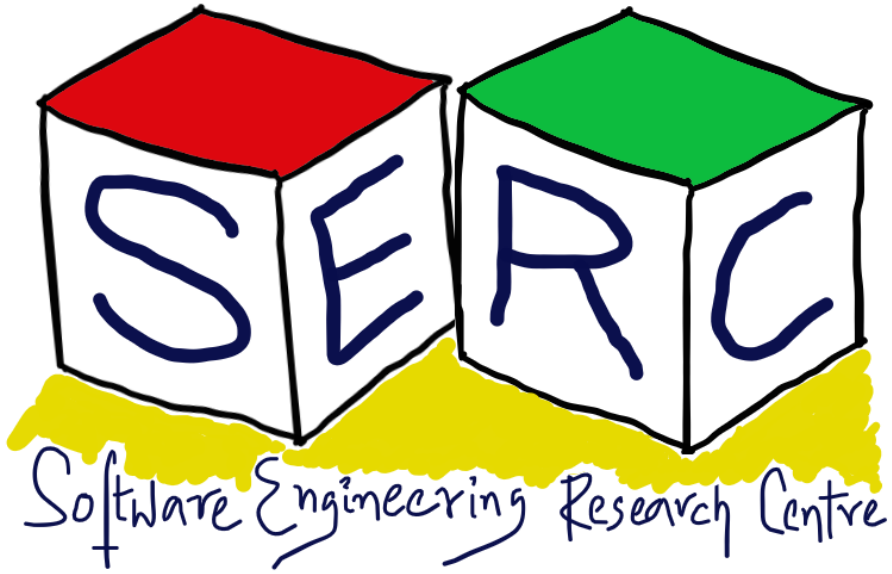
Introduction and Course Overview

CS6.401 Software Engineering



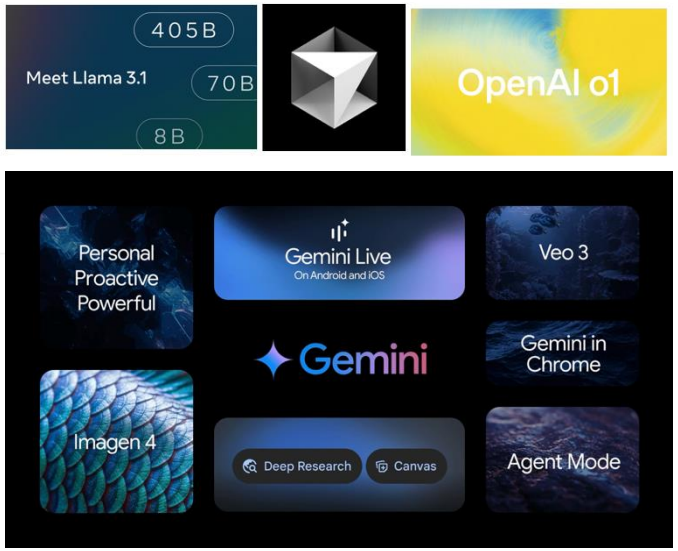
Karthik Vaidhyanthan

karthik.Vaidhyanthan@iiit.ac.in

https://karthikvaidhyanathan.com

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

The Evolving Technology Landscape



GenAI Everywhere!!

Digital Twins with 5G, AR/VR/XR

AI Coming to the Edge!

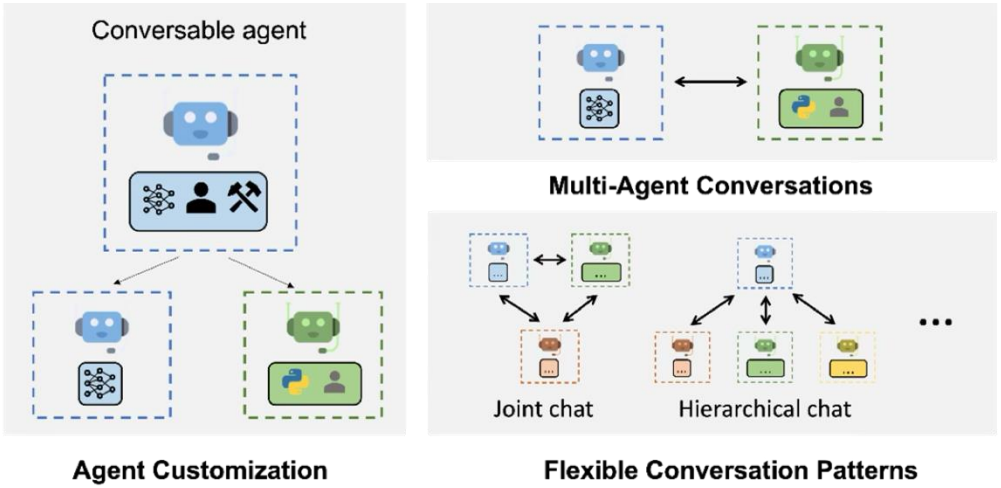
Increasing concern over Sustainability!

https://www.roadtovr.com
https://www.apple.com/in

Software-driven World

Everything is increasingly powered by Software!

Smart bottles, watches



Agentic AI is about SE for AI

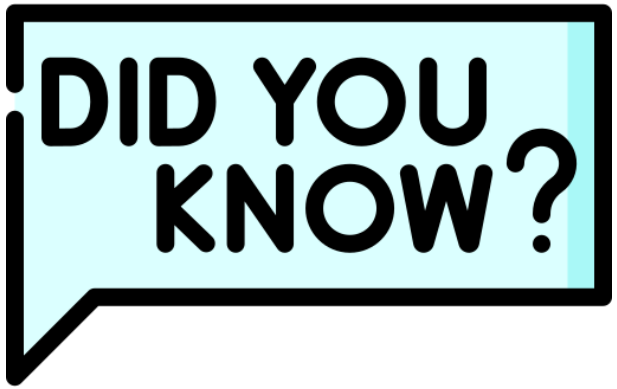
Boston Dynamics

Tesla Car in Autopilot mode

Source: Google images, tesla, openai

Some Interesting yet Serious Facts

1. About **70%** of the cost goes for software maintainence
2. Modern car runs **100 million** lines of code - Smartphone on wheels!
3. Single autonomous car can generate up to **4TB of data** per day!
4. **> 50%** of the ML systems considered failures in production
5. **95%** of GenAI Pilots at companies are failing
6. Software emissions are equivalent to air, rail and shipping combined!!
7. ……



Sources: Gartner, Cisco

What is the big deal?

