

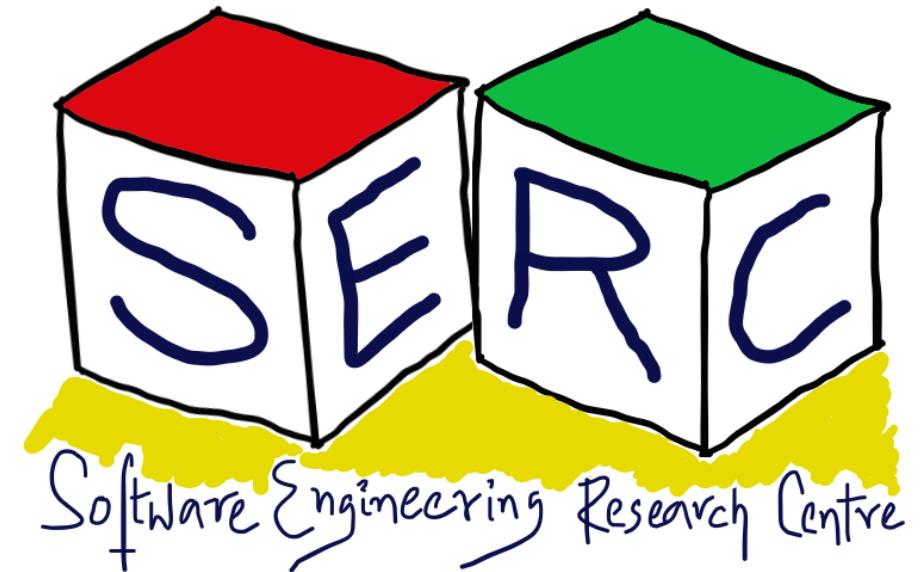
# Concluding Thoughts

**CS6.401 Software Engineering**

Dr. Karthik Vaidhyanathan

[karthik.vaidhyanathan@iiit.ac.in](mailto:karthik.vaidhyanathan@iiit.ac.in)

