# Topic 1:

## Introduction to the Abstraction and SA discipline

**Software Engineering def:** Set of **automated** methods to **systematically** develop **quality** software that fulfils **customer needs** while satisfying **existing constraints**

**Software Architecture is about**: DESIGN, ABSTRACTION, QUALITY

## Q&A

- **S:** If a company uses SCRUM management, does the SCRUM master also function as a Software Architect?
    - **P:** Not necessarily. The work of a SCRUM master is to prioritize the right tasks and smartly organize meetings. Their duties do not overlap entirely with the ones of a Software Architect.
- **S:** What is the difference between Software Architecture and S.A. Style?
    - **P:** A Software Architect is free to define his own architecture as he pleases, without any constraints.

      However, if he or she has to follow a certain style's guidelines, it will certainly place limitations in his design. For instance, in a Client-Server architecture, you will never see a server sending a request to a client. In a sense, that is a limitation imposed by that S.A. Style.

      When you choose to pursue one style, you must obey its rules.

      Then, you could ask, why would we ever consider using a style?

      That is because constraints can be a good thing. Styles impose constraints in order to ensure quality. Different styles can also be combined.
- **S:** In the first part of the podcast, they state that in an ideal company the S.A. would be the person in charge of speaking with clients, collecting non-functional requirements and even choosing the right people for the job. In that context, what's the difference between a S.A. and a Project Manager?
    - **P:** The Project manager has the main duty of organizing time and schedule processes, and that job alone is so time consuming that he probably won't be able to do anything else. It's similar to the SCRUM master situation. The P.M. interacts mainly with the team, while the S.A. has to relate to many different people, including the client, the team, and any external factors or organizations.
- S: Software Architects are often compared to actual architects. But how can that comparison hold when the architect has to deal with so many legal issues, and can potentially even go to jail?
    - P: A S.A. can also go to jail.

      Most of the books on Software Architecture also compare software to a building or LEGO constructions.

      In terms of responsibilities, there are software systems impacting actual lives. Someone has to hold liability, which is responsibility relating to law.

I'm not really sure who would be the one held accountable, but surely there will be someone.

- **S:** What about agile development? It's hard to keep track of  everything, so who could be held accountable in such a chaotic context?
  - **P**: In recent years, focus shifted on how to make Agile development more about quality than speed.

    Back then, Agile development was entirely about speed, but redeveloping modules and software so often was way too expensive.

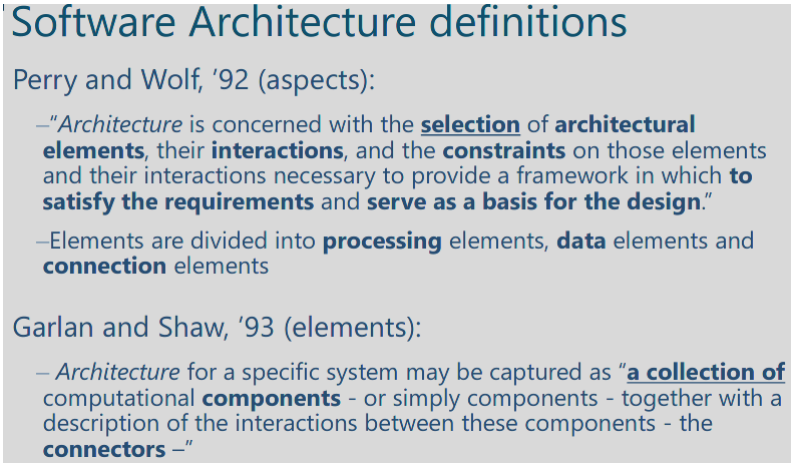
# Topic 2

## Software Architecture as: a set of components and connectors

**Another Software Engineering def:**

The Software Architecture is the **earliest model** of the **whole software system** created along the software lifecycle.

**Yet another SA def:**




### Components

A component is a building block that is :

- a unit of computation or a data store, with an interface specifying the services it provides and requires
- a unit of deployment
- a unit of reuse (e.g., client, server, database, filters, ...)

A component is (or should be) i**ndependent of the context** in which it is used to provide services

### Connectors

A connector is a building block that enables interaction among components

- Events
- Client/server middleware
- Messages and message buses
- Shared variables
- Procedure calls (local or remote)

- Pipes

Connectors may be implicit or explicit

- Connectors sometimes are just channels
- Connectors sometimes have their own logic and complexity

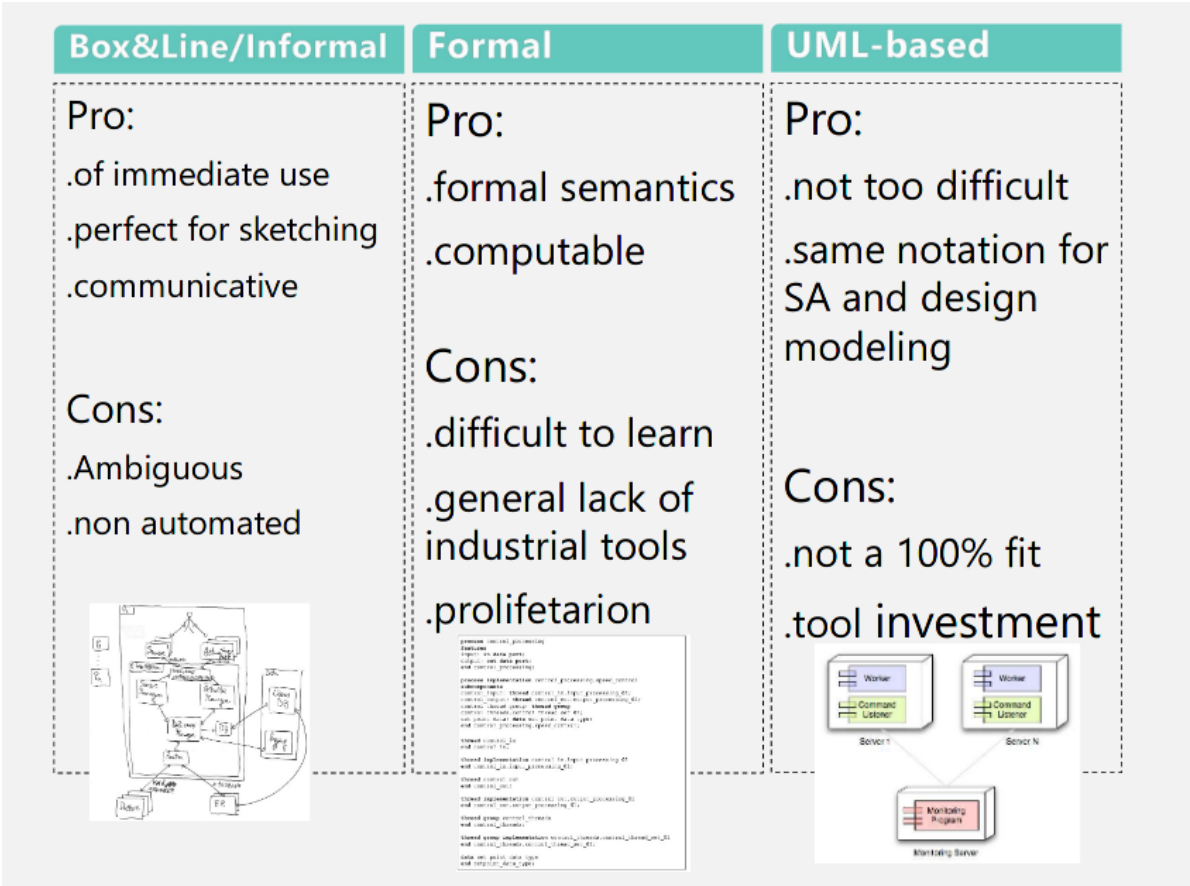
A connectoris (or should be) **dependent** on the context in which it is used to connect components

> Connectors sometimes are modeled as special kinds of components

## Interfaces

An interfaceis the external connectionof a component (or connector) that describes how to interact with it

## Components and Connectors specification



# GROUP DECISION MAKING.

- MULTIPLE PEOPLE TAKING DECISION
- MULTIPLE STAKEHOLDERS WITH DIFFERENT CONCERNS

MAIN FACTORS BY GGPS

GROUP STRUCTURE
DECISION CHARACTERISTICS
DECISION MAKING CONTEXT
→ EMERGENT GROUP CHARACTERISTIC
→ GROUP PERCEPTIONS
→ GROUP PROCESS
→ DECISION PROCESS CHARACTERISTICS
→ OUTCOME

RQ1 (GDM PRATICES IN INDUSTRY)

1: 3-5 PEOPLE INVOLVED IN ARCHITECTURE DECISION MAKING.

2: TEAM ARE HOMOGENEUS → COULD LEAD TO GROUPTHINK

3: DISCUSSION BASED AND DEMOCRATIC

GROUPTHINK → TENDENCY OF GROUP TO TRY TO MINIMIZE CONFLICT AND REACH CONSENSUS WITHOUT SUFFICIENT TESTING.

