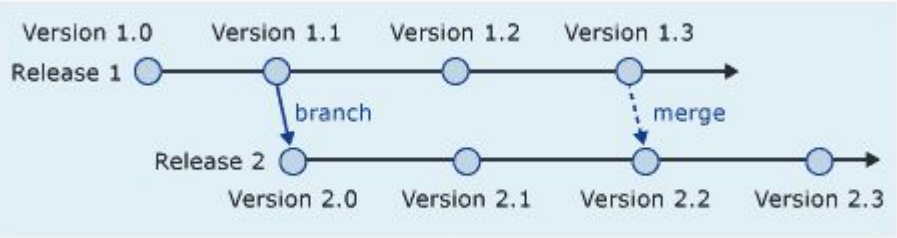
Distributed Version Control



https://www.edureka.co/blog/what-is-git/

Version Control Branching - Industry





http://blog.codinghorror.com/software-branching-and-parallel-universes/

Version Control

We will work more with git next week!

Topics in Software Engineering

Requirements Gathering
Project Management
Designing
Programming
Repository - version control
Testing
Security
Debugging - stepping through
Refactoring
DevOps
Maintenance

Highlights of Software Testing

- Unit Testing utilizes module testing techniques (white-box / black- box techniques).
- Integration Testing involves checking subsets of the system.
- Acceptance, Function and System testing is performed upon the entire system.
- Regression Testing involves fixing errors during testing and the re-execution of all previous passed tests.

How can LLMs assist with Testing

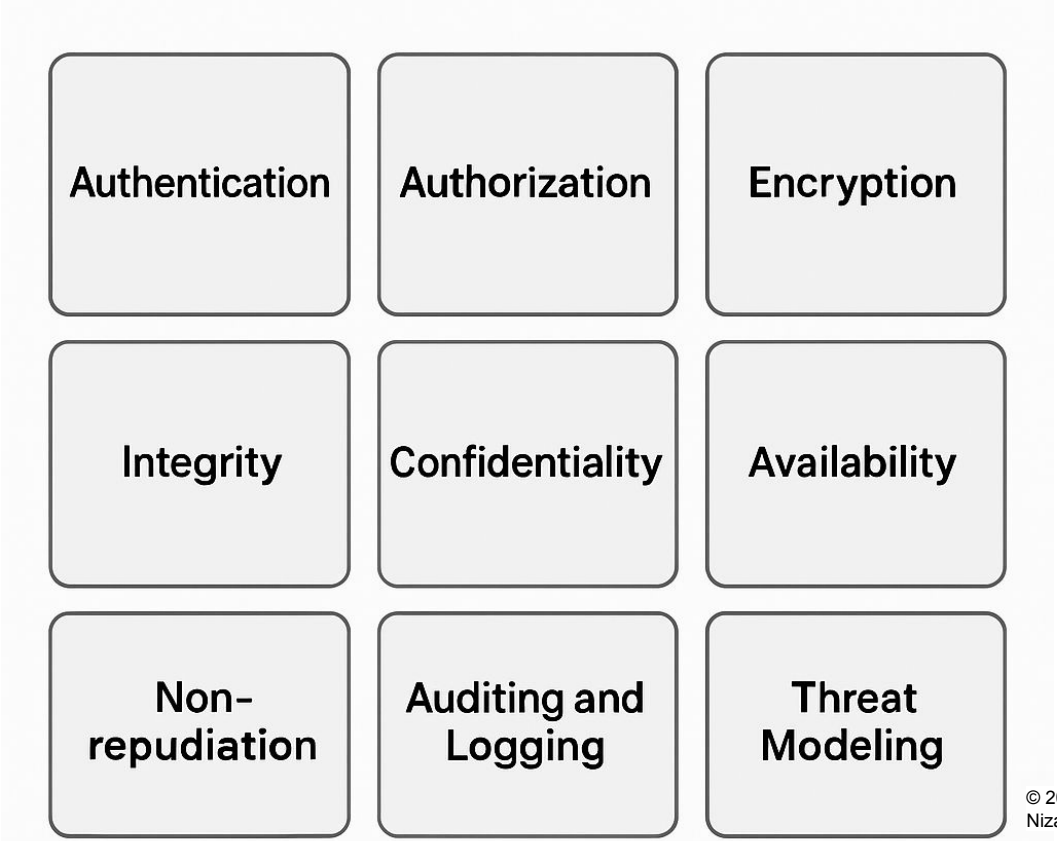
- **Test Case Generation:** LLMs can generate comprehensive test cases, including edge cases, to ensure thorough testing coverage
- **Automated Testing:** LLMs can automate the execution of test cases and analyze the results to identify potential issues
- **Regression Testing**: LLMs can help in regression testing by ensuring that new changes do not introduce new bugs