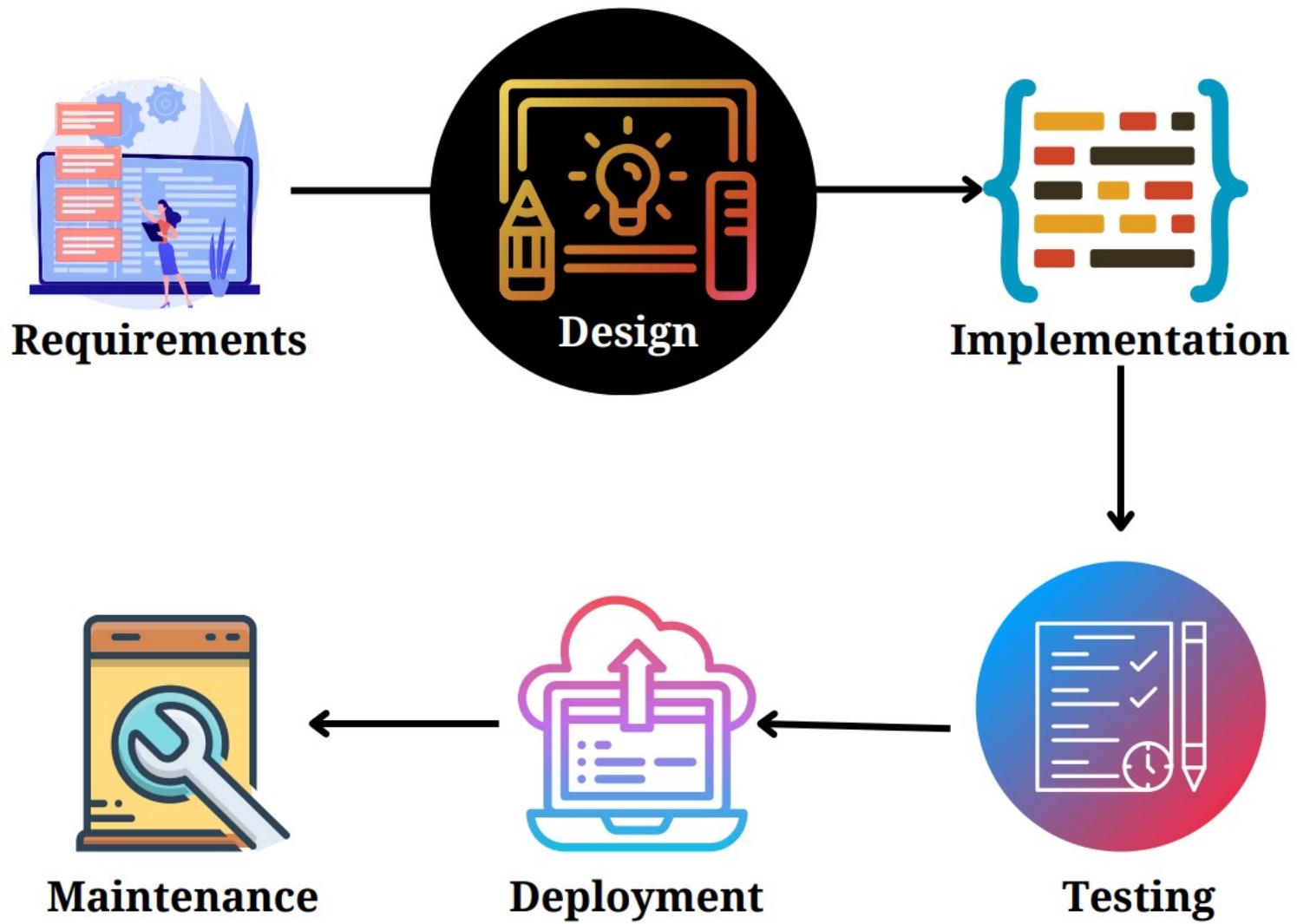
<https://karthikvaidhyanathan.com>



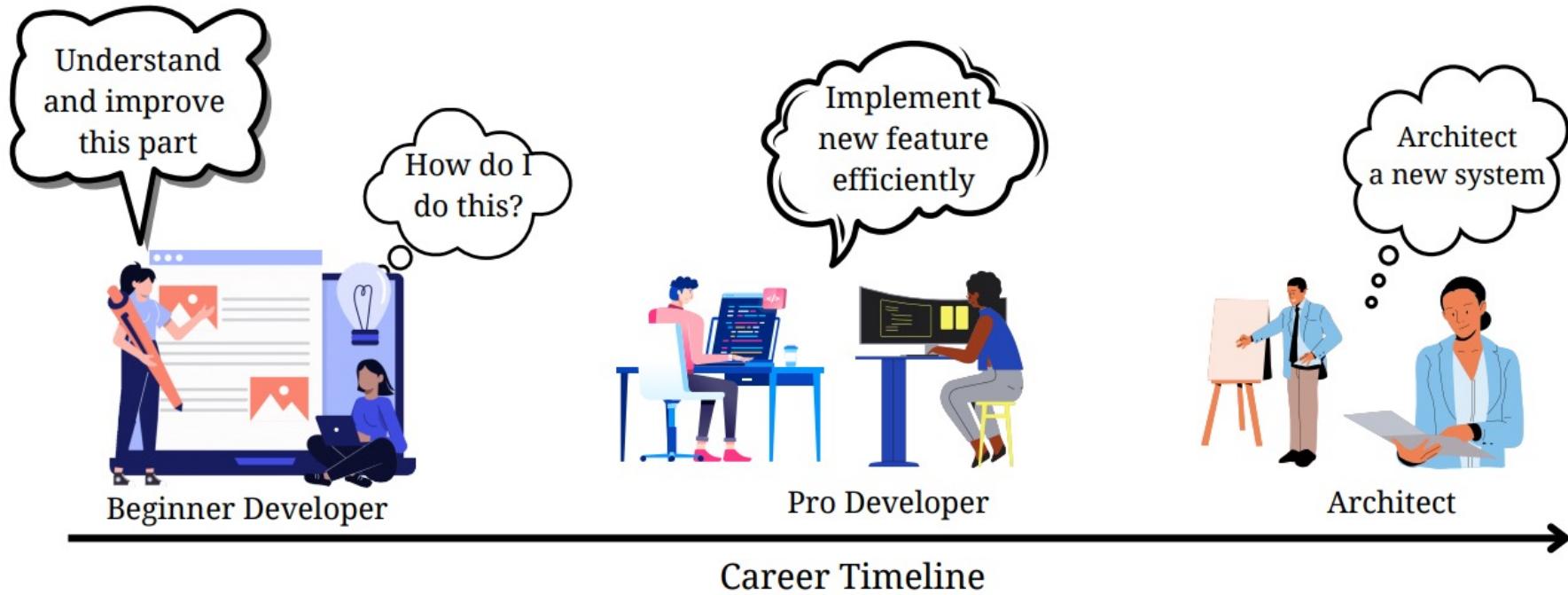


Software Engineering!