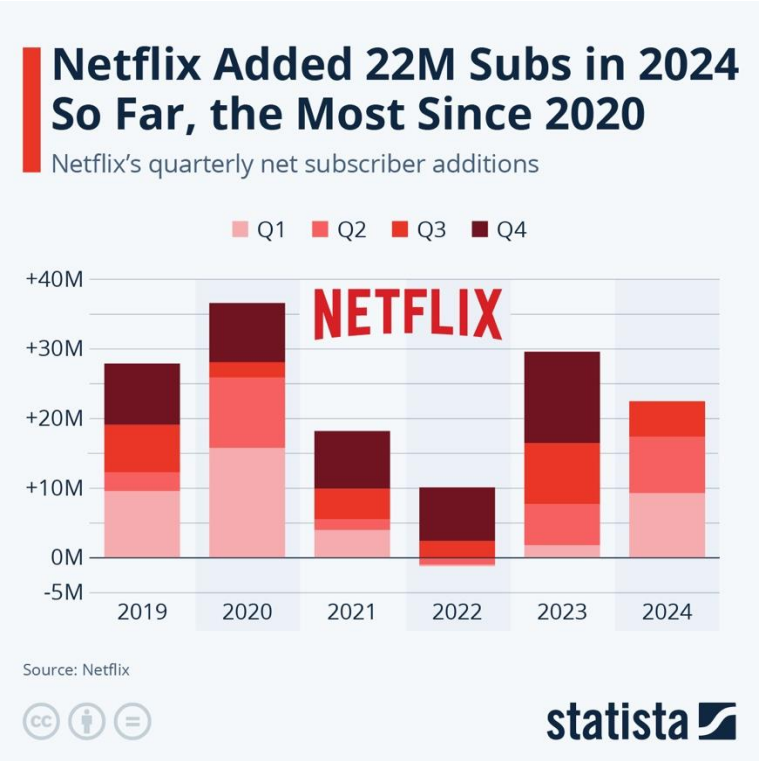
Let’s draw some parallels

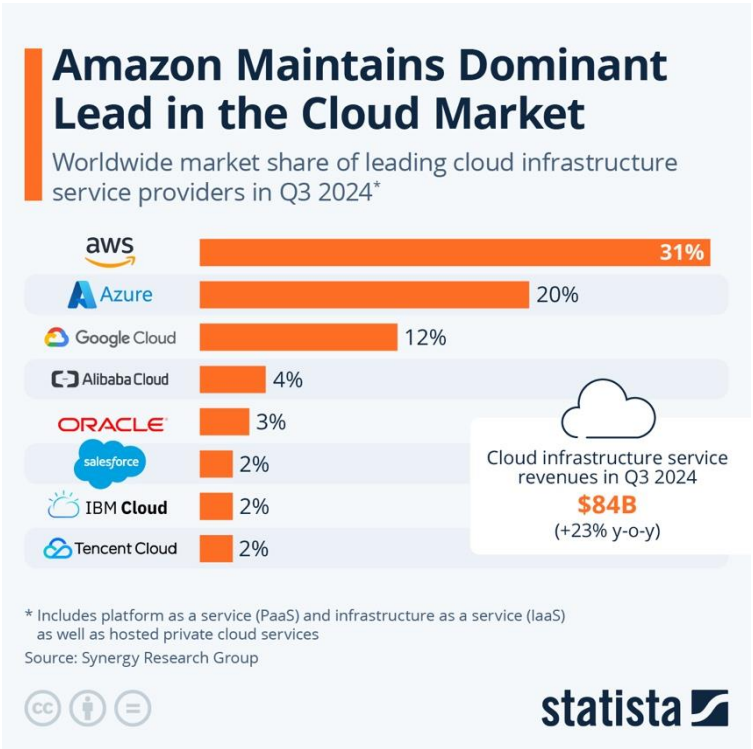
When do you say something is well engineered?

Functionality + Performance, Scale, Maintainability,.....

How Disney+ Hotstar managed its historic 5.9 crore concurrent viewership during WC finals

Mukund Acharya, Head of Technology, Disney+Hotstar, in an exclusive interview, revealed the tech behind Disney+ Hotstar's benchmark achievement during the ICC World Cup 2023.





Source: Googe images, Indianexpress, statista

Is Software that Critical?

CrowdStrike issue – The consequences

- 8.5 million devices affected! - 1% of the entire windows users
- Airlines - Delta, KLM, Ryanair, Lufthansa,. Many passengers were stranded in airports
- Financial organisations - London stock exchange, Visa, …
- Healthcare - independent pharmacies and GP surgeries
- Media, retail, train operating companies…….

Economic and social impact!!!

Few of many software issues

Microsoft Azure down: What caused the outage? Which services are affected?

A DNS failure in Azure Front Door triggered widespread outages across Azure Portal, Outlook, Xbox multiplayer, and other Microsoft 365 services. Microsoft is investigating and offering temporary workarounds as recovery continues.

Written By Ravi Hari

Published • 29 Oct 2025, 10:47 PM IST

Add mint

Cloudflare apologises for outage which took down X and ChatGPT

18 November 2025

Share Save

Liv McMahon

Technology reporter

ChatGPT down for users globally, OpenAI shares major update on technical outage

By HT News Desk

Dec 12, 2024 08:15 AM IST

A significant outage has impacted OpenAI's ChatGPT and API services, leading to user frustration and increased complaints.

Amazon reveals cause of AWS outage that took everything from banks to smart beds offline

AWS explains in a lengthy post how a bug in automation software brought down thousands of sites and applications

Even downdetector was down ;)

Sources: Guradian, BBC Livemint

Few of the many software issues

IT failures causing patient deaths, says NHS safety body

19 December 2023

Share Save

Sharon Barbour, Nat Wright and Philippa Roxby
BBC News

Catastrophic software errors doomed Boeing's airplanes and nearly destroyed its NASA spaceship. Experts blame the leadership's 'lack of engineering culture.'