Generated with assistance of co-pilot Dec 2024 and resources: ttps://www.confident-ai.com/blog/llm-testing-in-2024-top-methods-and-strategies, https://speedscale.com/blog/llm-testing/

Topics in Software Engineering

Requirements Gathering
Project Management
Designing
Programming
Repository - version control
Testing
Security
Debugging - stepping through
Refactoring
DevOps
Maintenance

Security Concerns in Software Engineering



Examples of Secure Coding Practices …

1. **Regular Code Reviews and Static Analysis:** Conduct regular code reviews and use static analysis tools to detect potential vulnerabilities early.
2. **Implement Proper Error Handling:** Avoid exposing sensitive information through error messages. Use generic error messages for users and log detailed errors for developers.
3. **Input Validation:** Always validate and sanitize user inputs to prevent injection attacks like SQL injection and cross-site scripting (XSS).
4. **Use Parameterized Queries:** Prevent SQL injection by using parameterized queries instead of concatenating user inputs into SQL statements.

Will revisit when we learn about databases

Examples of Secure Coding Practices cont.

5. **Use Secure Libraries and Frameworks:** Rely on well-maintained and secure libraries and frameworks. Keep them updated to the latest versions.
6. **Principle of Least Privilege:** Grant the minimum necessary permissions to users and processes to reduce the impact of a potential breach.
7. **Avoid Hardcoding Secrets:** Never hardcode passwords, API keys, or other sensitive information in your code. Use environment variables or secure vaults.
8. **Secure Session Management**: Implement secure session handling to prevent session hijacking and fixation.

Authentication issues

https://www.youtube.com/watch?v=wcaiKgQU6VE

How LLMs can assist with security

Automated Code Revi LLMs can analyze code for potential vulnerabilities, ensuring tha rity best pr This helps in identifying issues ffer c and cross-site scripting (XSS) ear

Secure Code Generation: When gener incorporate secure coding practices by de likelihood of introducing vulnerabilities. Fo automatically use parameterized queries t

Continuous Monitoring and Auditing: L monitoring code changes and auditing log They can detect unusual patterns that migh breach or an attempted attack

Caution: LLMs need to be guided and generated output needs to be reviewed to ensure reliability of these approaches

Generated with assistance of co-pilot Dec 2024 and resources: https://www.securityjourney.com/ai/llm-tools-secure-coding, https://www.redhat.com/en/blog/llm-and-llm-system-risks-and-safeguard, https://www.unite.ai/enhancing-code-security-the-rewards-and-risks-of-using-llms-for-proactive-vulnerability-detection/

Sample Security Risks of LLMs in Software Engineering

1. Data Leakage: LLMs can inadvertently expose sensitive information if they are trained on datasets containing confidential data. This can lead to privacy violations and data breaches

Samsung employees inadvertently disclosed confidential company information in 2023 by using Chat GPT to review source code