# Software Development Lifecycle



# What we learned so far?



## This Course

Modeling

Design Principles

Architectural Framework

Refactoring

Design Patterns

Architectural Patterns

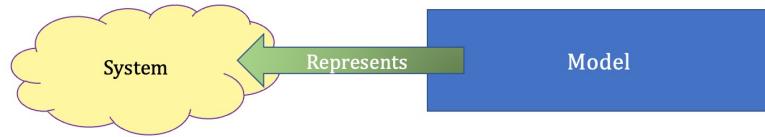


# Quick Recap

# Modeling and Refactoring

## So what is a software model?

A simplified or partial representation of a real system, defined in order to accomplish a task or to reach an agreement.

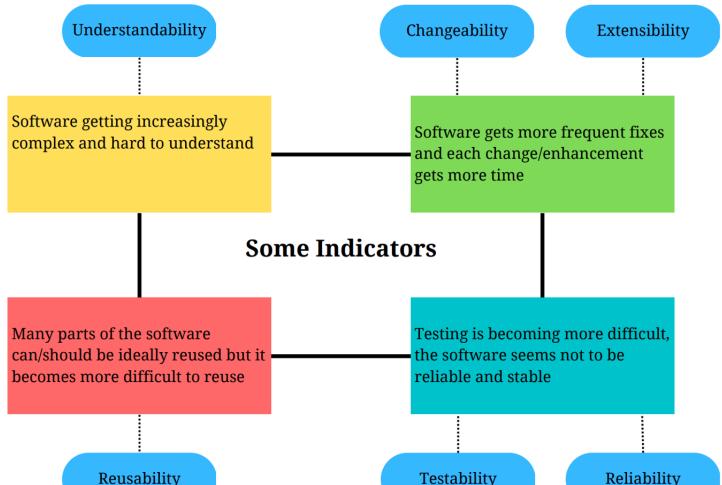


**Mapping:** A model is always a mapping of some real system

**Reduction:** A model reflects only relevant set of properties of original system

**Pragmatism:** A model needs to be usable in place of the actual system with respect to some purpose

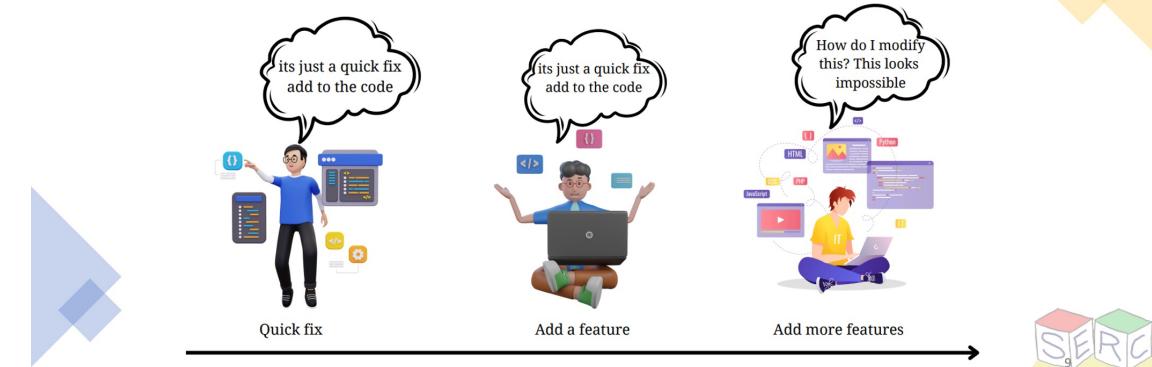
## Software Quality as an Indicator



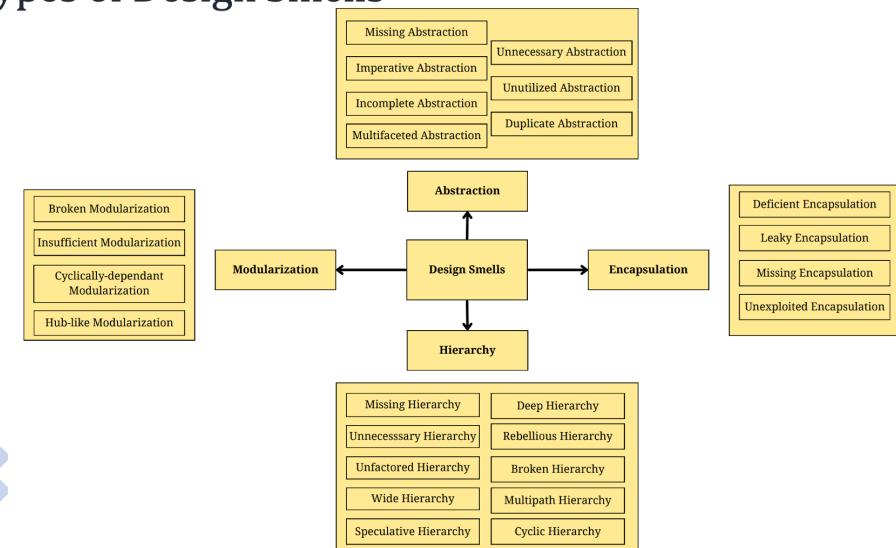
## Technical Debt - Definition

Technical debt is the **debt that accrues** when you knowingly or unknowingly make **wrong or non-optimal design decisions**

Metaphor coined by *Ward Cunningham*, 1992



## Types of Design Smells



# Design Principles and Design Patterns

## Design Patterns

Each Pattern describes a problem which **occurs over and over again** in our **environment** and then **describes the core of the solution** to that problem, in such a way that you can **use this solution a million times over**, without ever doing it the same way twice

-- Christopher Alexander

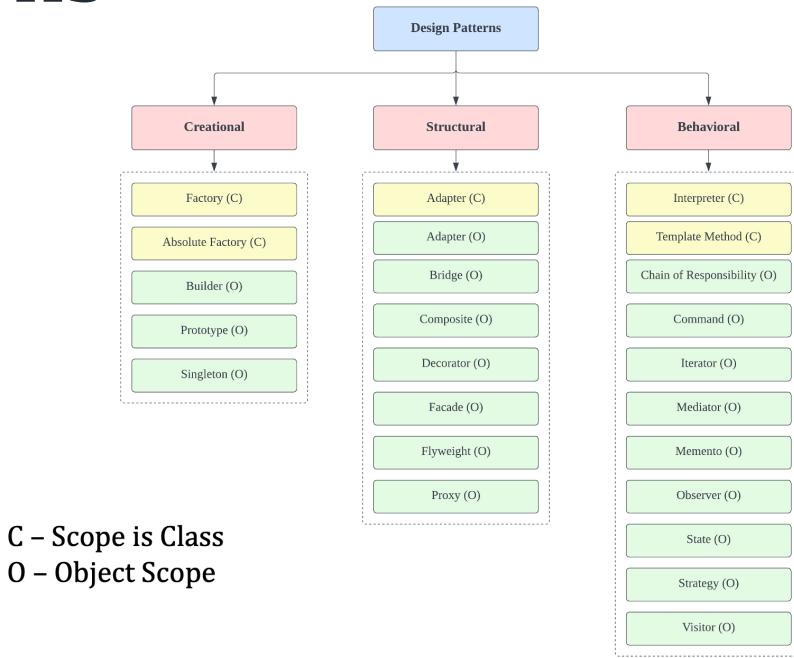
Patterns captures {Context, Problem, Solution}

What are some of the patterns you can think of?



## Program to Interface Not Implementation

- One of the most important OO Design Principles
- “Program to interface” refers to the idea of ensuring loose coupling
  - Does not only mean the “Interface”?
- Very useful when lot of changes are expected
- Create an interface, define methods -> create classes that implements them
- Allows external objects to easily communicate
- Maintainability and flexibility increases



C – Scope is Class  
O – Object Scope

## Favor Object Composition over Class Inheritance

- Two most common techniques: Inheritance and Composition
- Class inheritance: White-box reuse
  - Internals of parent class are visible to child class
  - Defined statically at compile time
  - Sub class can override methods of parent class
- Inheritance is not always the go to solution - "breaks encapsulation"
- Composition: Black-box reuse
  - Objects acquiring references to other objects
  - Defined dynamically at run time
  - Encapsulation is not broken – Objects are accessed through interfaces
  - Get what is needed by assembling and not by creating



# Software Architecture Framework and Patterns

## Software Architecture Definitions

- Garlan and Shaw, '93:  
*Architecture* for a specific system may be captured as "a collection of computational **components** - or simply components - together with a description of the interactions between these components - the **connectors**"
- Bass et al.:  
"The software architecture of a program or computing system is the **structure or structures** of the system, which comprise **software elements**, the **externally visible properties** of those elements, and the **relationships** between them."