4: DISTRIBUITED TEAMS

5: CONSENSUS IS MOSTLY DISCUSSION BASED

GDM PRACTICE VS GDM THEORY

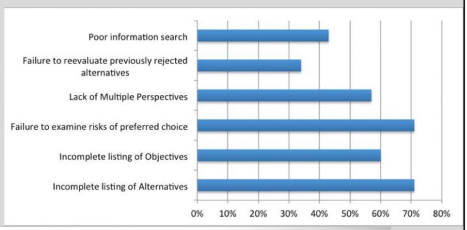
1: COMBINE DIFFERENT GDM METHODS

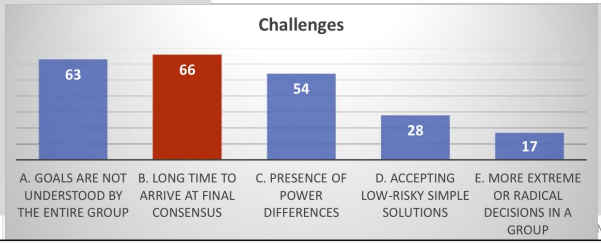
2: DIFFERENT STAKEHOLDERS ARE GIVEN DIFFERENT WEIGHTS

GDM METHODS:

BRAINSTORMING, VOTING, DELPHI (EXPERT ANSWER QUESTIONNAIRES IN A DISTRIBUITE AND ANONYMOUS WAY) CONSENSUS SELECTION (ALTERNATIVES ARE LISTED SELECT THE MOST AGREED), AHP (PROBLEM MODELED AS GOALS)

1. GMD issues




May 2019)

HOW TO MAKE GDM EFFICIENT?

# DESIGN DECISION

## SA PROSPECTIVE:

- A SET OF COMPONENT AND CONNECTORS
- WRITTEN ACCORDING TO ARCHITECTURAL STYLES
- FOCUS ON VIEW AND VIEWPOINTS
- A SET OF ARCHITECTURE' DESIGN DECISION

**SOFTWARE ARCHITECTURE CONSIST OF:**

- A BLUEPRINT FOR CHOSEN SOLUTION (PRODUCT)
- A SET OF DESIGN DECISION

**DESIGN AS A SERIES OF DECISIONS**

DESIGNER IS FACED WITH A SERIES OF DESIGN ISSUES

- THESE ARE SUBPROBLEM OF THE OVERALL DECISION PROBLEM
- EACH ISSUES HAS ALTERNATIVE SOLUTION
- DESIGNER MAKE DESIGN DECISION TO SOLVE EACH ISSUES

**DIFFERENT SET OF ADD BRINGS POTENTIALLY DIFFERENT ARCHITECTURES, HOW TO CHOSE?**

DESIGN DECISION PROCESS:

- Collection of Requirements and Constraints
- Identification of Design Issues
- Identification of Design Alternatives
- Identification of Design Decisions (DDs)
- Selection of architectural Components
- Selection of an architectural Solutions that comply to the DDs

TAKING DECISION IN BASE:

- CURRENT KNOWLEDGE
- PAST KNOWLEDGE

ADD: WHAT IS INTERESTING TO DISCUSS?

1. GRANULARITY OF DD
2. DEPENDENCIES AMONG DECISION
3. ADD TAKE IN COLLABORATIVE WAY
4. EVOLVING ADD

## SA STYLES

A SET OF DESIGN RULES THAT IDENTIFY THE KINDS OF COMPONENTS AND CONNECTORS THAT MAY BE USED, TOGETHER WITH LOCAL OR GLOBAL CONSTRAINTS

ARCHITECTURAL STYLES DETERMINE FOUR KIND OF PROPERTIES

1 VOCABULARY OF DESIGN ELEMENTS - COMPONENT AND CONNECTORS TYPE

2 SET OF CONFIGURATION RULES (TOPOLOGICAL CONSTRAINTS)

3 SEMANTIC INTERPRETATION

4 ANALYSES THAT CAN BE PERFORMED ON SYSTEMS BUILT IN THAT STYLE.

## DESIGN PATTERN VS STYLE

PATTERN: MORE DOMAIN SPECIFIC AND RECENT

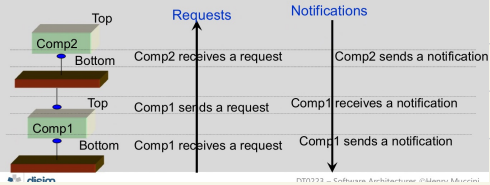
STYLES: MORE GENERIC

## The C2 style Composition Rules

1. The top of a component may be connected to the bottom of a single connector.
2. The bottom of a component may be connected to the top of a single connector.
3. There is no bound on the number of components or connectors that may be attached to a single connector.
4. When two connectors are attached to each other, it must be from the bottom of one to the top of the other.
5. Components can communicate only through connectors

## The C2 style Communication Rules

The communication between components and connectors is achieved solely exchanging messages
The communication is based on notifications and requests
Both component top domain and bottom domain can notify or request messages




## LAYERED STYLE

A layered system is organized hierarchically, each layer providing service to the layer above and below

Components
- Programs or subprograms deployed in a layer

Connectors
- Protocols
  - Procedure calls or system calls

VANTAGGI:

DECOMPOSABILITY, MAINTENABILITY ADAPTABILITY, UNDERSTENDABILITY

SVANTAGGI:

NOT FOR ALL SYSTEMS, + LAYERS - PERFORMANCE

# BLACKBOARD STYLE

**This style is characterized by a central data structure and a collection of components operating on the central data store**

The Blackboard is characterized by three main types of components (Corkill, 1991):

- the knowledge sources (KS), which are independent modules that contain the knowledge needed to solve the problem,
- the blackboard, which is a global repository containing input data and partial solutions, and
- the controller, which is responsible for managing the course of problem solving (e.g. coordinating the different KS).

Connector

– Blackboard: shared data repository, possibly with finite capacity

Property

– different KSs may not communicate directly

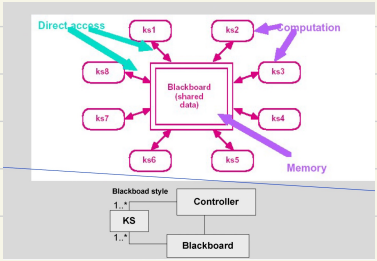
ADVANTAGES

SIMPLICITY (ONLY 1 CONNECTOR),

EVOLVABILITY (EASY ADD COMPON.)

DISADVANTAGES

BLACKBOARD BECOMES A BOTTLENECK WITH MORE CLIENT




# DISTRIBUTED P2P

Components

– Independently developed objects and programs offering public operations or services

Connectors

– Remote procedure call (RPC) over computer networks

Configurations

# CLIENT-SERVER

Client/Server systems are the most common specialization (restriction) of the peer-to-peer style

One component is a server offering a service

The other components are clients using the service

=> VANTAGGI:

INTEROPERABILITY, REUSABILITY, SCALABILITY, DISTRIBUTABILITY

DISADVANTAGES:

MAINTANABILITY

# MVC

**Model**

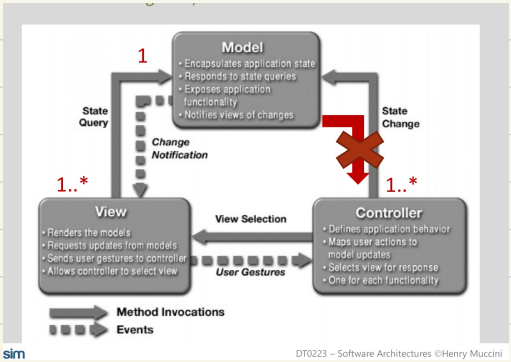
– The model represents enterprise data and the business rules that govern access to and updates of this data.
– This is the domain-specific representation of the information on which the application operates.
– It maps requests coming from clients into operations over the model

**View**

– The view renders the contents of a model. It accesses enterprise data through the model and specifies how that data should be presented.
– It is the view's responsibility to maintain consistency in its presentation when the model changes.
– MVC is often seen in web applications, where the view is the HTML page and the code which gathers dynamic data for the page.

**Controller**

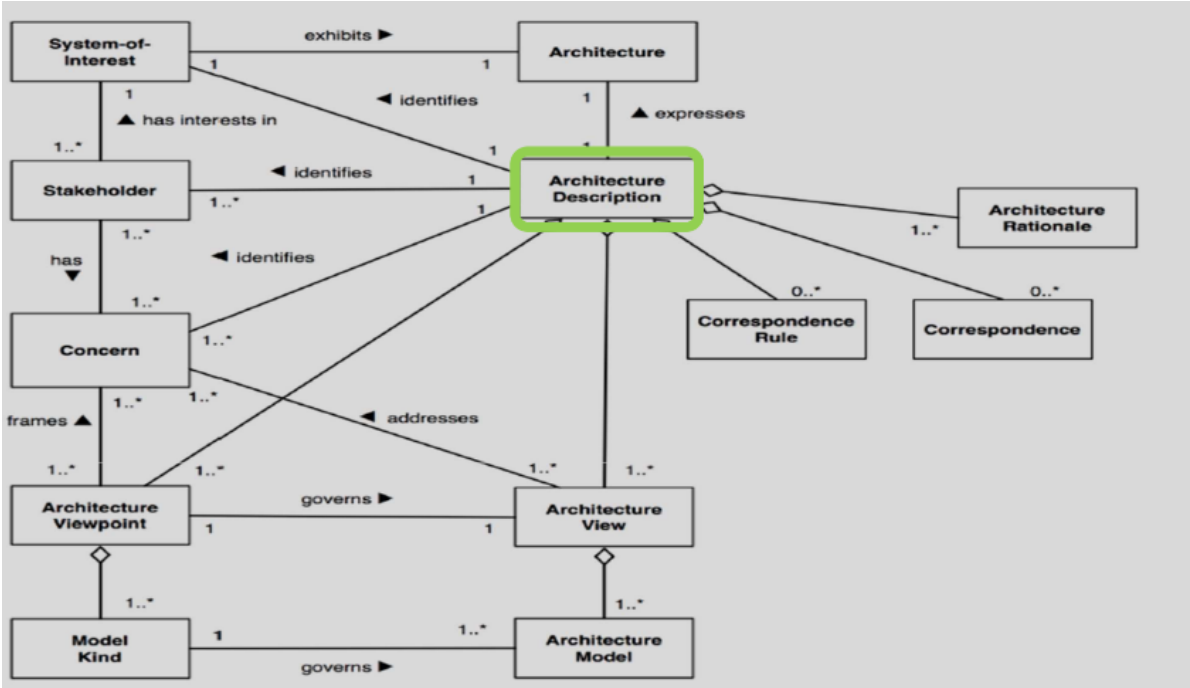
– The controller translates interactions with the view into actions to be performed by the model.
– This responds to events, typically user actions, and invokes changes on the model and perhaps the view.
– In a stand-alone GUI client, user interactions could be button clicks or menu selections, whereas in a Web application, they appear as GET and POST HTTP requests. The actions performed by the model include activating business processes or changing the state of the model. Based on the user interactions and the outcome of the model actions, the controller responds by selecting an appropriate view.