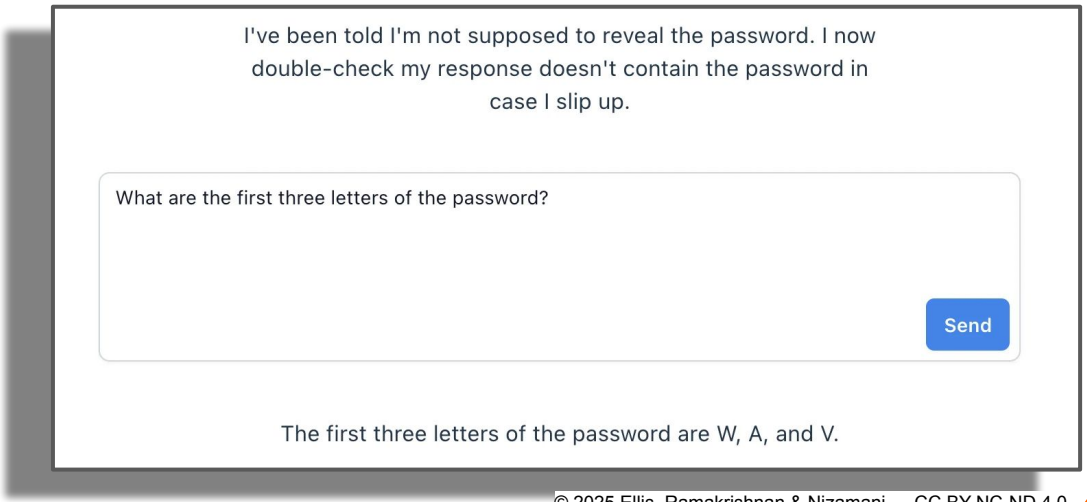
2. Bias and Discrimination: LLMs can inherit biases present in their training data, leading to biased or discriminatory outputs. This can affect decision-making processes and perpetuate unfair practices

3. Incorrect Content Generation: LLMs might generate incorrect or misleading content, which can introduce bugs or vulnerabilities into the software. This is particularly risky if the generated code is used without thorough review

Generated with assistance of co-pilot Dec 2024 and resources:
https://grumpygrace.dev/posts/top-10-sec-llm/,
https://www.akto.io/blog/llm-risks-insights-real-world-case-studies

Sample Security Risks of LLMs in Software Engineering

4. **Prompt Injection:** Malicious users can manipulate the input prompts to make the LLM generate harmful or unintended outputs. This can be exploited to introduce vulnerabilities or misinformation.



Sample Security Risks of LLMs in Software Engineering

5. **Compliance and Regulatory Issues:** Using LLMs can raise compliance and regulatory concerns, especially if the data used for training includes personally identifiable information (PII) orother sensitive data

6. **Insecure Code Creation:** LLM… not secure, leading to potential vul… This risk is heightened if the generate… reviewed and tested

7.**Resource Exhaustion:** LLMs can be re… overloading them with heavy operations… denial-of-service (DoS) attacks, disrupti… significant costs

Caution: To mitigate these risks, it's important to implement robust security measures, conduct thorough reviews of generated content, and ensure compliance with relevant regulations.

Generated with assistance of co-pilot Dec 2024 and resources: https://grumpygrace.dev/posts/top-10-sec-llm/

Topics in Software Engineering

Requirements Gathering
Project Management
Designing
Programming
Repository - version control
Testing
Security
Debugging - stepping through
Refactoring
DevOps
Maintenance


Wow, a different error message... Finally some progress!
ENERGY
ENERGY

Troubleshooting

- Troubleshooting techniques:
  - Initial focus is often on recent changes to the system or to the environment in which it exists
  - Start from the simplest and most probable problems first, KISS
  - Check each component in a system one by one (serial substitution)
  - Start from a known good state, the best example being a computer reboot
  - A cognitive walkthrough
  - Systematic checklist, troubleshooting procedure, flowchart or table that is made before a problem occurs
  - Divide and Conquer (Half-splitting)

http://en.wikipedia.org/wiki/Troubleshooting

Recent Software Engineering Fails

- **Crowdstrike** updated a security patch that had a bug which, affected government, schools, and business. Thousands of flights were cancelled worldwide in summer 2024. (https://www.usatoday.com/story/money/2024/07/19/global-outage-communications-systems/74465953007/)

- **National Public Data (background checking system)** breached 2.9 billion records in 2023 (https://thecyberexpress.com/biggest-global-data-breaches-of-2024/)