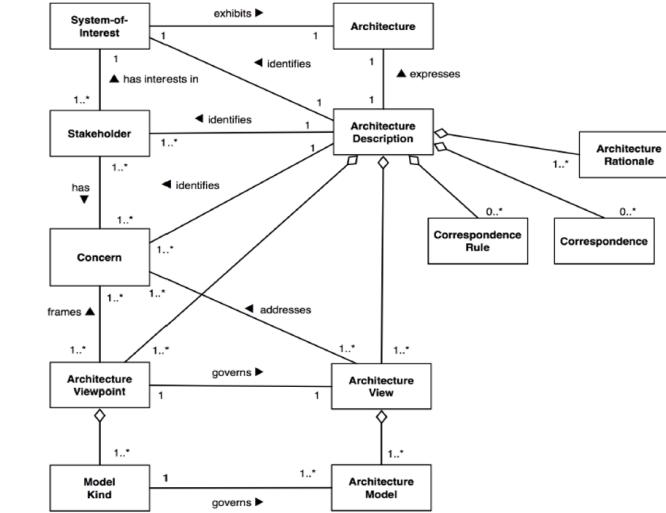


## What about Architectural Tactics?

*"Characterization of architectural decisions that are used to achieve a desired quality attribute response"*



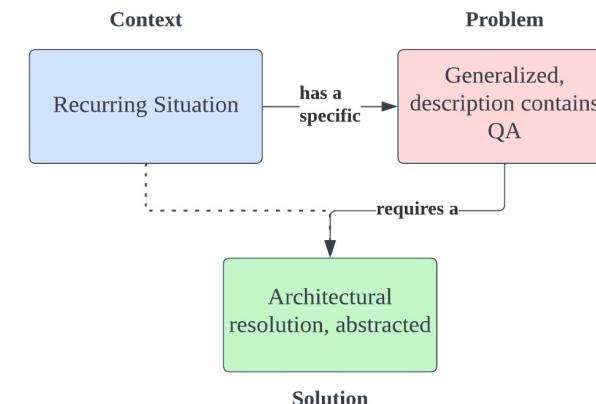
## Architecture Description



ISO/IEC/IEEE 42010, Systems and Software Engineering – Architecture Description



## Architectural Patterns



Pattern documentation template: {context, problem, solution}



# Research Avenues

# Active Research Areas

Software Testing

Software  
Architecture

Software  
Maintenance and  
Evolution

Model Driven  
Engineering

Software Verification

Software  
Sustainability

Software Processes

Human Aspects

Many More ...  
(RE, PL,...)



# Research Areas in SERC



**Virtual Labs**



**Gamification**



**VR and AR**



**IoT**



**HCI**



**Software Quality**



**Programming Languages**



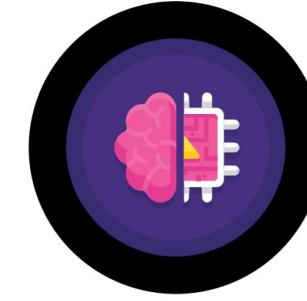
**Formal Methods**



**Software Analytics**



**Self-adaptive Systems**



**SE and ML**



**Software Sustainability**

# Some Conferences



FSE 2024

ASE 2024



MSR 2025



27<sup>th</sup> International Conference on  
Model Driven Engineering Languages and Systems