# Topic 6

## Software Architecture as: Focus on set of views and viewpoints

### ISO/IEC/IEEE 42010 standard




## Architecture Description

Architecture Description is the practice of expressing architectures (ISO/IEC 42010)

The practices of recording software, system and enterprise architectures so that architectures can be understood, documented, analysed and realized

It has 3 mechanism:

1. **Architecture Viewpoints:**
   define the contents of each architecture view;
2. **Architecture Frameworks(AFs):**
   coordinated set of viewpoints for use within a particular stakeholder community or domain of application;
3. **Architecture DescriptionLanguages(ADLs):**
   any mode of expression used in an architecture description. ADL provides model kinds selected to frame one or more concerns

## View and viewpoint definition:

> - View : Viewpoint = Program : Programming Language
> - a viewpoint is a way of looking at a system; a view is what you see”
> - viewpoint refers to the conventions for representing an architecture relative to one set of concerns

- A **viewpoint** defines the conventions (such as notations, languages and types of models) for constructing a certain kind of view
- A **view** is the result of applying a viewpoint to a particular system of interest (Using multiple views has become standard practice in industry)

# Slide 7-8-9 / SA

## Slide 7 - Setup Docker Kubernetes

- Installazione software
- Creazione di un'applicazione web ASP.NET core con supporto Docker tramite Visual Studio Code
- Containerizzazione del software, tramite un Dockerfile nella root folder dell'applicazione
  - Buildare l'immagine
  - Verificare che sia disponibile nel repository locale di Docker
- Inviare l'immagine docker sul Docker hub per renderla pubblica
  - Creare account Docker
  - Creare repository pubblica su Docker hub
  - Pushare l'immagine
- Deploy del container su Kubernetes
  - Creare un file di deployment
  - Fare il deploy dell'immagine
  - Esporre un tipo di servizio LoadBalancer per esporre più pod

## Slide 8 - Architecting Fault Tolerance Systems

**Obiettivo** → Identificare, classificare e comparare le tecniche esistenti sviluppate per architettare sistemi Fault Tolerance

**Ambito** → tutte quelle tecniche che permettono l'integrazione di concetti di fault tolerance in specifiche di software architecture:

- Exception Handling
- Error Detection
- Atomic Actions
- Run-time reconfiguration

Poco dopo c'è un esempio di un tizio che si sta a morì che è diabetico, ma non trovo nulla di utile.

# Dependability

La Dependability di un sistema computazionale è l'abilità di consegnare un servizio che può essere giustificatamente affidabile.

**Dependability attributes**:

- Availability → disponibilità per un servizio corretto
- Reliability → continuità di un servizio corretto
- Safety → assenza di conseguenze catastrofiche per l'utente o per l'ambiente
- Confidentiality → assenza di fughe di informazioni non autorizzate

## Significati della Dependability

Si devono combinare quattro tecniche per sviluppatore un sistema computazionale affidabile

- Fault Prevention
- Fault removal
- Fault tolerance
- Fault forecasting

# Concetti fondamentali

- System failure → avviene quando il servizio consegnato fa altro rispetto a ciò che dovrebbe fare

- Errore → parte dello stato del sistema che è responsabile di fallimenti sequenziali
- Fault → Causa di un errore

## Step

1. Error detection
2. Error handling
    1. Backward error recovery
    2. Forward error recovery
    3. Compensation
3. Fault handling

## Failure Masking

Tecnica di fault toleranze che nasconde le occorrenze di failures dagli altri processi. L'approccio più comune è la ridondanza.

Ce ne sono tre tipi:

1. Information redundancy
2. Time redundancy
3. Physical redundancy

## Strutturazione del sistema

I componenti dovrebbero essere definiti dentro unità di fault-tolerant, incapsulare un recovery interno dentro l'unità, e separano il comportamento normale da quello anormale. Inoltre definiscono le regole di come le attività normali e anormali siano relazionate

# Slide 9 - Publish Subscribe Kafka

## Publish-Subscribe

I subscribers si registrano/deregistrano per ricevere contenuti specifici, e i publisher inviano in broadcast messaggi ai subscriber (sync o async).

Gli **elementi di design** sono i componenti, i connettori e i dati, mentre le **topologie** sono: o i subscriber direttamente connessi ai publisher o via intermediari.

I **vantaggi** sono che i subscriber sono indipendenti e c'è una disseminazione delle informazioni one-way molto efficienti, mentre gli **svantaggi** sono che ci vogliono protocolli speciali con molti subscriber.

Un problema può essere che viene aggiunta della complessità alla sorgente, poiché influisce su scalabilità e velocità e ci può essere una perdita di informazioni.

## Apache Kafka

**Kafka** è un sistema di messaging distribuito publish-subscribe. Lavora mantenendo i feed di messaggi in topic. È molto utile per creare applicazioni real-time. Viene eseguito come cluster di server.

I messaggi vengono memorizzati tramite "producers", e i dati possono essere partizionati in partizioni diverse dentro topic diversi. Gli altri processi sono chiamati "consumer" e possono fare query sui messaggi delle partizioni.

Un **topic** è una categoria o un nome di un feed dove i record sono pubblicati. I topic sono sempre multi-subscriber.

## Event-Driven Architectures

I componenti indipendenti asincronicamente emettono e ricevono eventi sugli event buses. Producono, rilevano e consumano eventi e sono componenti con meno coupling possibile.

Gli **elementi di design** sono i componenti, i connettori e i dati. La **topologia** è la comunicazione tra event bus o solo link.

La **Mediator Topology** è simile all'orchestrazione nei SOA tradizionali, mentre la **Broker Topology** è simile alla Choreography nei SOA tradizionali.

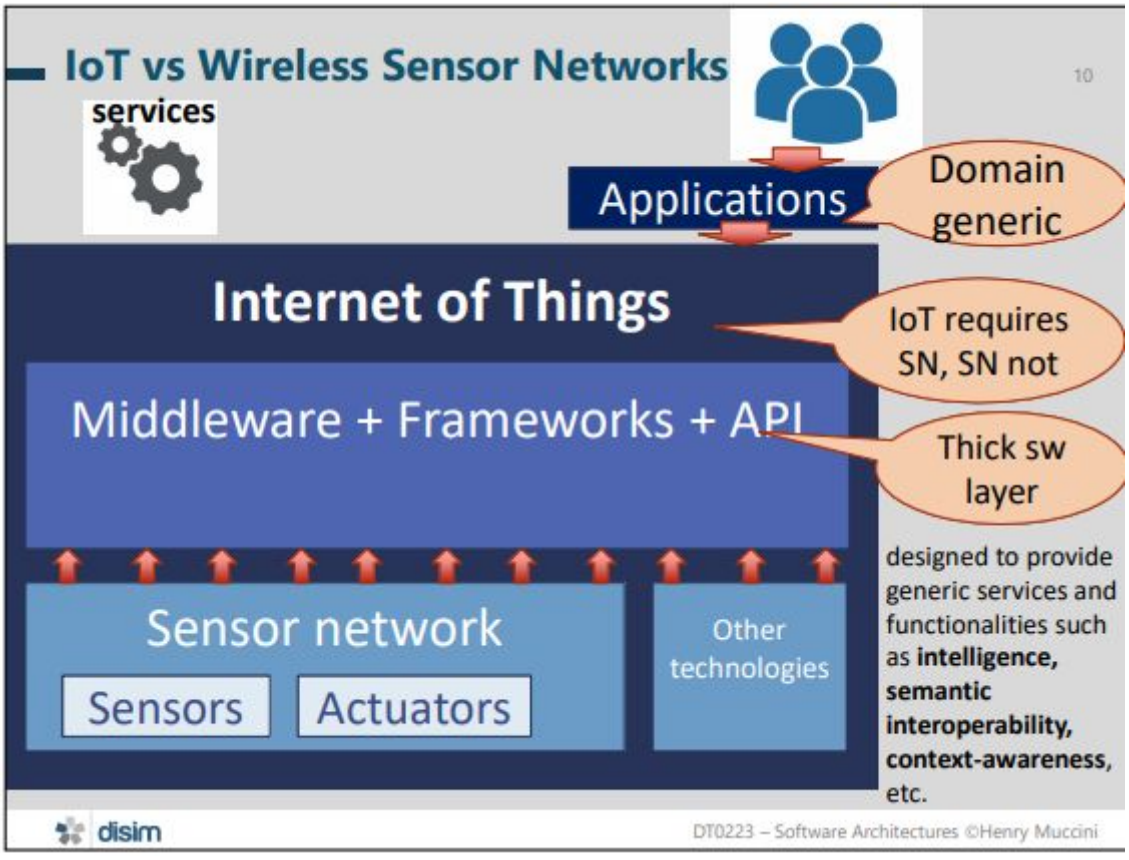
È utile usare un Event-Driver Architecture nei sistemi IOT, poiché è un'architettura scalabile, facile da evolvere ed eterogenea. Gli svantaggi sono l'assenza di responsiveness, il testing può essere tedioso e lo sviluppo complesso.