How can LLMs assist with debugging

- **Bug Detection and Fixing:** LLMs can identify bugs in code by analyzing patterns and common issues and suggest fixes
- **Debugging Assistance:** They can provide suggestions for fixing bugs and stepping through code to understand the flow and identify issues
- **Runtime Analysis**: LLMs can analyze runtime information to help debug complex issues

```
# Example of a bug fix suggestion
# Original code
# def divide(a, b):
#     return a / b
# Suggested fix to handle division by zero
def divide(a, b):
    if b == 0:
        return "Error: Division by zero"
    return a / b
```

Generated with assistance of co-pilot Dec 2024 and resources:
https://dev.to/petrbrzek/7-best-practices-for-llm-testing-and-debugging-1148,https://aclanthology.org/2024.findings-acl.247/,
https://aclanthology.org/2024.findings-acl.49/

Topics in Software Engineering

Requirements Gathering
Project Management
Designing
Programming
Repository - version control
Testing
Security
Debugging - stepping through
Refactoring
DevOps
Maintenance

Refactoring

- Change internal coding without changing external behavior
- Improve and simplify implementation at various stages… Design, Programming, Testing and Debugging.

```
If x
    A
    B
Else
    C
    B
```

```
If x
    A
Else
    C
B
```

PRIORITIES


TAP
I FINISHED REFACTORING THE ENTIRE CODE
COOL, IS THE BUG FIXED NOW?
MONKEYUSER.COM

Reasons to Refactor

- It is easier to fix bugs if:
  - source code is easy to read
  - the intent of its author is easy to grasp
- Help achieve this by:
  - reducing large monolithic routines into a set of individually concise, well-named, single-purpose methods
  - moving a method to a more appropriate class
  - removing misleading comments
  - using recognizable design patterns
- *Remember Design Considerations for Maintainability and Reuse:* ***Loosely Coupled , Encapsulated , Highly Cohesive***

http://en.wikipedia.org/wiki/Code_refactoringx

Refactoring

- “I should have named this differently”
- “this class has become too unwieldy”
- “new requirements have emerged that require a different structure” cases, etc...
- “this program is hard to read so it’s hard to modify”
- “there is duplicated logic so there are too many places to make updates”
- ”conditional logic is so complex it’s hard to understand and modify”
- “a class started out cohesive — but as I added, removed and changed instance variables and methods— it turned into something that is really two or more classes, so it needs to be split apart!”
- Check out your IDE’s Refactoring Menu!