MODELS 24

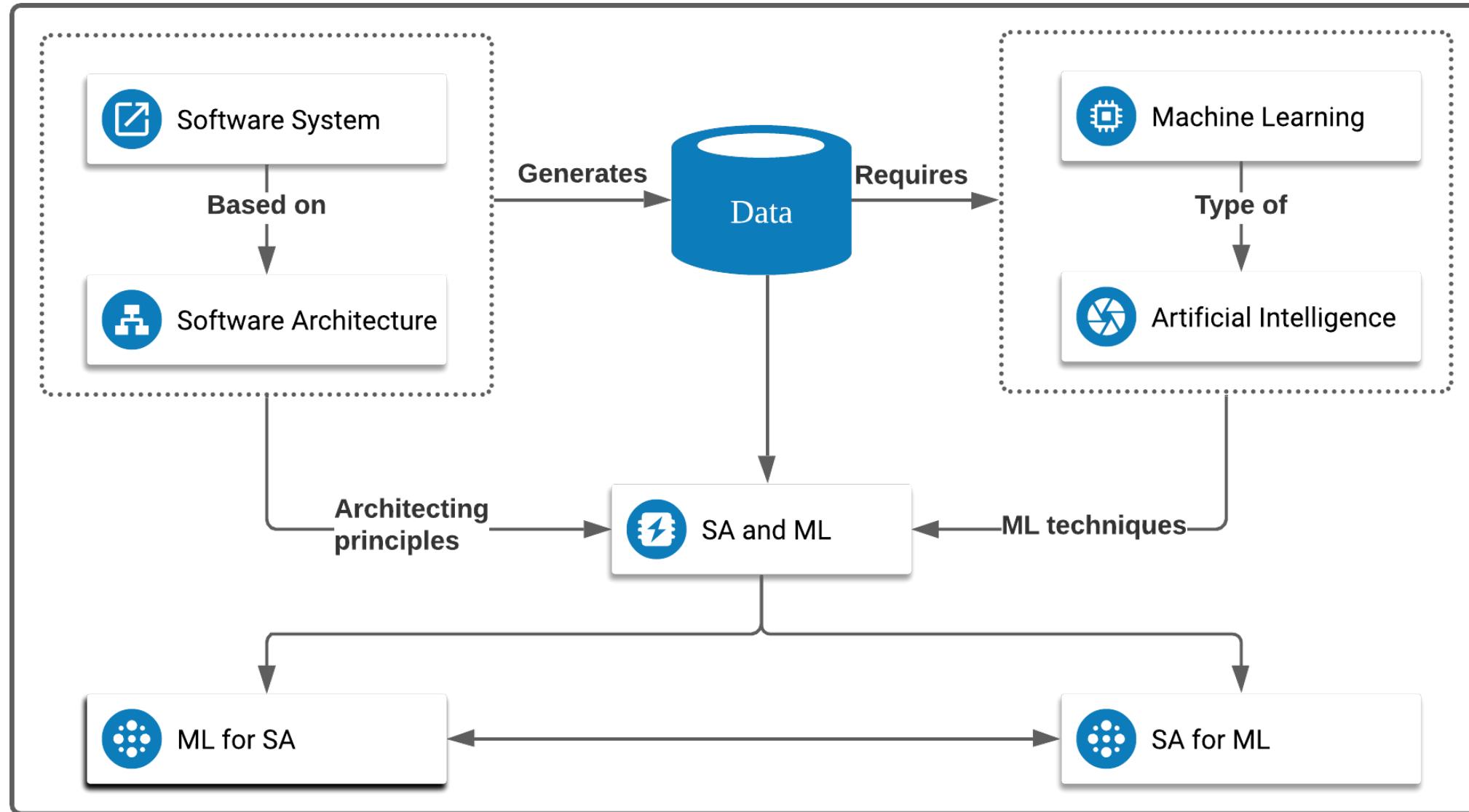
22 - 27 September 2024  
Linz , Austria

**ECSA 2024**  
Mon 2 - Fri 6 September 2024  
Parc Hotel Alvisse, Luxembourg, Luxembourg

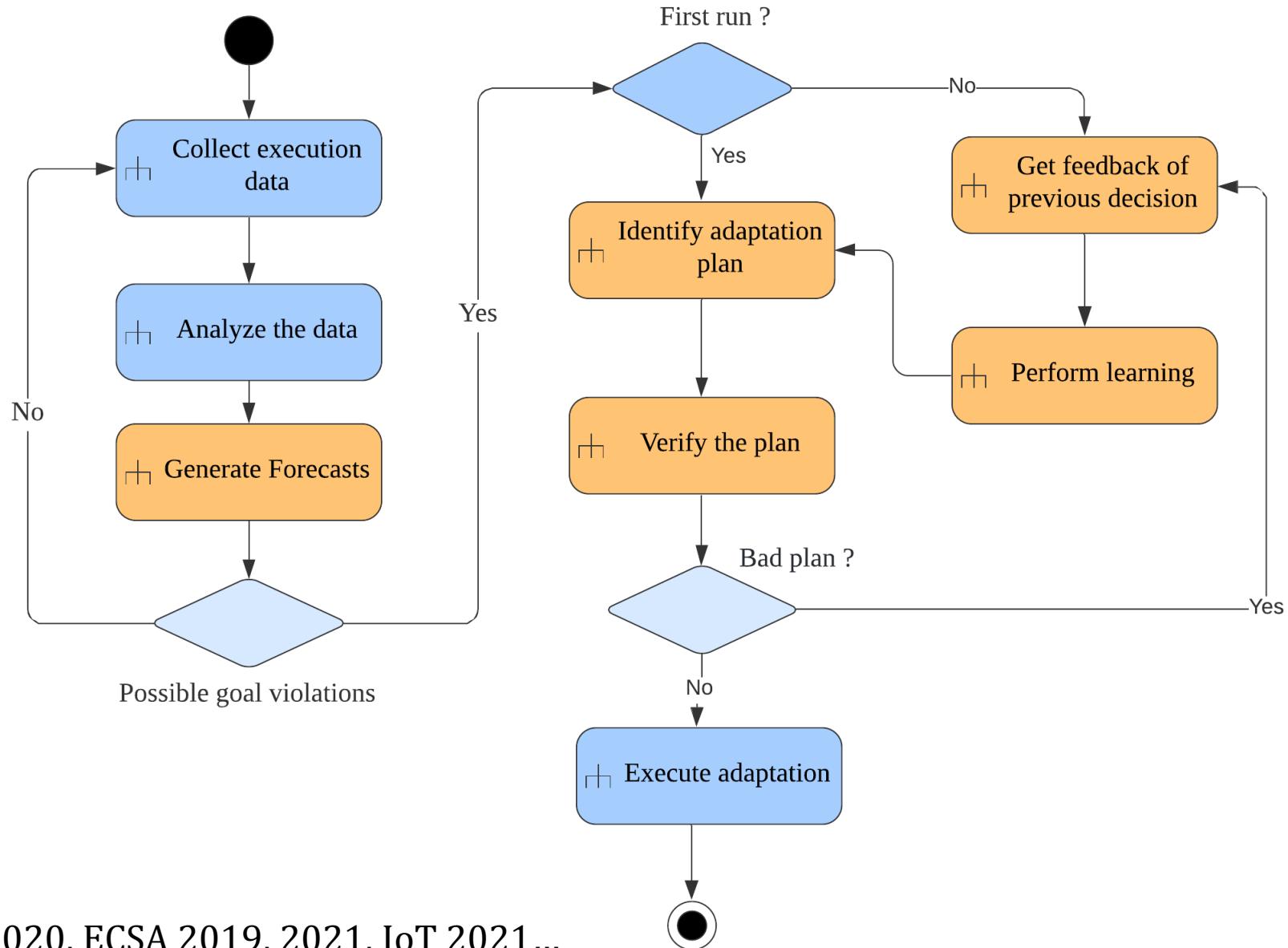
EASE 2024  
Evaluation and Assessment in Software Engineering  
SALERNO.ITALY



# Intersection of SA and ML



# Learning Driven Adaptation



# SA4ML: Self-adaptive ML and SWITCH

## Towards Self-Adaptive Machine Learning-Enabled Systems Through QoS-Aware Model Switching

Shubham Kulkarni, Arya Marda, Karthik Vaidhyanathan

Software Engineering Research Center, IIIT Hyderabad, India

shubham.kulkarni@research.iiit.ac.in, arya.marda@students.iiit.ac.in, karthik.vaidhyanathan@iiit.ac.in