MORGAN MCFALL-JOHNSEN | FEB 29, 2020, 18:41 IST

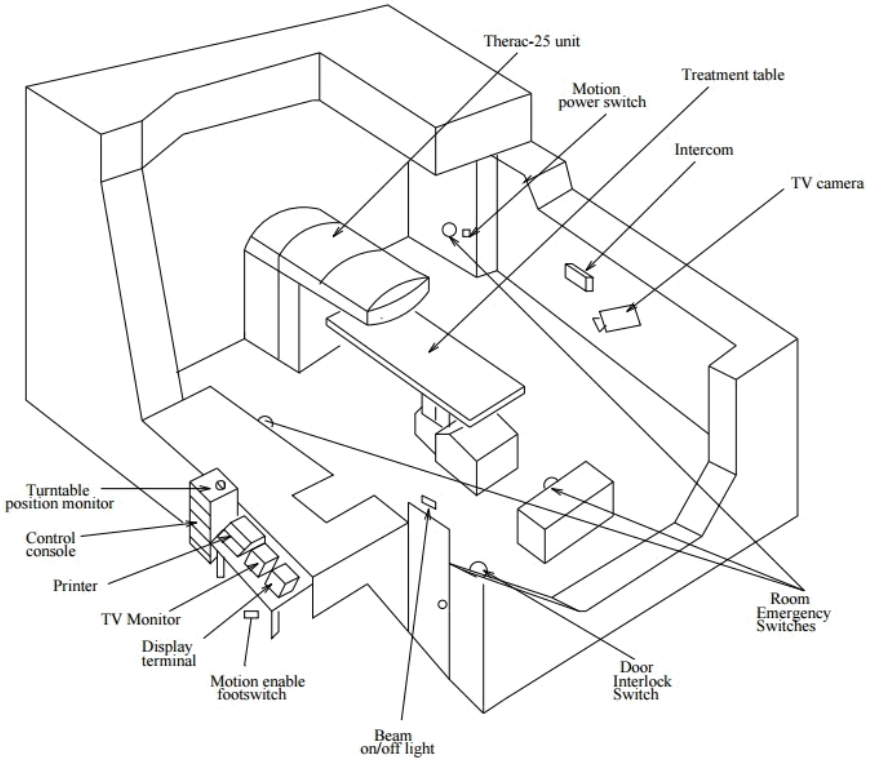


Figure 5: A typical Therac-25 facility after the final CAP.

Can be fatal!!

Source: Googe images, semantic scholar

Why Software Engineering?

Why Study Software Engineering?

- Acquire skills to understand and build high quality software systems
  - Systems could be large and complex - millions of lines of code
  - How to comprehend an existing system?
  - How to break a program into smaller and manageable parts?
  - Development of large and complex software requires teamwork
  - Achieve sufficient quality (eg: maintainability, performance, reliability etc.)
- Learn techniques: specification, designing, development, testing, project management, etc.

Its still a **young discipline –** The term “software engineering” was just coined in 1968

Source: Google images

Software Engineer is not just another title

- Software engineer is considered as just another role sometimes
- Many countries require software engineers to go through a certification or licensing process
- Going forward software engineers will be dealing with more and more critical systems
- Software for critical systems need to go through certification process

Licensing is already in place in some countries for some specific SE roles!!

Licensing Professional Software Engineers: Seize the Opportunity

By Phillip A. Laplante
Communications of the ACM, July 2014, Vol. 57 No. 7, Pages 38-40
10.1145/2618111
Comments (1)

VIEW AS: SHARE:

In his July 2013 *Communications* column, ACM President Vint Cerf revisited the controversy of licensing professional software engineers in the U.S. This issue has been one that divides the profession since reasonable cases can be made both pro and con. Officially, ACM has opposed and IEEE has supported such licensure. Even within both organizations, though, there is substantial support and opposition.[1,2,3,5] I think President Cerf was right in suggesting that, because of the dramatic growth in interacting software in both devices and in applications that significantly impact peoples' lives, and more importantly, because licensing is happening, we have reached a tipping point.

Source: Googe images, c-ACM

Don’t worry we have SE course!!

What probably you have done so far?

- **IIIT Undergrad:** Introduction to software systems (ISS), low level OO principles and some high-level design in Design and Analysis of Software Systems (DASS)

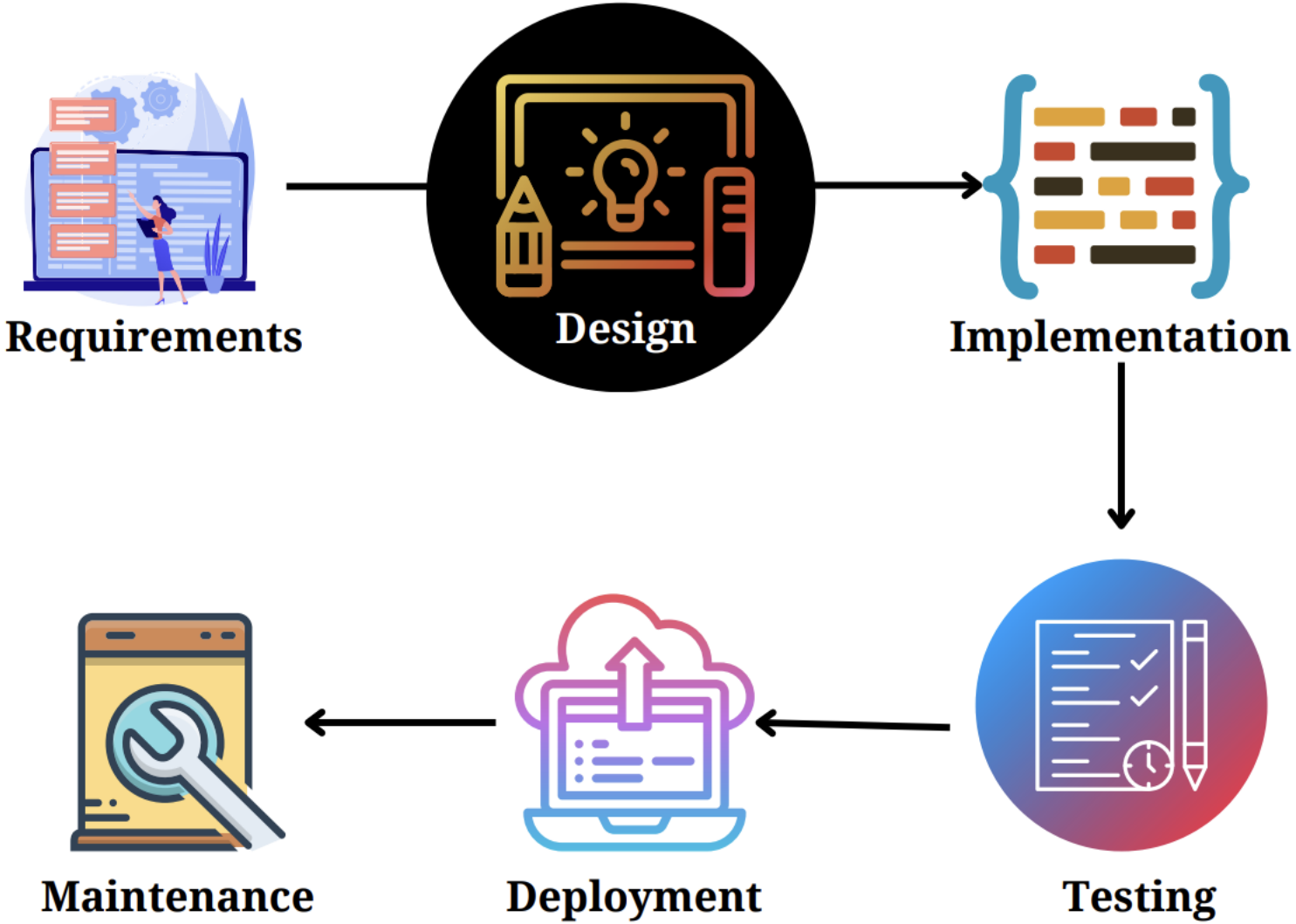
- **Grads at IIIT:** Problem solving course and Software Systems Development