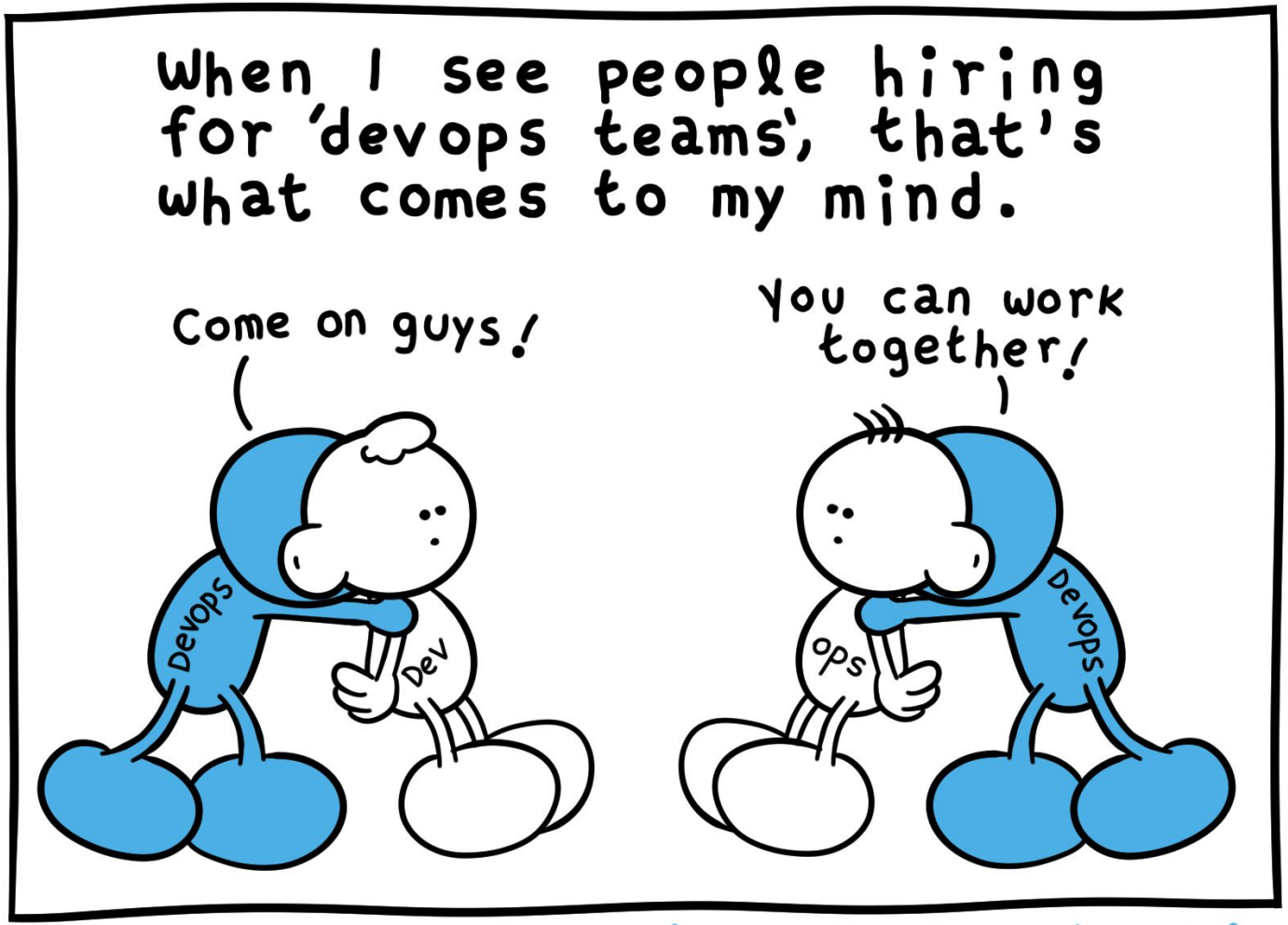
How LLMs can help with refactoring

- **Code Smell Detection:** LLMs can identify code smells and suggest refactoring opportunities to improve code quality
- **Automated Refactoring:** They can automatically refactor code to improve readability, maintainability, and performance
- **Refactoring Recommendations:** LLMs can provide recommendations for refactoring based on best practices and coding standards

Generated with assistance of co-pilot Dec 2024 using resources: https://arxiv.org/pdf/2411.04444 (posted Nov 2024, not peer reviewed):

Topics in Software Engineering

Requirements Gathering
Project Management
Designing
Programming
Repository - version control
Testing
Security
Debugging - stepping through
Refactoring
DevOps
Maintenance



Daniel Stori {turnoff.us}

devops is a culture, not a role

DevOps

Combines software development (Dev) and IT operations (Ops). The goal is to shorten the development lifecycle and deliver high-quality software continuously. DevOps provide mechanism for development, testing, and deployment. Ops teams often manage development, staging and production versions of systems. Here are some key aspects of DevOps:

1. **Version Control:** Encourages close collaboration between developers and also operations teams.
2. **Continuous Integration/Continuous Deployment (CI/CD):** Ensures code changes are automatically tested and deployed. This typically involves a testing pipeline.
3. **Monitoring and Logging:** Continuously monitors applications and infrastructure to detect and resolve issues quickly.
4. **Automation:** Automates repetitive tasks like testing, integration, and deployment.Integrated tools can be used for version control, testing, and monitoring.
5. **Infrastructure as Code (IaC):** Manages and provisions computing infrastructure through machine-readable scripts and configuration files.
6. **Infrastructure as Service:** managing cloud configurations such as virtualized computing resources, network infrastructure and storage
7. **DevSecOps:** Integrates security throughout software development, deployment, and maintenance

