LabIss2018 - Overveiw

A reference map for our journey

Design and Build a Software Application starting from a set of Requirements

RESUSE Library, Framework, Infrastructure - Pattern

Bottom-up *Synthesis*

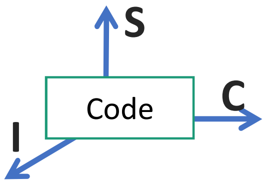
Computer Machine - Computational paradigm – Programming Language

From algorithms to systems - Transformational / Interactive / Reactive

Elements Functions – Objects – Active Objects - Actors – Agents – (Micro)Services

Patterns - Architectures

Architecture Structure / Interaction / Behaviour – Layered / Hexagonal



Interaction Calls, Messages, Events, MOMs, TupleSpaces …

Messages: Dispatch, Invitation, Request

Behavior : Message or Event-driven / State Machines

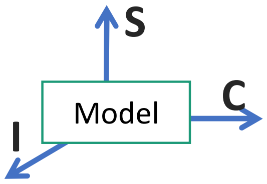
Riduzionismo/Olismo

Test plans and testing

Top-down *Analysis* / Design

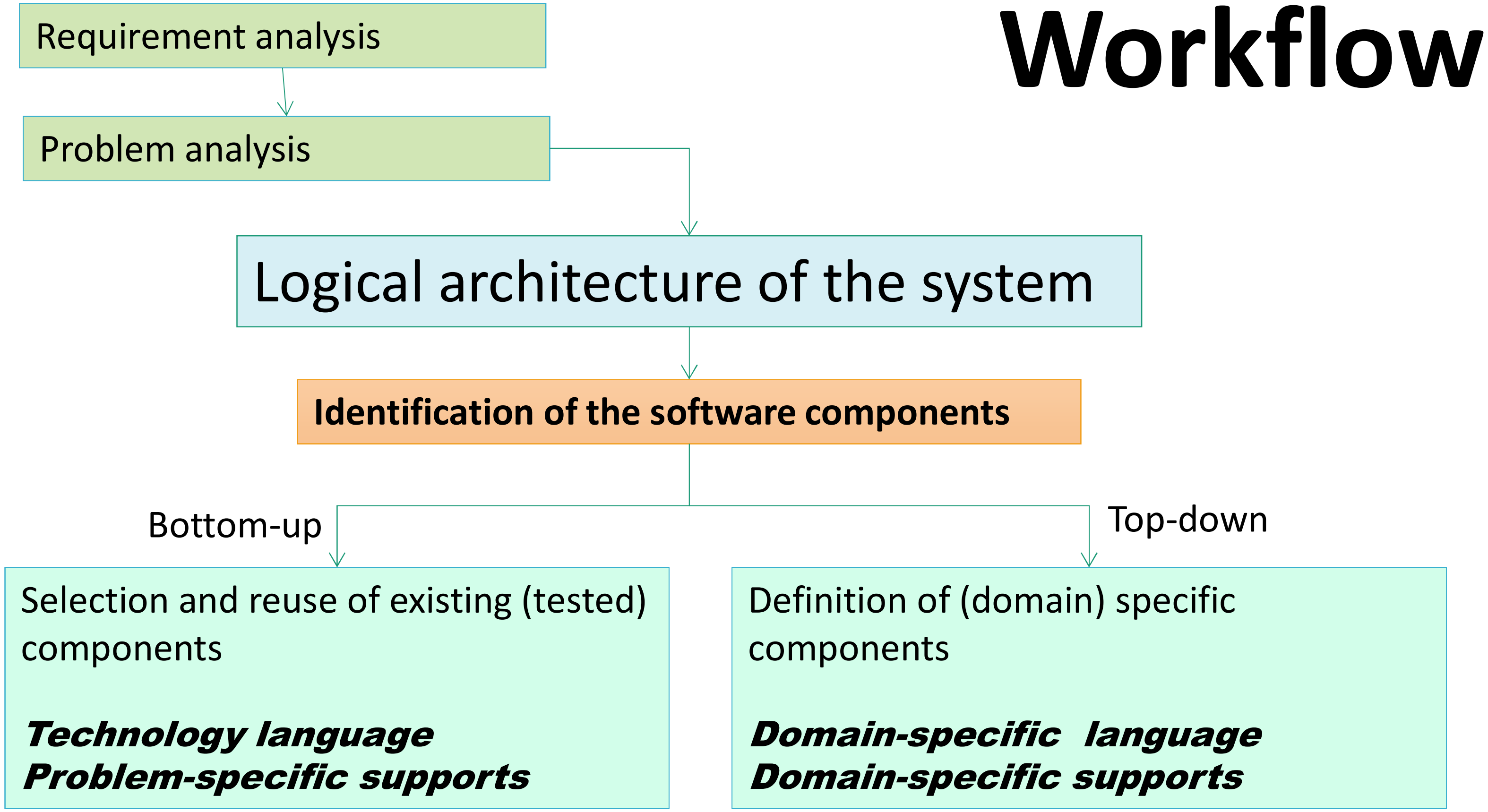
Models

From models to code – Software Factories



Domains Vocabulary – Domain Specific Languages – Domain-Driven Design

Workflow



Requirement analysis

Problem analysis

Logical architecture of the system