- **Others:** Some software development experience or an undergrad course in software/systems engineering or similar courses

Minimum Prerequisites

1. Basic idea on: Abstraction, Modularization/Decomposition, Coupling, Cohesion, etc.
2. Some programming language(s) (eg:, python, JS etc.)
3. Basic OO principles and implementations: Encapsulation, inheritance, Polymorphism
4. At least 1 OOP language, & 1 RDBMS.
5. Idea about various phases in SDLC
6. Minimal knowledge of practices: Version control, bug tracking, task management, etc.

Software Development Lifecycle



This course is about
Software Design!

Lets take an Example Scenario

Assume that you have been appointed to XYZ.com which is an e-commerce company. You have been asked to develop the authentication feature for the e-commerce system.

It can be broken down to something like this!

Write a simple program that takes two strings, inputString1 and inputString2 as inputs and compares them with userName and password respectively. If the strings match, then output should be True, else False

Sample input

user password

Sample Output

True

This is a simple string match program!!!

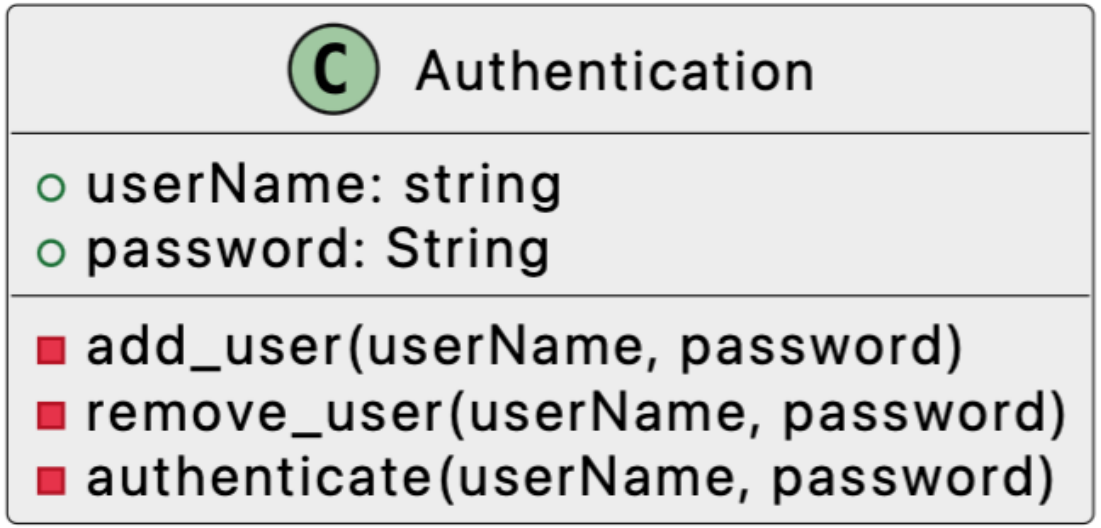
```
boolean compare(inputString1, inputString2)
{
.....
}
```

Not at this level!!

What kind of design? - Lets' go little more higher

Modify the above into an authentication module. You should be able to add two new strings (username, password) and the program should be able to find a match given two strings.

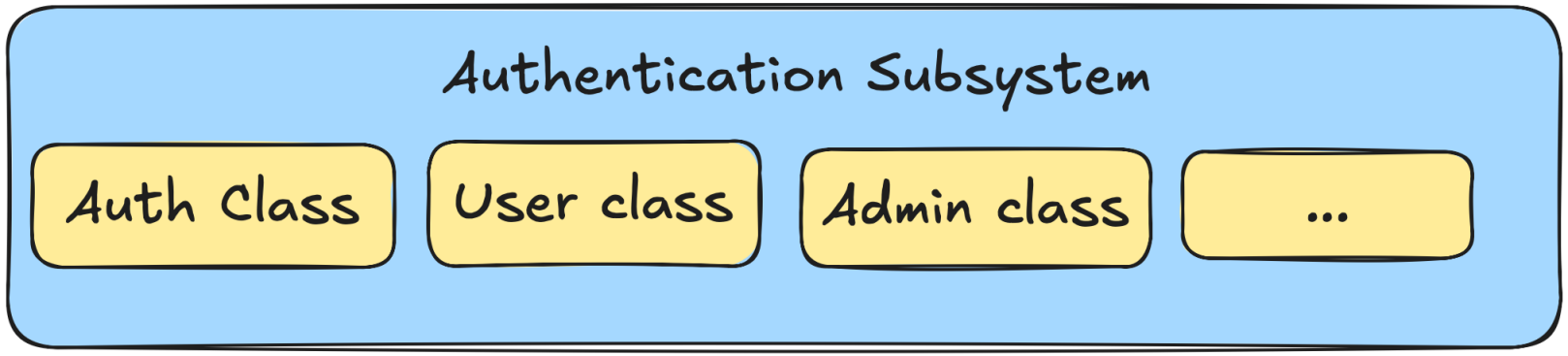
Little at this level!!