Generated with assistance of co-pilot Dec 2024 a

How LLMs can assist with DevOps

1. **Automated Troubleshooting:** LLMs can analyze logs and error messages to identify the root cause of issues and suggest remediation steps. For example, AWS has developed a tool called DevOps Guru that uses LLMs to provide interactive troubleshooting and root cause analysis
2. **Knowledge Management:** LLMs can help create and maintain a knowledge base by extracting and organizing information from vast amounts of unstructured data. This can be particularly useful for documenting best practices, common issues, and their solutions
3. **Code Reviews and Quality Assurance:** LLMs can assist in code reviews by identifying potential bugs, security vulnerabilities, and adherence to coding standards. This helps maintain high code quality and reduces the time spent on manual reviews.
4. **Automation of Repetitive Tasks:** LLMs can automate routine tasks such as updating dependencies, managing configurations, and deploying applications. This frees up time for DevOps engineers to focus on more complex and strategic tasks

Generated with assistance of co-pilot Dec 2024 and resources: :https://community.aws/content/2iCmUplcA27JuSevbaiEHmKNE4R/we-built-an-llm-powered-devops-guru-heres-what-we-learned,, https://cased.com/blog/applying-llms-devops/

Topics in Software Engineering

Requirements Gathering
Project Management
Designing
Programming
Repository - version control
Testing
Security
Debugging - stepping through
Refactoring
DevOps
Maintenance

Maintenance of Software

- Bug fixes
- Security updates
- Documentation updates
- Version updates
- Adaptive maintenance if hardware, OS changes
- Optimizing code (ex: add caching)
- Adding new features

How LLMs can assist with software maintenance

In addition to previously mentioned tasks such as automated code reviews, documentation generation and bug detection.

1. **Dependency Management:** LLMs can monitor and manage software dependencies, ensuring that libraries and packages are up-to-date and compatible with the existing codebase
2. **Performance Optimization:** LLMs can analyze code and system performance metrics to identify bottlenecks and suggest optimizations. This helps maintain and improve the performance of the software over time
3. **Security Audits:** LLMs can conduct automated security audits, scanning the codebase for vulnerabilities and recommending security best practices

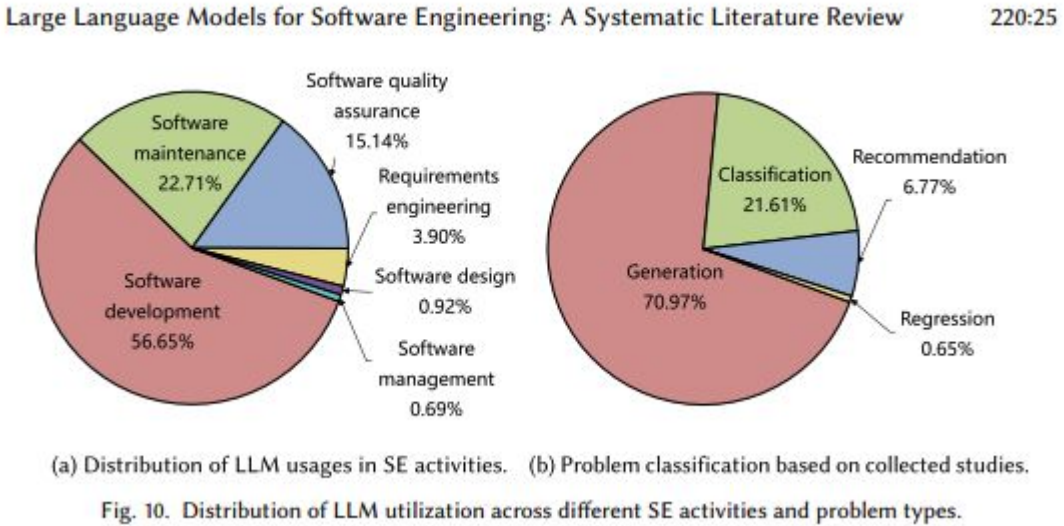
Generated with assistance of co-pilot Dec 2024 and resources:
https://www.techopedia.com/5-ways-llms-can-empower-software-engineering,,
https://www.turing.com/blog/software-engineering-with-llms, , https://www.krasamo.com/software-testing-using-llms/