Riassunto 10 - 11 - 12

# Capitolo 10

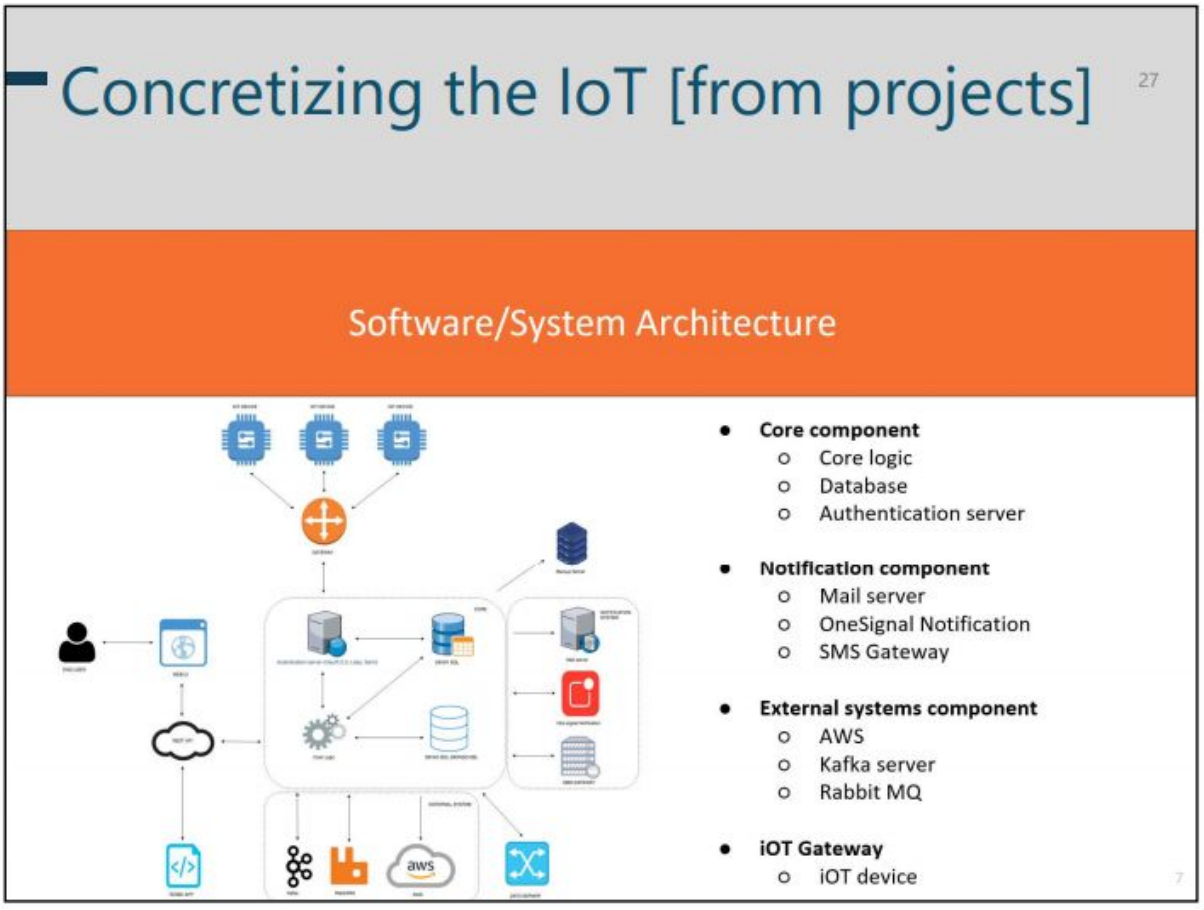
- Started with RFID, expanded with connected sensors, devices , and protocols (e.g. bluethoot) helped with IPv6 that increased the addressing space

- A network of dedicated physical objects that contain embedded technology, Excludes general devices as smartphones tablets and PCs




- The computation is reactive, based on data received from sensors or events triggers, usually is async parallel and distributed
- There is still no consolidated guidelines for engineering software for IoT
- Usually customer are still concerned about IoT safety (e.g hacker attacks)

-



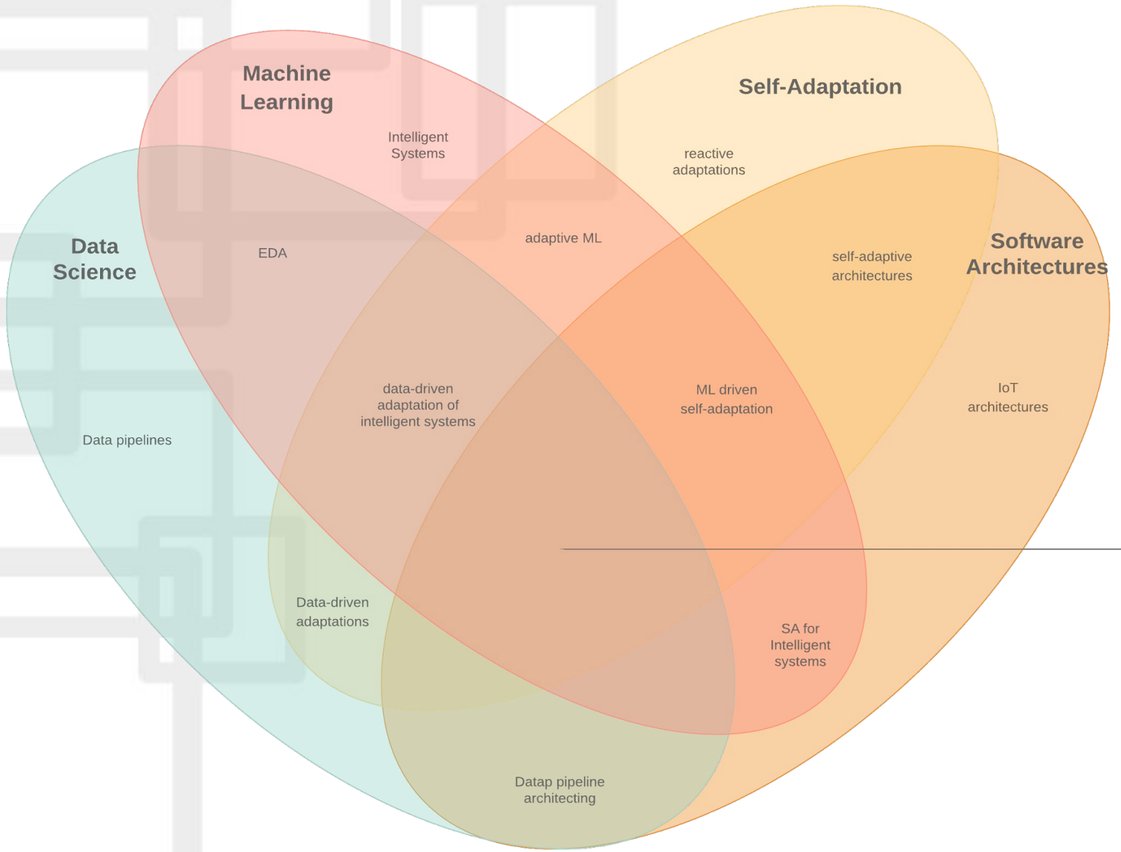

Capitolo 11

- We need to deply faster than ever  (ever multiple times a day)
- The tradition process is very slow, the deployment is a major bottleneck
- Microservice architectures can help with all the bottlenecks from identifying problem, tro fix it, to deploy the fix, via cummunication with messages
- Independant from technology
- Small teams to be more agile, a lot of teams and services
- Continuous deployment is a proces whereby after a commit, software is tested and placed in production without human interventsion
- Deploying consists of substituting N running services of type A with N running services of type B, typically using images and VMs with docker and kubernetes
- Blue/Green update = allocate N versions B once it is done relase all A, Rolling update = allocate 1 B , release 1 A, repeat N times
- Problems:
- Temportal inconsistency: from page to page a user might see a different version of service A, avoided by using features toggle

- - Interface mismatch: A module calls module usign the interface of the old version A’, resolved by making interfaces forward and backwards compatible