What kind of design? - Lets' go little more higher

Modify the above into an authentication subsystem. Any user should be able to register and login. The system should support multiple types of user

Yes, this level!!



What kind of design? - Lets' go even further higher

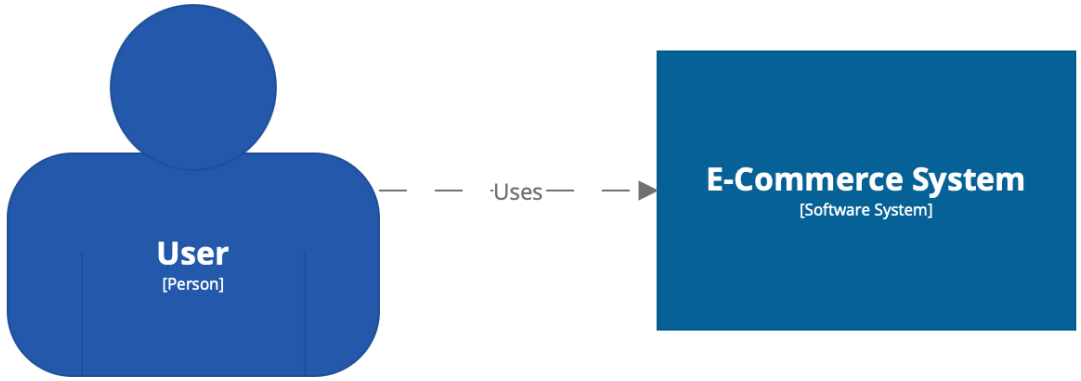
Assume that there is another sub system that allows users to add payment information and make payments. Now, integrate this with the authentication system.

Yes, this level!!

[Container] E-Commerce System
Thursday 2 January, 2025 at 9:28 PM India Standard Time

What kind of design? - Lets' go even further higher

What is the highest level of abstraction we can go into



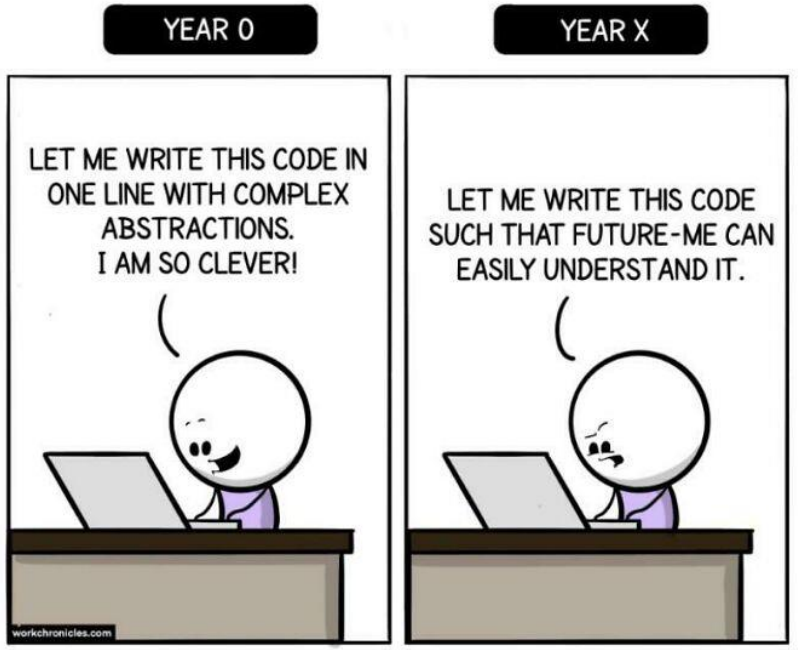
[System Context] E-Commerce System
Thursday 2 January, 2025 at 9:29 PM India Standard Time

Yes, starting from this level!!

What all needs to be considered? – scalability, performance, reliability,....

Why Design is important?

- It is like technical kernel of Software engineering
- Ensures traceability between requirements and final product
- Allows to have a complete picture of the software before construction
- Allows to discuss, confirm, reason and perform different analysis even before development - **why?**



Source: google images

*"Software is **not limited by physics**, like buildings are. It is **limited by imagination**, **by design**, by organization. In short, it is limited by properties of people, not by properties of the world. We have **met the enemy, and he is us**"*

Ralph Jhonson

Martin Fowler, ***Who needs an Architect?*** IEEE Software, 2003

Source: Googe images

SE - Major Principles

Abstraction

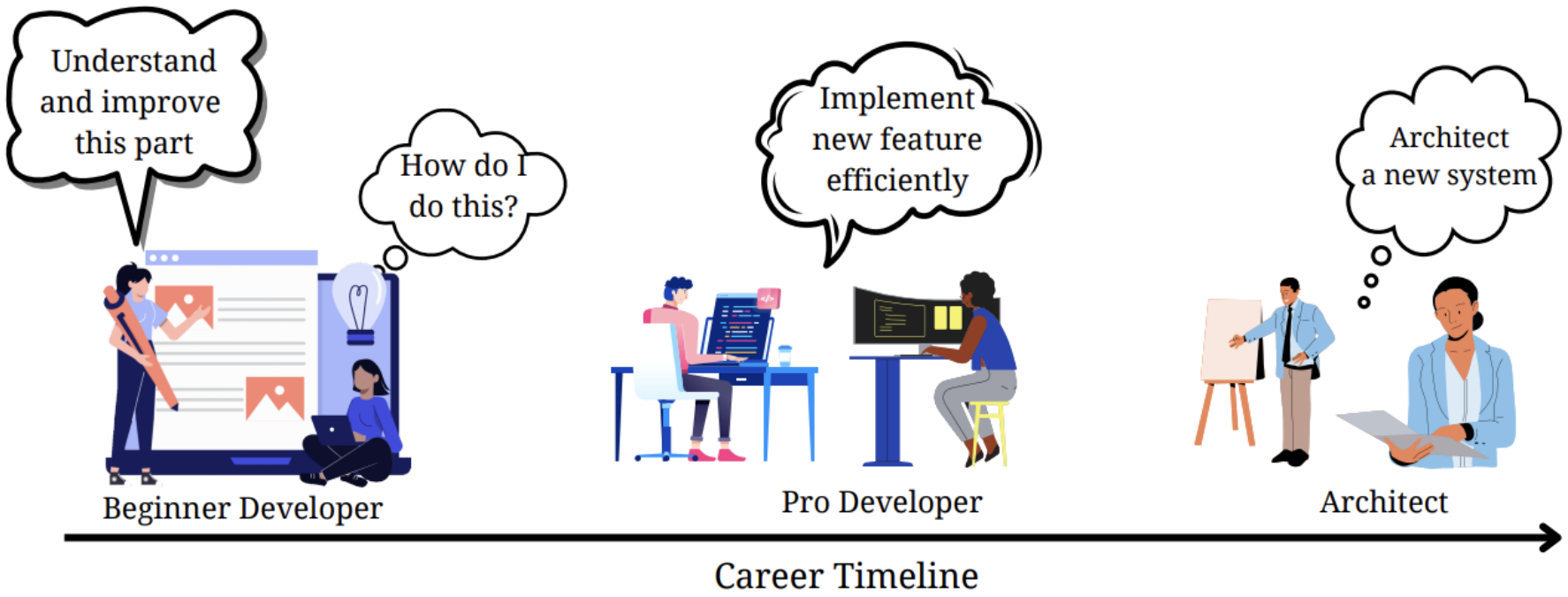
1. Focus on the relevant parts of the problem
2. Helps simplifying the problem and look at a larger picture