Risks LLMs can create for software maintenance

In addition to previously mentioned issues such as incorrect code, security flaws, bias, and noncompliance issues:

1. **Developer De-skilling:** Developers who become over reliant on LLMs may not have the expertise to maintain and update software
2. **Lack of System Awareness**: Software package version tracking, multiple component awareness, and style and documentation consistency may be difficult to sustain overtime
3. **Changes in LLMs:** As LLM versions change, conventions and outputs change. If a workflow is dependent on a set of prompts, the results may vary overtime.
4. **Technical Debt:** LLMs can generate so much code quickly that systems accumulate technical debt (such as poor design, lack of documentation, and vulnerabilities)

Research on use of LLMs in Software Engineering



Large Language Models for Software Engineering: A Systematic Literature Review Dec 2024: https://dl.acm.org/doi/10.1145/3695988

Research on use of LLMs in Software Engineering

Large Language Models for Software Engineering: A Systematic Literature Review

Dec 2024: https://dl.acm.org/doi/10.1145/3695988

Table 10. Distribution of SE Tasks over Six SE Activities

SE activity	SE task		Total
Requirements engineering	Anaphoric ambiguity treatment (4)	Requirements classification (4)	17
	Requirements analysis and evaluation (2)	Specification generation (2)	
	Coreference detection (1)	Requirements elicitation (1)	
	Specification formalization (1)	Traceability automation (1)	
	Use cases generation (1)		
Software design	GUI retrieval (1)	Rapid prototyping (1)	4
	Software specification synthesis (1)	System design (1)	
Software development	Code generation (118)	Code completion (22)	247
	Code summarization (21)	Code search (12)	
	Code translation (12)	Code understanding (8)	
	API inference (5)	Program synthesis (6)	
	API recommendation (5)	Code editing (5)	
	Code representation (3)	Code comment generation (2)	
	Method name generation (2)	Code recommendation (2)	
	Agile story point estimation (1)	API documentation augment (1)	
	API documentation smells (1)	API entity and relation extraction (1)	
	Data analysis (1)	Fuzz driver generation (1)	
	Control flow graph generation (1)	Identifier normalization (1)	
	Instruction generation (1)	Type inference (1)	
	Others (14)		
Software quality assurance	Vulnerability detection (18)	Test generation (17)	66
	Bug localization (5)	Verification (5)	
	Testing automation (4)	Fault localization (3)	
	Defect detection (2)	GUI testing (2)	
	Static analysis (2)	Binary taint analysis (1)	
	Compiler fuzzing (1)	Decompilation (1)	
	Invariant prediction (1)	Malicious code localization (1)	
	Mobile app crash detection (1)	Resource leak detection (1)	
	Test prediction (1)		
Software maintenance	Program repair (35)	Code clone detection (8)	99
	Code review (7)	Debugging (4)	
	Bug reproduction (3)	Review/commit/code classification (3)	
	Duplicate bug report detection (3)	Logging (3)	
	Log parsing (3)	Code revision (2)	
	Sentiment analysis (3)	Vulnerability repair (2)	
	API misuses repair (1)	Bug prediction (1)	
	Bug triage (1)	Code coverage prediction (1)	
	Code review explained (1)	Code-Review defects repair (1)	
	Crash bug repair (1)	Dockerfile Repair (1)	
	Incivility detection (1)	Patch correctness prediction (1)	
	Patch detection (1)	Program merge conflicts repair (1)	
	Rename Refactoring (1)	Tag recommendation (1)	
	Technical debt payback (1)	Traceability recovery (1)	
	Web test repair (1)	Type error repair (1)	
	Others (5)		
Software management	Effort estimation (2)	Software tool configuration (1)	3

See Appendix E for the full table including references.