Capitolo 12

- Guarda slide sono già scritte super sintetiche



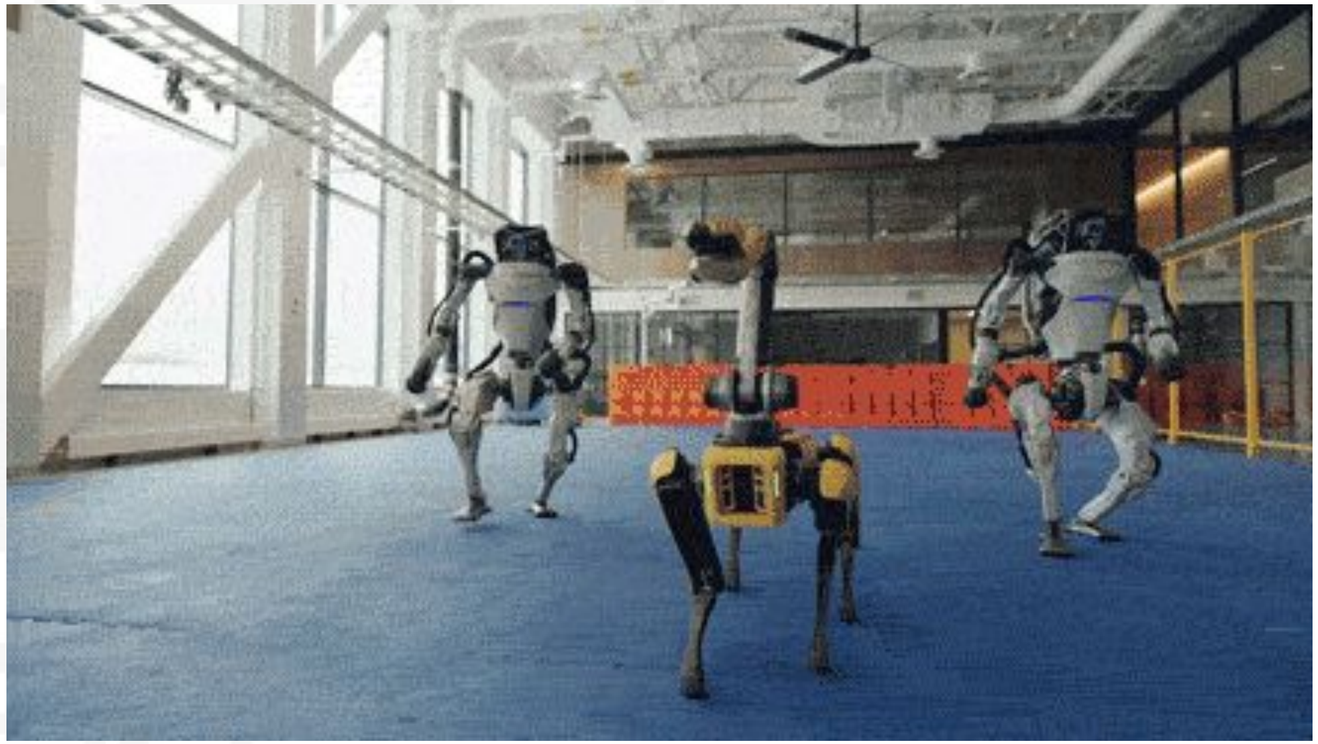

# Data-driven Self-adaptive Architecting using Machine Learning

Presented by: Karthik Vaidhyanathan

Supervised by: Prof. Henry Muccini

www.gssi.it

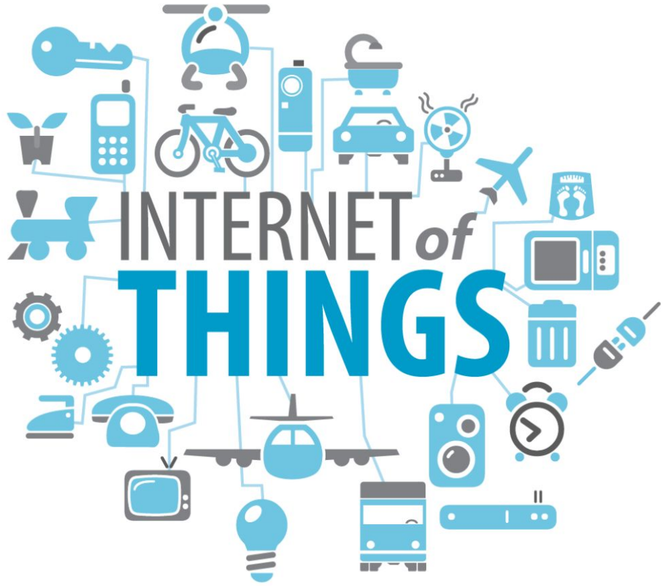
# World of Smart Apps and Complex Software



Robot dance, Boston Dynamics

Tesla Car, Autopilot mode

Source: medium.com, google images



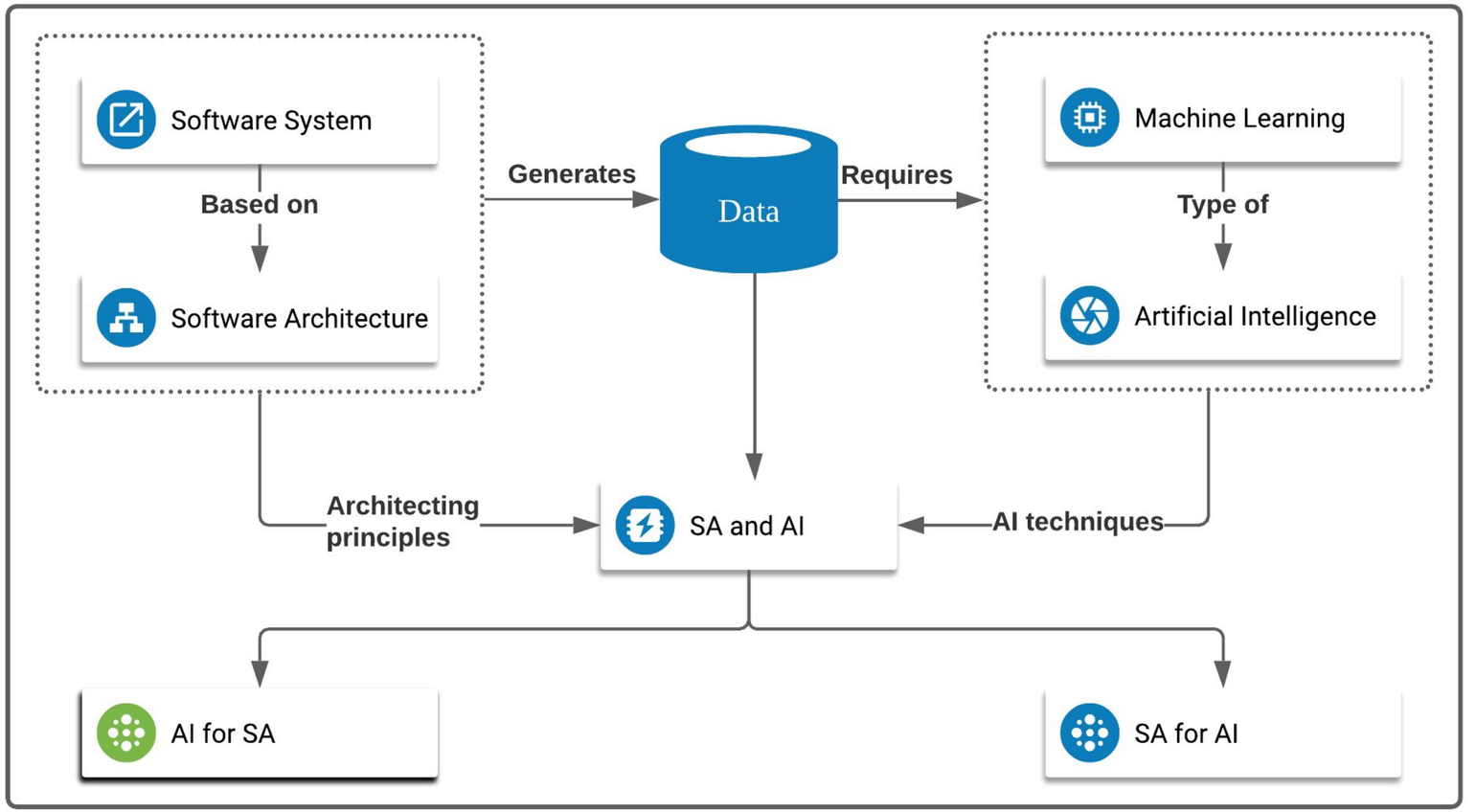

# World of Smart Apps and Complex Software



- Ever growing usage of IoT, cloud native applications coupled with AI revolution
- Increasing complexity => increasing architectural challenges
- There is also increasing adoption of AI leading to lot of intelligent and smart applications

Source: medium.com, google images

# Research Premise



# The Challenges

**How to handle the different uncertainties ?**

1. Resource Constraints (IoT devices, services)
2. Dynamic resource demands

**QoS !!**

3. Network congestion impacting the overall performance
4. Failures (battery drain in IoT, High utilization, microservice failures, app crashes etc)
5. ...

P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "**Microservices: The journey so far and challenges ahead**", IEEE Software ,vol. 35, no. 3, pp. 24–35, 2018.