Decomposition

1. Decompose problem into small and independent pieces
2. Each pieces can be solved separately
3. The overall solution can be attained by solving smaller pieces and putting them together

What will you learn?

Course Outline



This Course

Modeling	Design Principles	Architectural Framework
Refactoring	Design Patterns	Architectural Patterns

Course Outline

How to comprehend existing software subsystems?

- Applying principles of refactoring

How to design software systems through abstractions and decompositions?

- Design Patterns and Architectural Patterns

How to apply them to your application?

- Deal with subsystems at higher level of abstraction provided by patterns

What if there is no exact match for a given problem?

- Evaluate alternatives and analyze trade-offs

How to capture and document the design decisions?

- Both at low level and high level

Across the above, if and how GenAI/AgenticAI can really help

Agentic AI tools can be an aid – Lot of changes to come in!

Code generation models

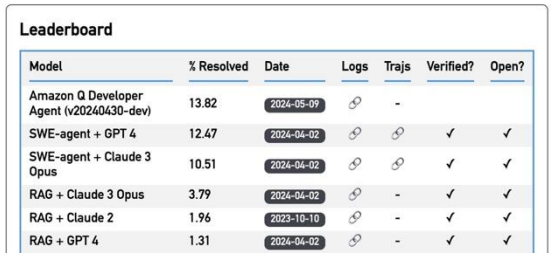
AI Programming Assistants

Tools

autocoderover

Vercel v0

Multi-agent frameworks



Leaderboard

Model	% Resolved	Date	Logs	Trajs	Verified?	Open?
Amazon Q Developer Agent (v20240430-dev)	13.82	2024-05-09	🔗	-		
SWE-agent + GPT 4	12.47	2024-04-02	🔗	🔗	✓	✓
SWE-agent + Claude 3 Opus	10.51	2024-04-02	🔗	🔗	✓	✓
RAG + Claude 3 Opus	3.79	2024-04-02	🔗	-	✓	✓
RAG + Claude 2	1.96	2023-10-10	🔗	-	✓	✓
RAG + GPT 4	1.31	2024-04-02	🔗	-	✓	✓

Leaderboards

Some of our works in GenAI for SE

Can LLMs Generate Architectural Design Decisions? - An Exploratory Empirical study

Rudra Dhar
Software Engineering Research Centre
IIIT Hyderabad, India
rudra.dhar@research.iiit.ac.in

Karthik Vaidhyanathan
Software Engineering Research Centre
IIIT Hyderabad, India
karthik.vaidhyanathan@iiit.ac.in

Vasudeva Varma
Language Technologies Research Centre
IIIT Hyderabad, India
vv@iiit.ac.in

***Abstract*—Architectural Knowledge Management (AKM) involves the organized handling of information related to architectural decisions and design within a project or organization. An essential artefact of AKM is the Architecture Decision Records (ADR), which documents key design decisions. ADRs are documents that capture decision context, decision made and various aspects related to a design decision, thereby promoting transparency, collaboration, and understanding. Despite their benefits, ADR adoption in software development has been slow due to challenges like time constraints and inconsistent uptake. Recent advancements in Large Language Models (LLMs) may help bridge this adoption gap by facilitating ADR generation. However, the effectiveness of LLM for ADR generation or understanding is something that has not been explored. To this end, in this work, we perform an exploratory study which aims to investigate the feasibility of using LLM for the generation of ADRs given the decision context. In our exploratory study, we utilize GPT and T5-based models with 0-shot, few-shot, and fine-tuning approaches to generate the Decision of an ADR given its Context. Our results indicate that in a 0-shot setting, state-of-the-art models such as GPT-4 generate relevant and accurate Design Decisions, although they fall short of human-level performance. Additionally, we observe that more cost-effective models like GPT-3.5 can achieve similar outcomes in a few-shot setting, and smaller models such as Flan-T5 can yield comparable results after fine-tuning. To conclude, this exploratory study suggests that LLM can generate Design Decisions, but further research is required to attain human-level generation and establish standardized widespread adoption.**

***Index Terms*—ADR, LLM**

I. Introduction

been a crucial reason restricting a wider adoption of AKM approaches, and more research is needed for automatically capturing this knowledge [3].

An *Architecture Decision Record (ADR)* is a crucial part of AKM. It entails the idea that software architecture is considered a set of Design Decisions [4]. It is a document used in software development to capture and document important *Architecture Design Decisions (ADD)*, made during the design and development of a software system. A detail explanation is given in Section II. Despite the well-established benefits of ADRs, their adoption has been slow to non-existent as described by Georg *et al.* [5]. Unsuccessful adoption of ADRs in software development can occur due to several factors, including inadequate tool support, effort needed to capture Architecture Knowledge (AK), interruptions to the design process caused by documenting AK, and uncertainty regarding which AK needs documentation. [5].