Identification of the software components

Bottom-up

Top-down

Selection and reuse of existing (tested) components

Technology language
Problem-specific supports

Definition of (domain) specific components

Domain-specific language
Domain-specific supports

In any case:

What is a software component?
How components interact?
Which components embed the ***business logic***?
Is it possible to fulfil the requirements with a sequence of systems (of increasing complexity)?

Material

- *http://infolab.ingce.unibo.it/iss2018/it.unibo.issMaterial/issdocs/Material/LectureCesena1819.html*
- *A Button-Led systems: from objects to actors and services*

GIT Repo: https://github.com/anatali/iss2018Lab

- Project *it.unibo.bls17.naive*

Languages

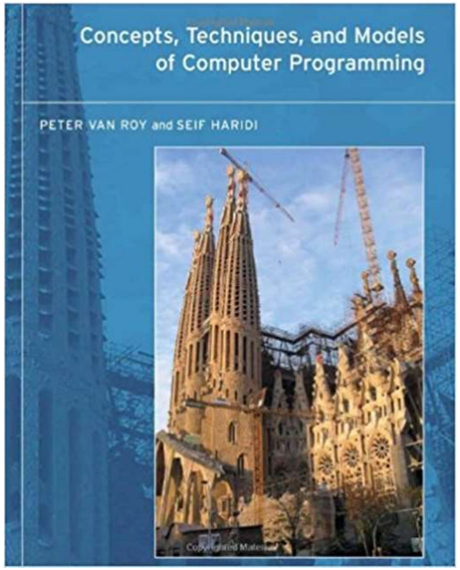
What language to use?

https://en.wikipedia.org/wiki/List_of_programming_languages

https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages

Why so many programming languages?

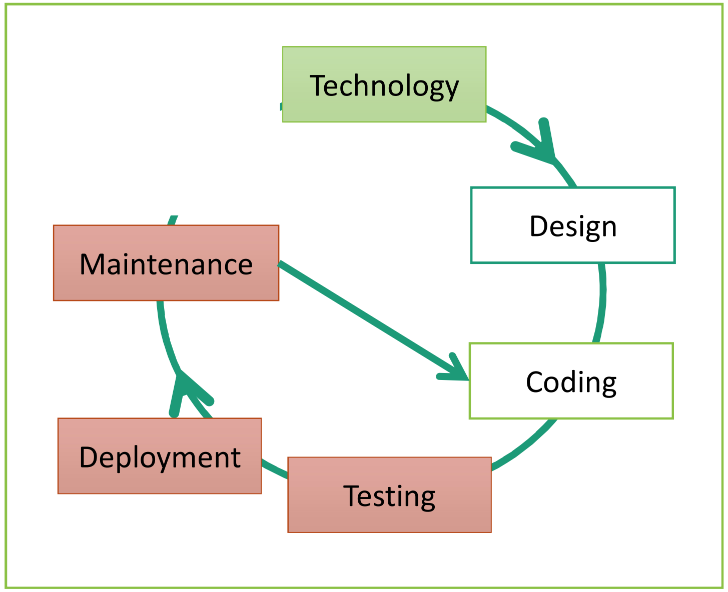
https://en.wikipedia.org/wiki/History_of_programming_languages