**Abstract**—Machine Learning (ML), particularly deep learning, has seen vast advancements, leading to the rise of Machine Learning-Enabled Systems (MLS). However, numerous software engineering challenges persist in propelling these MLS into production, largely due to various run-time uncertainties that impact the overall Quality of Service (QoS). These uncertainties emanate from ML models, software components, and environmental factors. Self-adaptation techniques present potential in managing run-time uncertainties, but their application in MLS remains largely unexplored. As a solution, we propose the concept of a Machine Learning Model Balancer, focusing on managing uncertainties related to ML models by using multiple models. Subsequently, we introduce AdaMLS, a novel self-adaptation approach that leverages this concept and extends the traditional MAPE-K loop for continuous MLS adaptation. AdaMLS employs lightweight unsupervised learning for dynamic model switching, thereby ensuring consistent QoS. Through a self-adaptive object detection system prototype, we demonstrate AdaMLS's effectiveness in balancing system and model performance. Preliminary results suggest AdaMLS surpasses naive and single state-of-the-art models in QoS guarantees, heralding the advancement towards self-adaptive MLS with optimal QoS in dynamic environments.

**Index Terms**—Self Adaptation, Self-adaptive systems, Software Architecture, ML-Enabled Systems, ML4SA, Unsupervised Learning, Object Detection

velopers can devise a spectrum of models, each with its speed and accuracy trade-offs. Recognizing this variability, we introduce the concept of an ML Model Balancer. This notion encapsulates the idea of dynamically evaluating and switching between models to optimize QoS. For instance, high-traffic situations might favor a faster model, while quieter periods prioritize accuracy. AdaMLS, our novel self-adaptive approach, operationalizes this concept of the ML Model Balancer. Nevertheless, AdaMLS consistently excels in navigating the intricacies of online ML deployments, ensuring superior QoS. This includes: i) monitoring model and system parameters; ii) analyzing model and system quality for QoS violations; iii) using knowledge from lightweight unsupervised learning to dynamically switch models, ensuring QoS; and iv) executing system adaptation. Prioritizing ML model adaptability, AdaMLS shifts from conventional load balancing to QoS-aware dynamic ML model switching. By continuously tuning model selections in response to environmental cues and system demands, AdaMLS guarantees MLS QoS, promoting consistent MLS operation in live settings. This represents a stride towards future-ready self-adaptive MLS, designed to

## SWITCH

### An Exemplar for Evaluating Self-Adaptive ML-Enabled Systems

#### Description

Addressing runtime uncertainties in Machine Learning-Enabled Systems (MLS) is crucial for maintaining Quality of Service (QoS). The Machine Learning Model Balancer is a concept that addresses these uncertainties by facilitating dynamic ML model switching, showing promise in improving QoS in MLS. Leveraging this concept, this paper introduces SWITCH, an exemplar developed to enhance self-adaptive capabilities in such systems through dynamic model switching in runtime. SWITCH is designed as a comprehensive web service, catering to a broad range of ML scenarios, with its implementation demonstrated through an object detection use case. SWITCH provides researchers a flexible platform to apply and evaluate their ML model switching strategies, aiming to enhance QoS in MLS. SWITCH features advanced input handling, real-time data processing, and logging for adaptation metrics. With its interactive realtime dashboard, SWITCH offers researchers a user-friendly interface for experiment management and system observability for MLS. This paper details SWITCH's architecture, self-adaptation strategies through ML model switching, and its empirical validation through case study, illustrating its potential to improve QoS in MLS. By enabling a hands-on approach to study adaptive behaviors in ML systems, SWITCH contributes a valuable tool to the SEAMS community for research into self-adaptive mechanisms and their practical applications

<https://tool-switch.github.io/>

SEAMS 2024@ICSE 2024



# ML4SA: GenAI for Architecture Knowledge Management

## 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

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

# What Next!

1. Topics in Software Engineering Course
2. PhD/Software researcher/MS in Software Engineering  
(Erasmus, SE4GD, EDISS, ...)
3. SDE/Data scientist/ML Engineer/Data engineer....
4. Junior architect/software architect/consultant...
5. Research Software Engineer



# Course Logistics Management

# Thanks to the Wonderful Team!



Ankith



Ayush



Mayush



Sai Venu



Seshadri



Shubham



Siddharth



*“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





Feedback/comments?

# Best Wishes!!

## Thank You



Course website: [karthikv1392.github.io/cs6401\\_se](https://karthikvaidhyanathan.github.io/cs6401_se)

Email: [karthik.vaidhyanathan@iiit.ac.in](mailto:karthik.vaidhyanathan@iiit.ac.in)

Web: <https://karthikvaidhyanathan.com>

Twitter: @karthi\_ishere