Large Language models (LLMs) excel in comprehending contexts and generating text accordingly. Over the recent years due to advancement of LLMs, text generation has become more accessible. This paper delves into the exploration of whether LLMs can effectively generate Architectural Decision Records (ADRs). While the prospect of generating entire ADRs from a codebase remains a task for future endeavours, the focus of this work is on utilizing LLMs to generate Design Decisions from decisions Contexts as these are recognized as the core components of an ADR[1].

In the realm of *Natural Language Processing (NLP)*,

Small Models, Big Tasks: An Exploratory Empirical Study on Small Language Models for Function Calling

Ishan Kavathekar*
International Institute of Information Technology
Hyderabad, India
ishankavathekar31@gmail.com

Raghav Donakanti*
International Institute of Information Technology
Hyderabad, India
raghav.donakanti@students.iiit.ac.in

Ponnurangam Kumaraguru
International Institute of Information Technology
Hyderabad, India
pk.guru@iiit.ac.in

Karthik Vaidhyanathan
International Institute of Information Technology
Hyderabad, India
karthik.vaidhyanathan@iiit.ac.in

Abstract

Function calling is a complex task with widespread applications in domains such as information retrieval, software engineering and automation. For example, a query to *book the shortest flight from New York to London on January 15* requires identifying the correct parameters to generate accurate function calls. Large Language Models (LLMs) can automate this process but are computationally expensive and impractical in resource-constrained settings. In contrast, Small Language Models (SLMs) can operate efficiently, offering faster response times, and lower computational demands, making them potential candidates for function calling on edge devices. In this exploratory empirical study, we evaluate the efficacy of SLMs in generating function calls across diverse domains using zero-shot, few-shot, and fine-tuning approaches, both with and without prompt injection, while also providing the finetuned models to facilitate future applications. Furthermore, we analyze the model responses across a range of metrics, capturing various aspects of function call generation. Additionally, we perform experiments on an edge device to evaluate their performance in terms of latency and memory usage, providing useful insights into their practical applicability. Our findings show that while SLMs improve from zero-shot to few-shot and perform best with fine-tuning, they struggle significantly with adhering to the given output format. Prompt injection experiments further indicate that the models are generally robust and exhibit only a slight decline in performance. While SLMs demonstrate potential for the function call generation task, our results also highlight areas that need further refinement for real-time functioning.

Keywords

Small Language Models, Function Calling, Edge AI, Reliability

ACM Reference Format:
Ishan Kavathekar, Raghav Donakanti, Ponnurangam Kumaraguru, and Karthik Vaidhyanathan. 2025. Small Models, Big Tasks: An Exploratory Empirical Study on Small Language Models for Function Calling. In *Evaluation and Assessment in Software Engineering (EASE '25), June 17–20, 2025, Istanbul, Turkiye.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3756681.3757001

1 Introduction

Advancements in LLMs have enabled strong performance across a wide range of tasks [23, 49]. They are increasingly being adopted in software engineering for tasks such as code generation [31], debugging [39], and API integration [48], as well as in systems for task automation. One critical application for LLMs is function calling and execution. LLMs often rely on extensive cloud-based infrastructure, raising concerns about privacy, latency, and high computational overhead [11, 45]. As a result, SLMs are emerging as a powerful alternative for real-world applications that can efficiently perform specific tasks locally, making them ideal for environments where responsiveness and data security are critical [40]. SLMs can potentially serve as a bridge between user inputs and backend systems in function calling applications, by converting natural language queries into structured function calls [40]. Function calling, though less common than NLP tasks like summarization or translation, is crucial for interpreting and executing user commands. It enables SLMs to perform real-world actions, making it essential in

Discussions during SE course in 2024 resulted in good work!

Reimagining Self-Adaptation in the Age of Large Language Models

Raghav Donakanti, Prakhar Jain, Shubham Kulkarni, Karthik Vaidhyanathan
Software Engineering Research Center, IIIT Hyderabad, India
raghav.donakanti@students.iiit.ac.in, prakhar.jain@research.iiit.ac.in, shubham.kulkarni@research.iiit.ac.in,
karthik.vaidhyanathan@iiit.ac.in

***Abstract*—Modern software systems are subjected to various types of uncertainties arising from context, environment, etc. To this end, self-adaptation techniques have been sought out as potential solutions. Although recent advances in self-adaptation through the use of ML techniques have demonstrated promising results, the capabilities are limited by constraints imposed by the ML techniques, such as the need for training samples, the ability to generalize, etc. Recent advancements in Generative AI (GenAI) open up new possibilities as it is trained on massive amounts of data, potentially enabling the interpretation of uncertainties and synthesis of adaptation strategies. In this context, this paper presents a vision for using GenAI, particularly Large Language Models (LLMs), to enhance the effectiveness and efficiency of architectural adaptation. Drawing parallels with human operators, we propose that LLMs can autonomously generate similar, context-sensitive adaptation strategies through its advanced natural language processing capabilities. This method allows software systems to understand their operational state and implement adaptations that align with their architectural requirements and environmental changes. By integrating LLMs into the self-adaptive system architecture, we facilitate nuanced decision-making that mirrors human-like adaptive reasoning. A case study with the SWIM exemplar system provides promising results, indicating that LLMs can potentially handle different adaptation scenarios. Our findings suggest that GenAI has significant potential to improve software systems' dynamic adaptability and resilience.**

***Index Terms*—Self-Adaptation, Software Architecture, Generative AI, LLM, Software Engineering**

The concept of autonomic computing, as proposed by Kephart and Chess [5], sought to enhance the autonomy of software systems through various strategies. Despite these efforts, a persistent challenge has been the ability of systems to dynamically generate new configurations and components. The advent of GenAI, particularly the capabilities of LLMs, introduces the possibility of developing adaptation strategies directly. This is supplemented by the fact that modern software systems generate vast amounts of data, including logs, metrics, and traces, which system administrators traditionally leverage for tasks such as root cause analysis and resource allocation. This data, encompassing code, APIs, JSON, XML, and more, can be converted into various natural language formats, enabling Large Language Models (LLMs) to interpret the data and generate adaptive responses akin to those formulated by human experts. To this end, this paper introduces an innovative framework, MSE-K (Monitor, Synthesize, Execute) that integrates Large Language Models (LLMs) into the self-adaptation process, enabling software systems to autonomously generate and implement contextually relevant and actionable architectural adaptation strategies aligned with their operational goals. This approach seeks to overcome the limitations of current self-adaptation mechanisms, paving the way for more efficient and intelligent software systems [6].