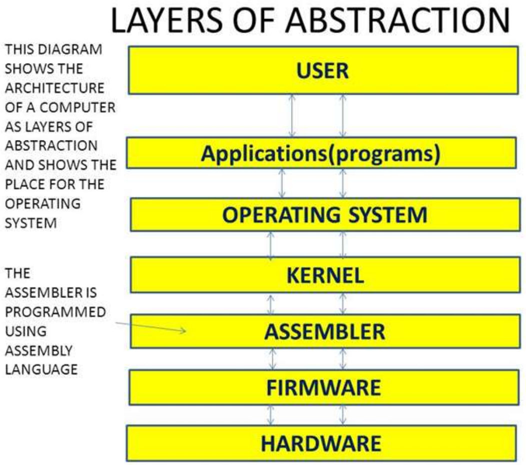

Concepts, Techniques, and Models of Computer Programming
PETER VAN ROY and SEIF HARIDI

This innovative text presents computer programming as a unified discipline in a way that is both practical and scientifically sound. The book presents both well-known and lesser-known computation models ("programming paradigms"). Each model has its own set of techniques and each is included on the basis of its usefulness in practice. The general models include declarative programming, declarative concurrency, message-passing concurrency, explicit state, object-oriented programming, shared-state concurrency, and relational programming. Specialized models include graphical user interface programming, distributed programming, and constraint programming.

Bottom-up

Tecnhology-first




LAYERS OF ABSTRACTION

THIS DIAGRAM SHOWS THE ARCHITECTURE OF A COMPUTER AS LAYERS OF ABSTRACTION AND SHOWS THE PLACE FOR THE OPERATING SYSTEM

THE ASSEMBLER IS PROGRAMMED USING ASSEMBLY LANGUAGE

In mathematical logic and theoretical computer science a **register machine** is a generic class of abstract machines used in a manner similar to a Turing machine.

The Minsky machine

ZERO cell
INC cell
SUBJZ cell label
HALT

Is Turing equiivalent (two tapes and a simple Gödelization).

The canonical reference is Minsky's book, *Computation: Finite and Infinite Machines* (Prentice-Hall International, 1967; ISBN 0131655639), in which he calls these machines *program machines*.

Language
- Java (C#, C++ …)
- JavaScript, Node
- Python, Lua ,…
- Koplin
- …

Paradigm /Style
- Oop
- Event driven
- Client-Server
- Actors
- Agents
- …

Kinds of software

It is convenient to distinguish roughly between three kinds of computer programs (Gérard Berry):

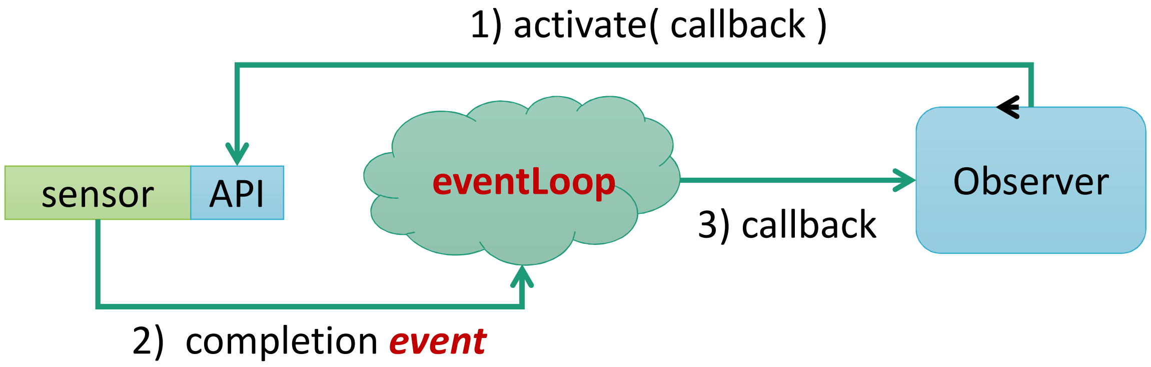
- **Transformational programs** compute results from a given set of inputs; typical examples are compilers or numerical computation programs.
- **Interactive programs** interact at their own speed with users or with other programs; from a user point of view, a time-sharing system is interactive.
- **Reactive programs** also maintain a continuous interaction with their environment, but at a speed which is determined by the environment, not the program itself.