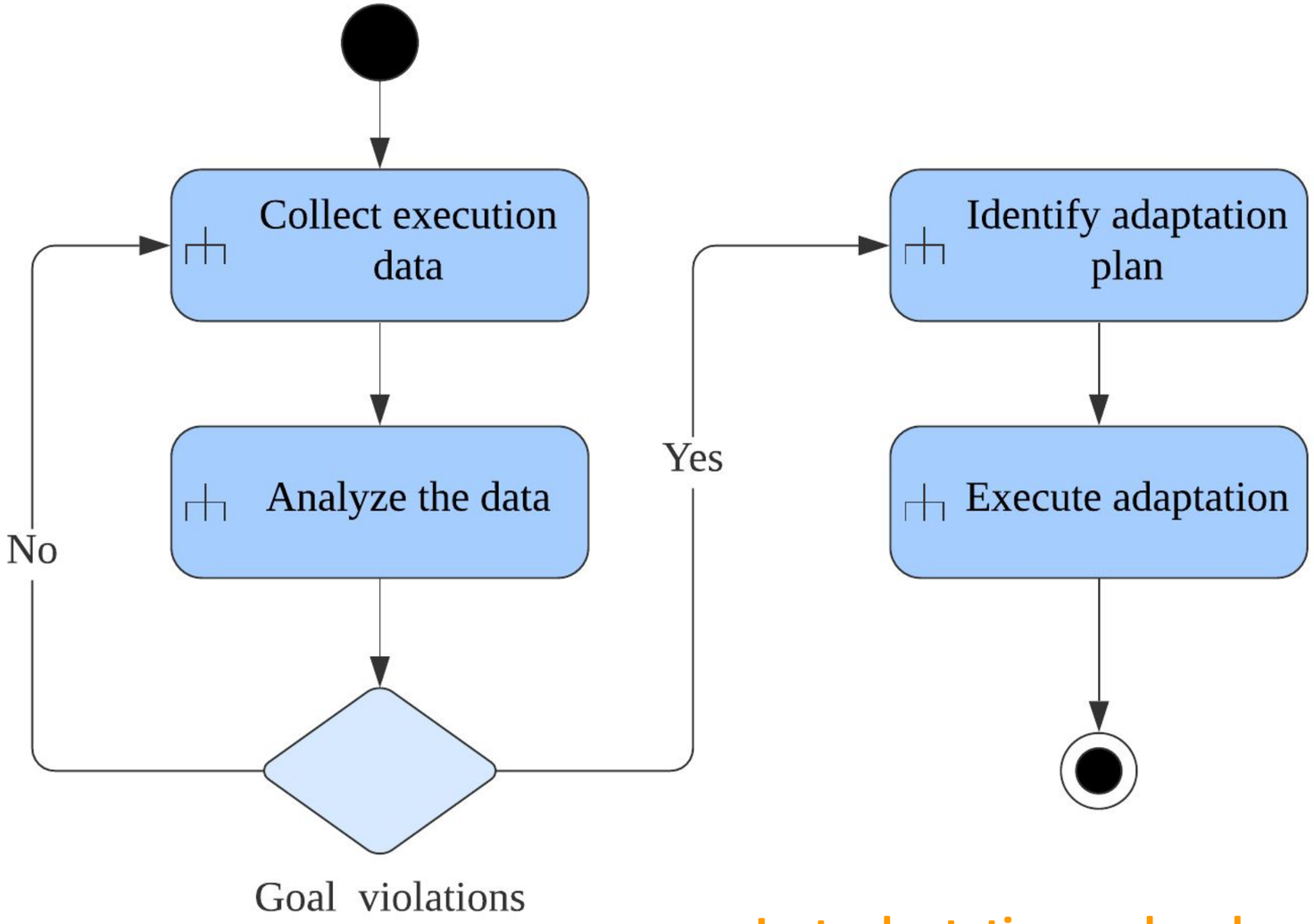
A. Taivalsaari and T. Mikkonen, "**A roadmap to the programmable world: software challenges in the iot era,**" IEEE Software, vol. 34, no. 1, pp. 72–80, 2017.

# Possible Solution

1. Use of self-adaptive architectures - How ?
2. Use the different data generated to aid adaptation - How ?
3. Machine learning can be used to leverage the data[1] - How and what about accuracy ?
4. Use of formal verification techniques - What about recurring adaptations ?
5. Moving from **self-adaptation to self-learning** - let adaptation drive learning

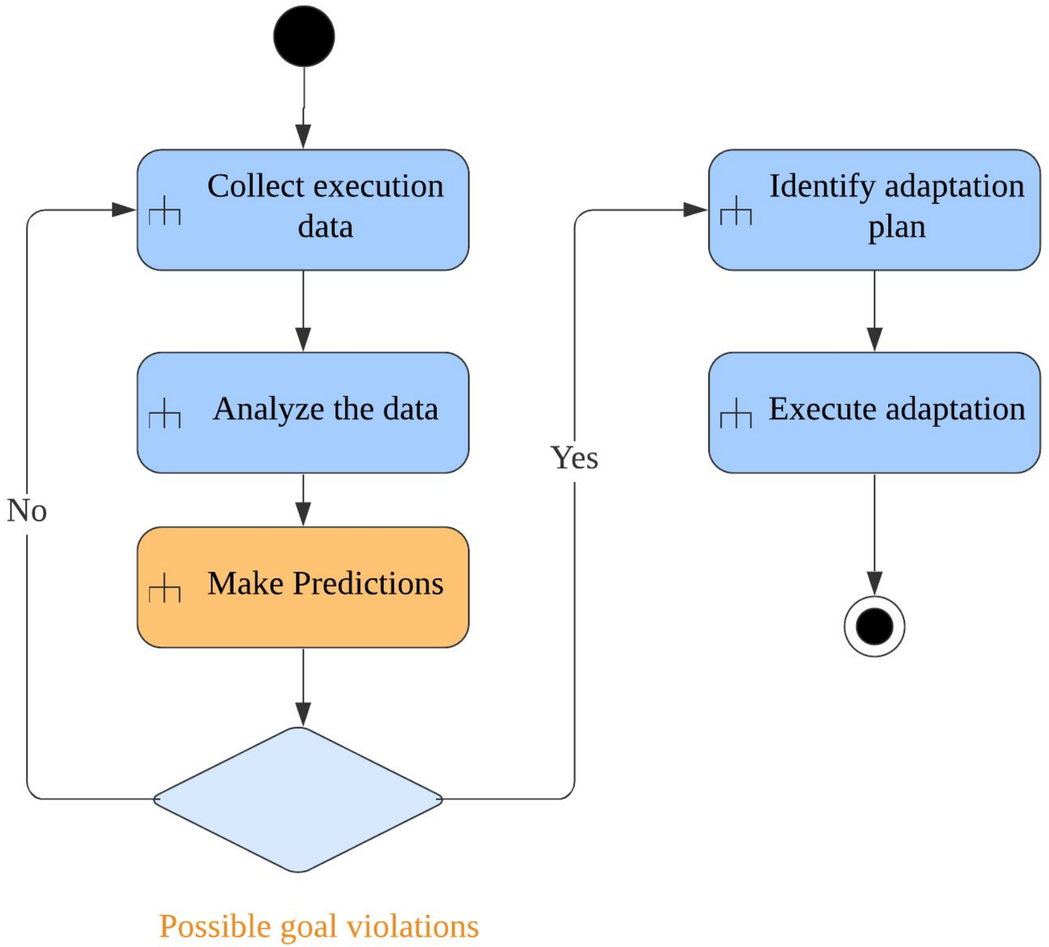
[1]Salehie, Mazeiar and Tahvildari, Lada, **Self-adaptive Software: Landscape and Research Challenges**,ACM Trans. Auton. Adapt. Syst, May 2009

# Self-adaptation process - Reactive




**Just adaptation and no learning !!**

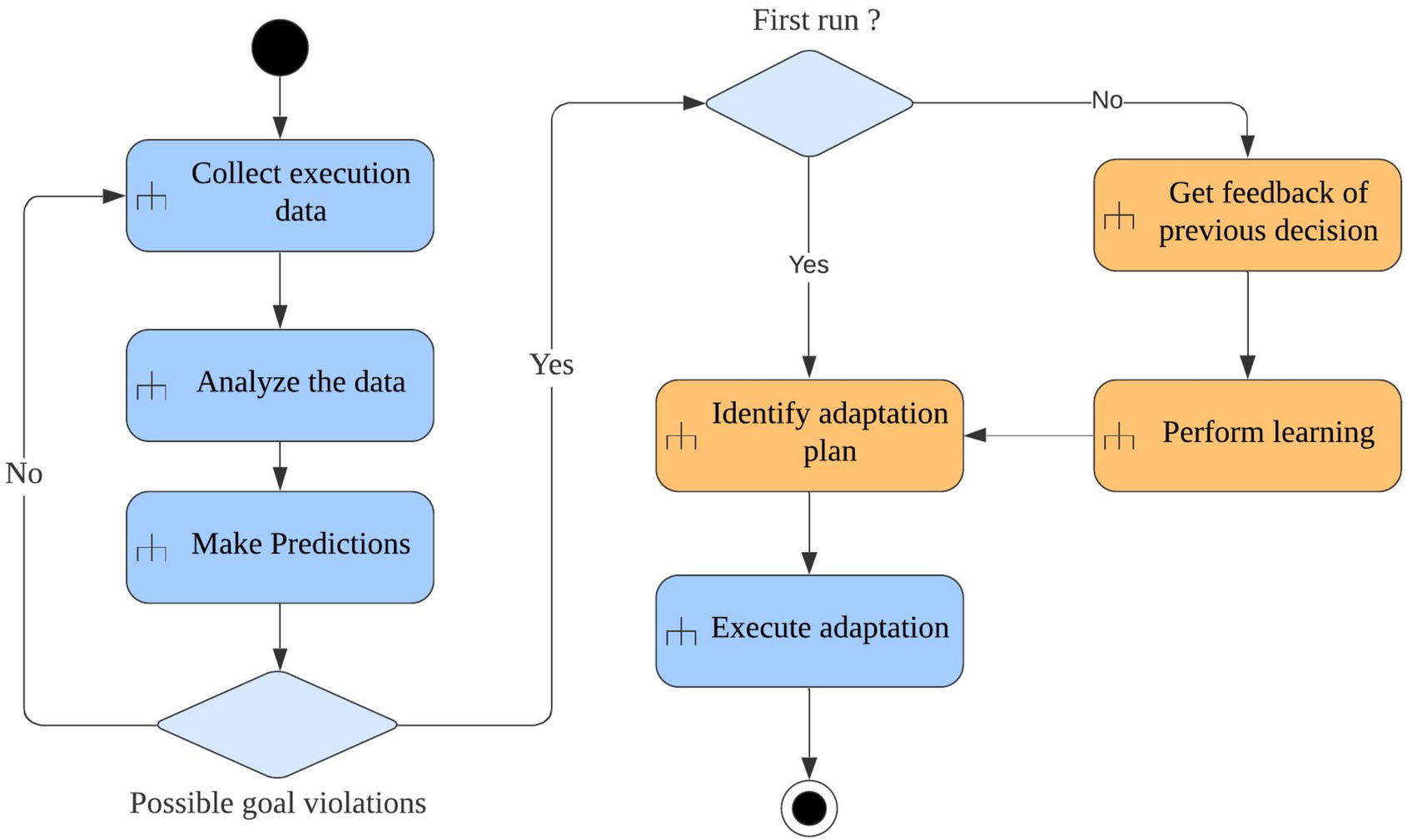
# Self-adaptation process - Proactive




**Just adaptation and no learning !!**

Henry Muccini and Karthik Vaidhyanathan, "**PIE-ML: A Machine learning-driven Proactive Approach for Architecting Self-adaptive Energy Efficient IoT Systems**", DISIM, University of L'Aquila, Italy, TRCS: 002/2020, July. 15, 2020.[Online]. Available: https://tinyurl.com/y98weaa

# Self-adaptation process - ML-driven




Henry Muccini, Karthik Vaidhyanathan. **Leveraging Machine Learning Techniques for Architecting Self-Adaptive IoT Systems.** Proceedings of the 6th IEEE International Conference on Smart Computing (SMARTCOMP 2020)

# ML is not always right !!

Vox recode

1

## Tesla needs to fix its deadly Autopilot problem

Tesla is facing heat from federal officials following an investigation into a fatal crash involving its Autopilot.