As an initial validation, we performed evaluations of our

POLARIS: Is Multi-Agentic Reasoning the Next Wave in Engineering Self-Adaptive Systems?

Divyansh Pandey*
SERC, IIIT - Hyderabad
divyansh.pandey@students.iiit.ac.in

Vyakhya Gupta*
SERC, IIIT - Hyderabad
vyakhya.gupta@students.iiit.ac.in

Prakhar Singhal*
SERC, IIIT - Hyderabad
prakhar.singhal@research.iiit.ac.in

Karthik Vaidhyanathan
SERC, IIIT - Hyderabad
karthik.vaidhyanathan@iiit.ac.in

Abstract

The growing scale, complexity, interconnectivity, and autonomy of modern software ecosystems introduce unprecedented levels of uncertainty, challenging the foundations of traditional self-adaptation. Existing self-adaptive techniques, often based on rule-driven controllers or isolated learning components, struggle to generalize across novel contexts and coordinate responses across distributed subsystems, leaving them ill-equipped to handle emergent *"unknown unknowns"* in complex, dynamic environments. Recent discussions on *Self Adaptation 2.0* established an equal partnership between AI and adaptive systems, merging learning-driven intelligence with adaptive control to enable predictive, data-driven, and proactive adaptation. Building on this foundation, we introduce a new wave of self-adaptation through *POLARIS* a three-layer multi-agentic self-adaptation framework that moves beyond reactive adaptation by combining (1) a low-latency Adapter layer for monitoring and safe execution, (2) a transparent Reasoning layer that generates and verifies plans using tool-aware, explainable agents, and (3) a Meta layer that records experiences and meta-learns improved adaptation policies over time. By using shared knowledge and predictive models, POLARIS can handle uncertainty, learn from its past actions, and evolve its strategies, enabling the engineering of autonomous systems that can anticipate change to ensure resilient and goal-directed behavior under uncertainty. Preliminary evaluation across two distinct self-adaptive exemplars, SWIM and SWITCH, demonstrates that POLARIS consistently outperforms existing state-of-the-art baselines. With this, we motivate a shift towards *Self-Adaptation 3.0* akin to *Software 3.0*, a new paradigm where systems move beyond merely learning from their environment to reasoning about and evolving their own adaptation. In this vision, adaptation contributes to a self-learning process that enables systems to continuously improve and respond to novel challenges.

intelligence (AI) into software ecosystems has fundamentally altered this landscape. AI-enabled systems introduce new forms of non-determinism, stemming not only from environmental variability but also from the adaptive and opaque nature of AI components themselves. While existing frameworks have started to incorporate AI-based prediction or decision modules, they largely treat AI as an auxiliary tool within classical control loops. This limits their ability to manage *emergent and evolving uncertainties* such as data drift, stochastic models, or co-evolving agents: scenarios increasingly common in AI-native systems. Moreover, adaptation in such systems must go beyond reactive adjustments; it must *learn from adaptation itself*. Systems should continuously refine their reasoning and decision strategies over time, improving with experience rather than merely responding to change [20]. This need for continual, self-improving adaptation marks a shift toward *AI-native self-adaptation*, where learning and reasoning are integral to the adaptation process.

Building on the vision of *Self-Adaptation 2.0* by Bureš [4], we argue that the next wave of self-adaptive systems [45] must treat intelligence as a first-class design principle. In this work, we present *POLARIS-Proactive Orchestrated Learning for Agentic and Reasoning-driven Intelligent Systems*, which redefines adaptation as an emergent property of interacting agents. POLARIS adopts and builds into a three-layer architecture proposed by Kramer et al. [22], comprising: (1) an Adapter layer for monitoring and safe execution; (2) a transparent Reasoning layer with tool-aware, explainable agents for planning and verification; and (3) a Meta layer to record experiences and meta-learn improved adaptation policies over time. These three layers combine to give the framework its orchestrated and continuous learning nature. We evaluate *POLARIS* on two distinct exemplars, *SWIM* [36] and *SWITCH* [29], highlighting its performance and broad applicability across both traditional legacy software systems and modern AI-based ones. On *SWIM, POLARIS*

Possibilities are always endless!!

Teaching and Learning Methodology

- Problem based learning methodology

- A blend of conceptual lectures with group/individual activities

- This has been a working formula
  - Learners engage in the materials
  - Hands on activities enables deeper understanding
  - Resembles the true career situation

Start forming the teams!!

Meet the Team

Karthik Vaidhyanathan

Chandrasekar S

Adarsh Sharma

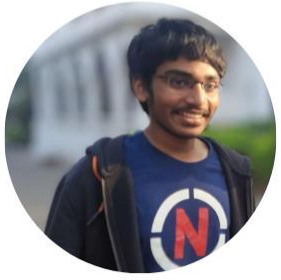
Akhila Matathammal

Anshi Gandhi

Chirag Damija

Divyansh Pandey

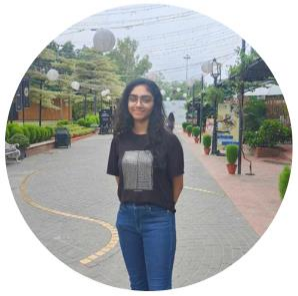
Gargie Tambe



Kritin Madireddy

Shaunak Biswas

Tejas Cavale



Vyakhya Gupta

Distribution of Grades

Component	Weightage
Final Exam	25%
Mid-Sem Exam	15%
Take home/In-class activities	15%
3 Unit Projects (3*15)	45%
Bonus	5%

Note: The instructor reserve the right to make minor changes
No extension requests please – **Extra days and soft deadlines!!**

Course Logistics

- Course announcements, management - moodle
- All resources, information and materials - https://karthikv1392.github.io/cs6401_se/
- At any point, feel free to contact either the instructor or TA
- Instructor office hours (Friday 11:00 AM to 12:00 PM)
- Feedbacks are always welcome!!

Thank You

Course website: karthikv1392.github.io/cs6401_se

Email: karthik.vaidhyanathan@iiit.ac.in
Web: https://karthikvaidhyanathan.com