Interactive programs work at their own pace and mostly deal with communication, while *reactive programs* only work in response to external demands and mostly deal with accurate interrupt handling.

Real-time programs are usually reactive. However, there are reactive programs that are not usually considered as being real-time, such as protocols, system drivers, or man-machine interface handlers.

EventDriven

Node (javascript) / **Asynchronous operations**



Single thread
One action

The callback hell

Dispense AN (pdf):

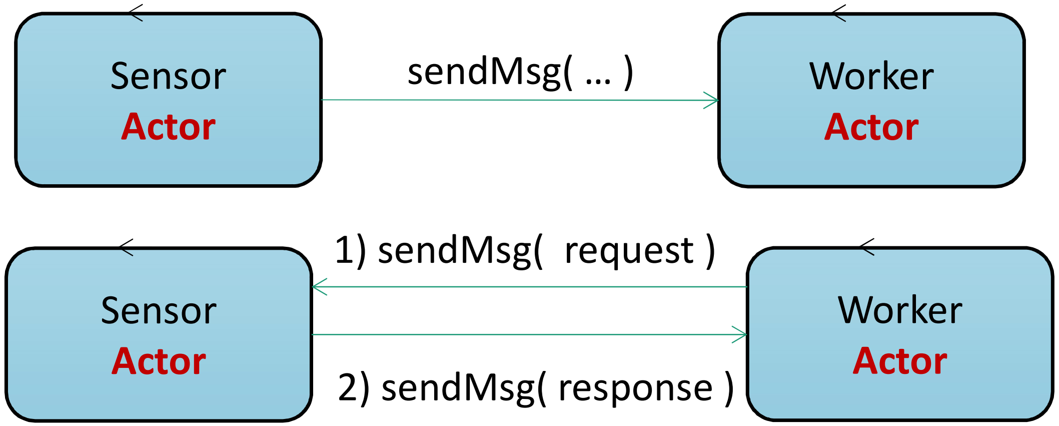
Event programming in JavaScript and Node.js: an introduction

The ButtonLed system in JavaScript and Node.js

Project:

it.unibo.nodejs.intro

Message Passing



Sensor **Actor**

sendMsg(…)

Worker **Actor**

Sensor **Actor**

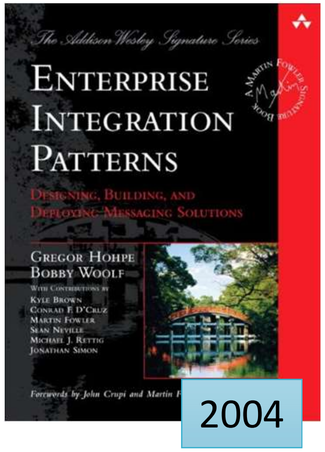
1) sendMsg(request)

2) sendMsg(response)

Worker **Actor**

```
onReceive( …) {
     ….
}
```

Single thread
One message


The Addison Wesley Signature Series
Enterprise Integration Patterns
Designing, Building, and Deploying Messaging Solutions
Gregor Hohpe
Bobby Woolf
A Martin Fowler Signature Book