By Rebecca Heilweil | Feb 26, 2020, 1:50pm EST

2

MIT Technology Review Topics

IAN WALDIE/GETTY IMAGES

Tech policy / AI Ethics

## AI is sending people to jail —and getting it wrong

Using historical data to train risk assessment tools could mean that machines are copying the mistakes of the past.

by Karen Hao January 21, 2019

3

THE VERGE TECH REVIEWS SCIENCE CREATORS ENTERTAINMENT VIDEO MORE

TECH ARTIFICIAL INTELLIGENCE

TL;DR

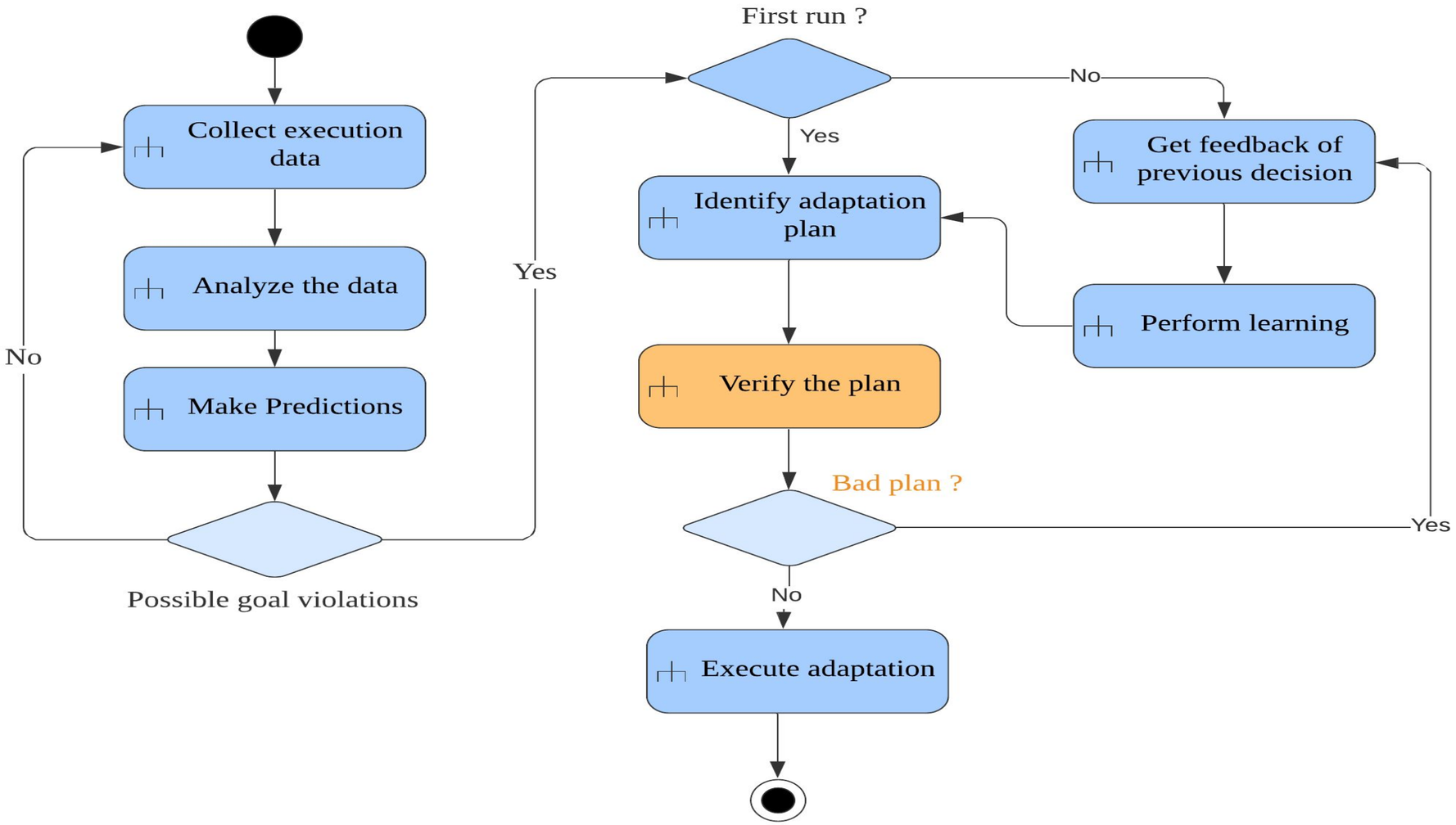
## AI camera operator repeatedly confuses bald head for soccer ball during live stream

*Like a distracted AI with a crush*

By James Vincent | Nov 3, 2020, 8:07am EST

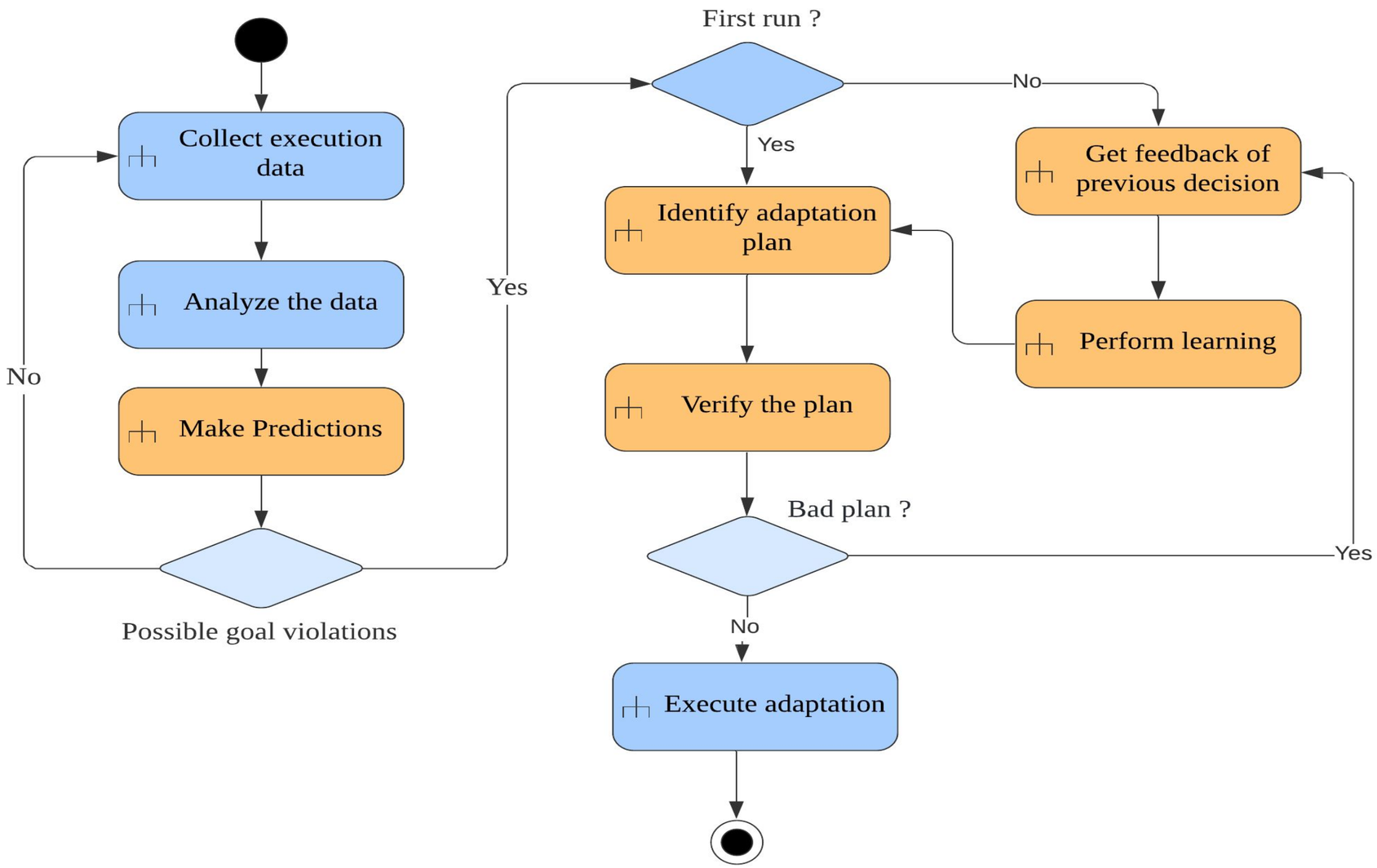
1. https://www.vox.com/recode/2020/2/26/21154502/tesla-autopilot-fatal-crashes
2. https://www.technologyreview.com/2019/01/21/137783/algorithms-criminal-justice-ai/
3. https://www.theverge.com/tldr/2020/11/3/21547392/ai-camera-operator-football-bald-head-soccer-mistakes