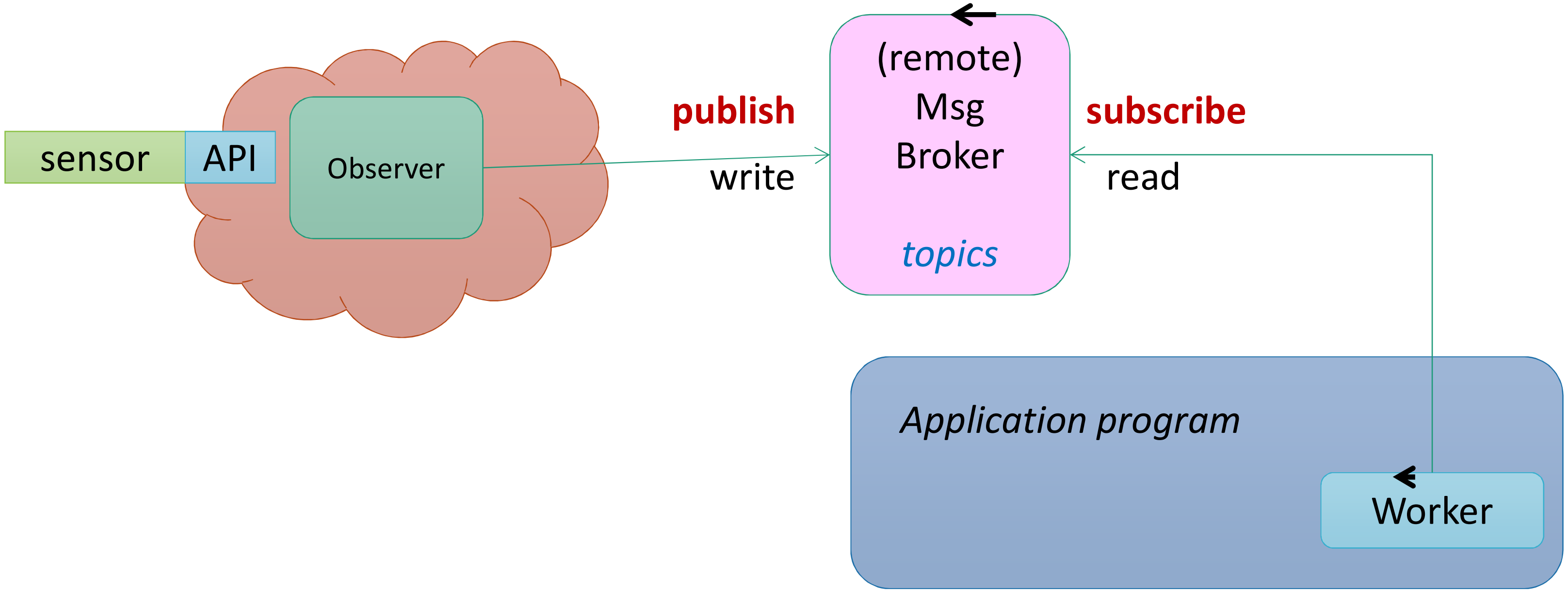
2004

This book is about how to use messages to **integrate applications.** This book provides a **consistent vocabulary** and visual notation framework to describe large-scale integration solutions across **many technologies**. It also explores in detail the advantages and limitations of asynchronous messaging architectures.

The authors also include examples covering a variety of different integration technologies, such as JMS, MSMQ, TIBCO ActiveEnterprise, Microsoft BizTalk, SOAP, and XSL. A case study describing a bond trading system illustrates the patterns in practice, and the book offers a look at emerging standards, as well as insights into what the future of enterprise integration might hold.