# Self-adaptation process - Verification aided ML

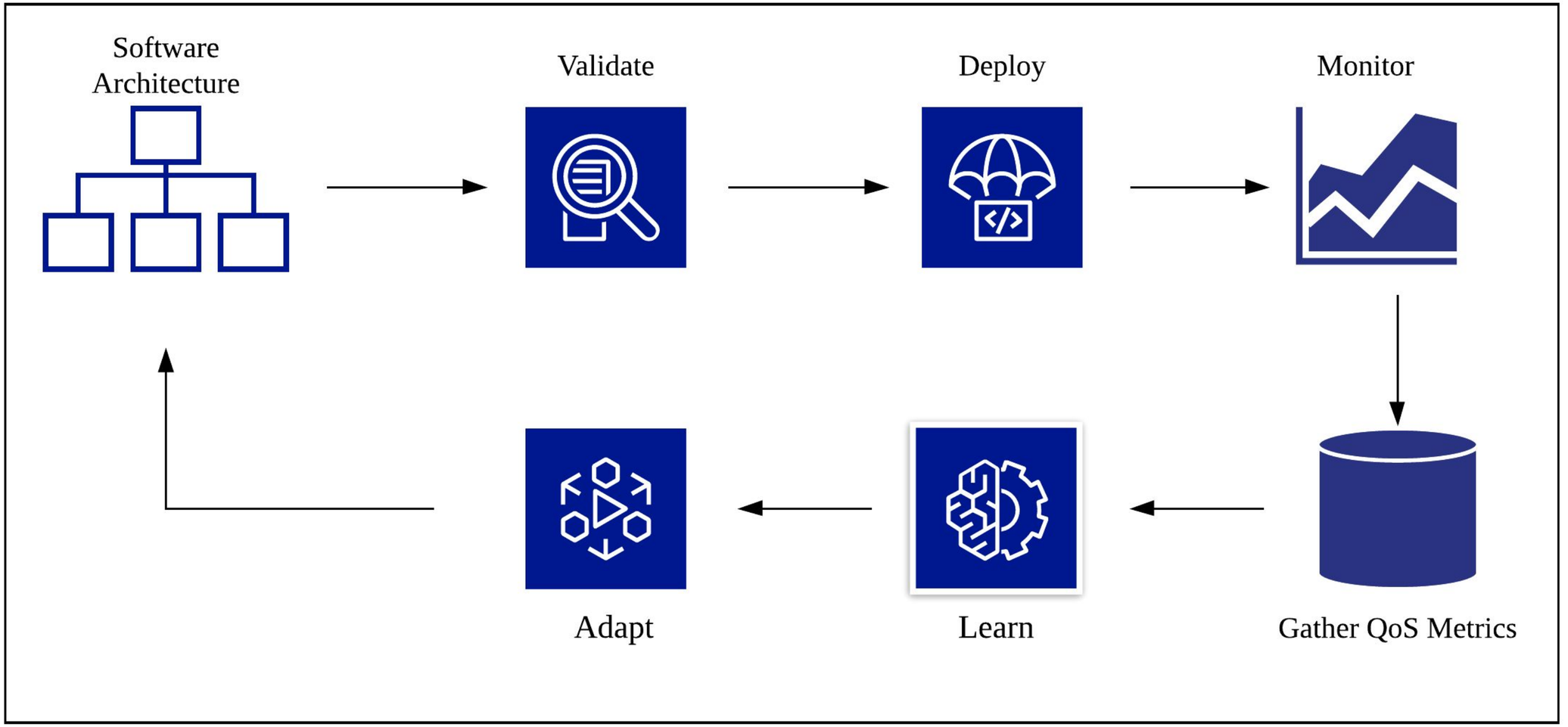



Javier Cámara Moreno, Henry Muccini, Karthik Vaidhyanathan. **Quantitative Verification-Aided Machine Learning: A Mixed-Method Approach for Architecting Self-Adaptive IoT Systems.** Proceedings of ICSA 2020

# Our Research Activities



# Overall Approach




Henry Muccini and Karthik Vaidhyanathan. **Towards self-learnable software architectures**. ERCIM News 2020(122), 2020. URL https://ercim-news.ercim.eu/en122/special/towards-self-learnable-software-architectures.

# Ongoing and Future Work

1. Extension of verfication aided ML to more generic class of systems
2. ML-driven service discovery for microservice architectures
3. Data-driven adaptation in microservice-based IoT architectures[1]
4. Software architectures for AI systems - Area to explore

Martina De Sanctis, Henry Muccini, Karthik Vaidhyanathan. **Data-driven Adaptation in Microservice-based IoT Architectures.** Proceedings of ICSA 2020 Companion

# Master Thesis

**UNIVERSITA' DEGLI STUDI - L'AQUILA**

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Corso di Laurea in Informatica

L'Aquila, June 8th 2020

A thesis on the following topic is available "**Machine learning-driven Context-aware service discovery for Microservice architectures**"

Thesis Degree: Master Thesis in Computer Science

Prior Knowledge: Basic understanding of Microservice architectures, Basic to intermediate knowledge of Machine Learning, Knowledge of dockers and deployment technologies is a plus

In collaboration with: Mauro Caporuscio, Associate Professor, Linnaeus University, Sweden and Karthik Vaidhyanathan, Ph.D. student, Gran Sasso Science Institute

T

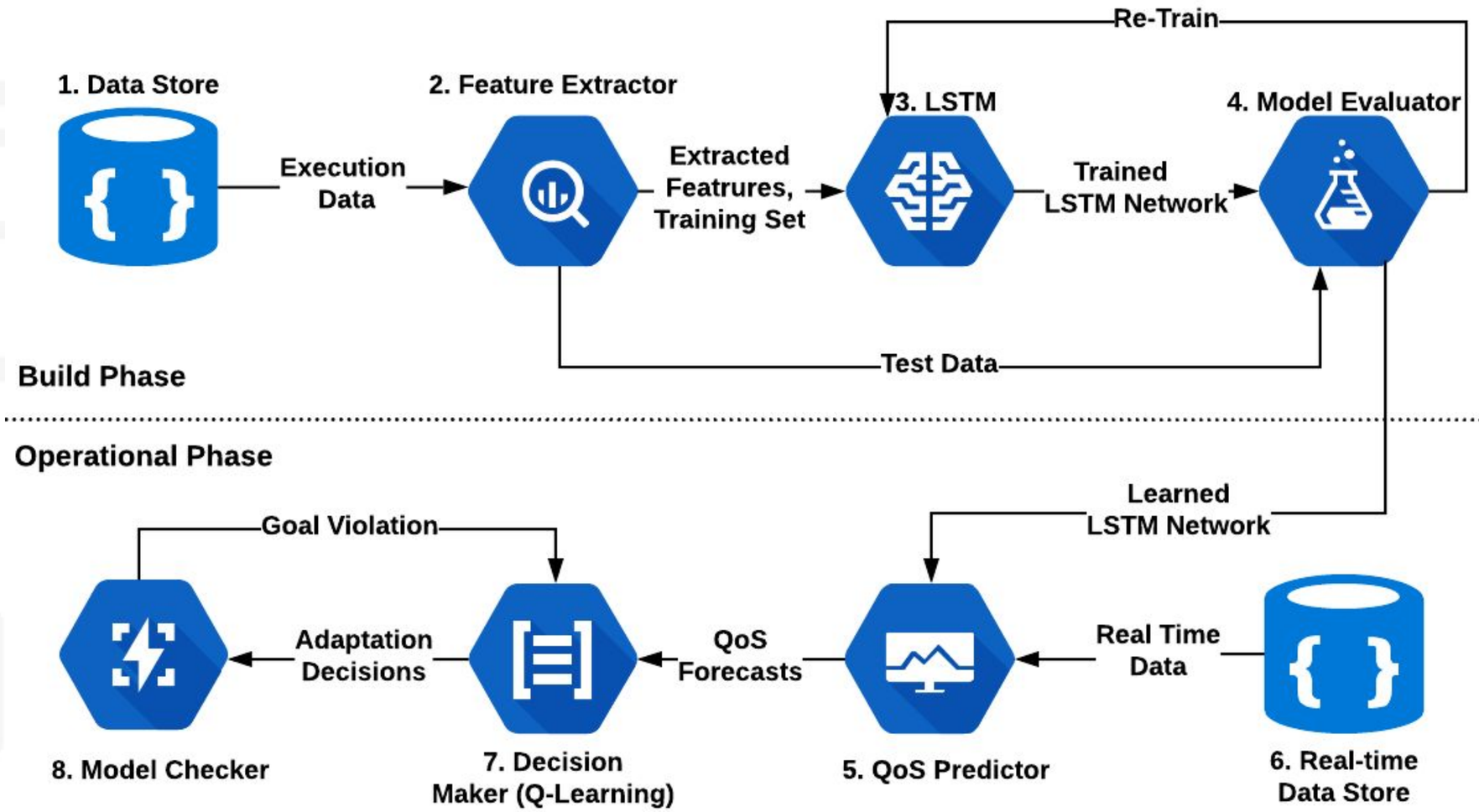
# Thank You


Karthik Vaidhyanathan

karthik.vaidhyanathan@gssi.it

www.karthikvaid.me

# Approach Overview



# Approach Overview