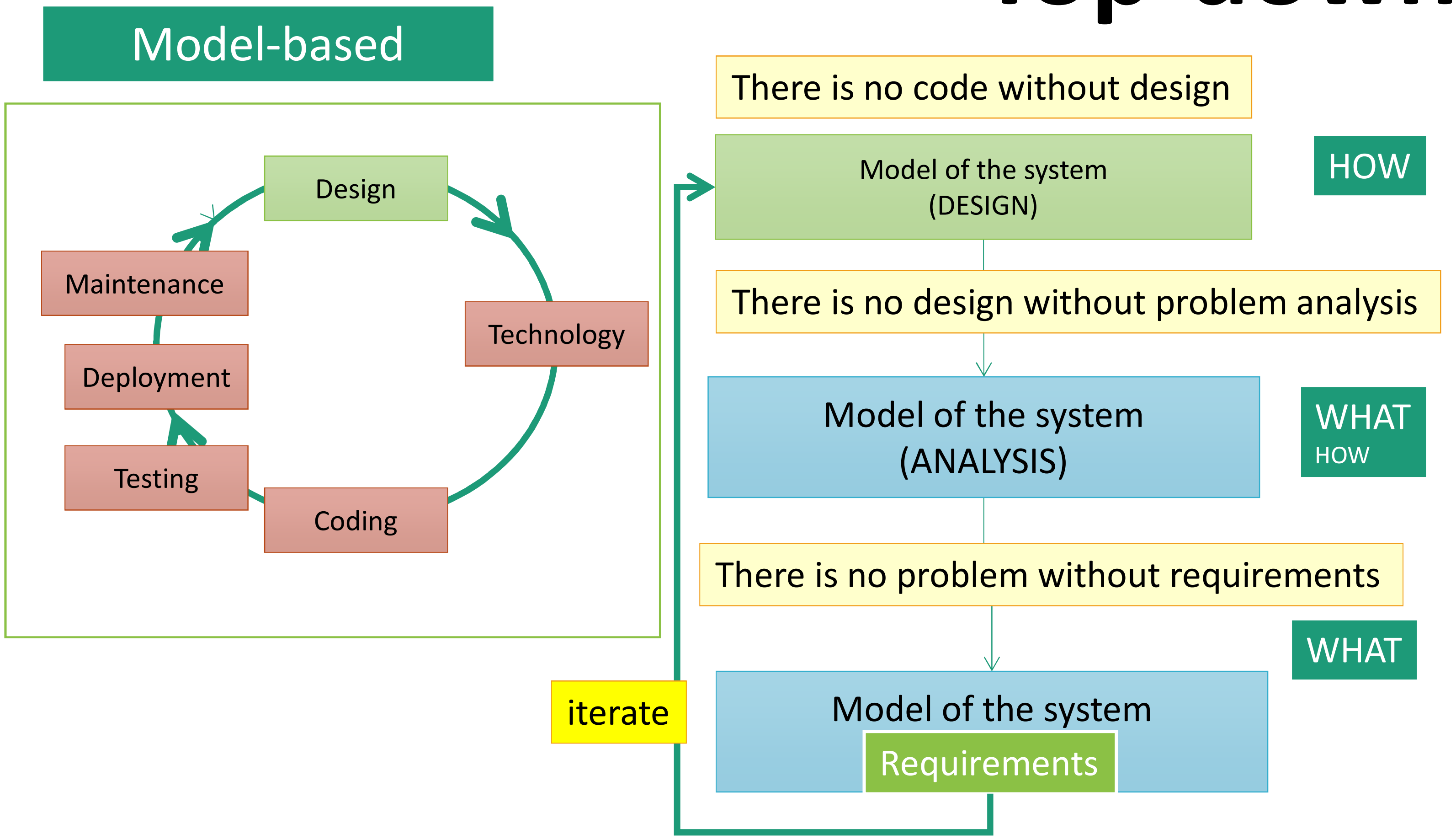
MOM

Publish–subscribe pattern



The MQ Telemetry Transport (MQTT) is an ISO standard (ISO/IEC PRF 20922) supported by the OASIS organization.
Eclipse Mosquitto™ is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 3.1 and 3.1.1.

Top-down



Model-based

Design

Technology

Coding

Testing

Deployment

Maintenance

There is no code without design

Model of the system (DESIGN)

HOW

There is no design without problem analysis

Model of the system (ANALYSIS)

WHAT HOW

There is no problem without requirements

WHAT

Model of the system

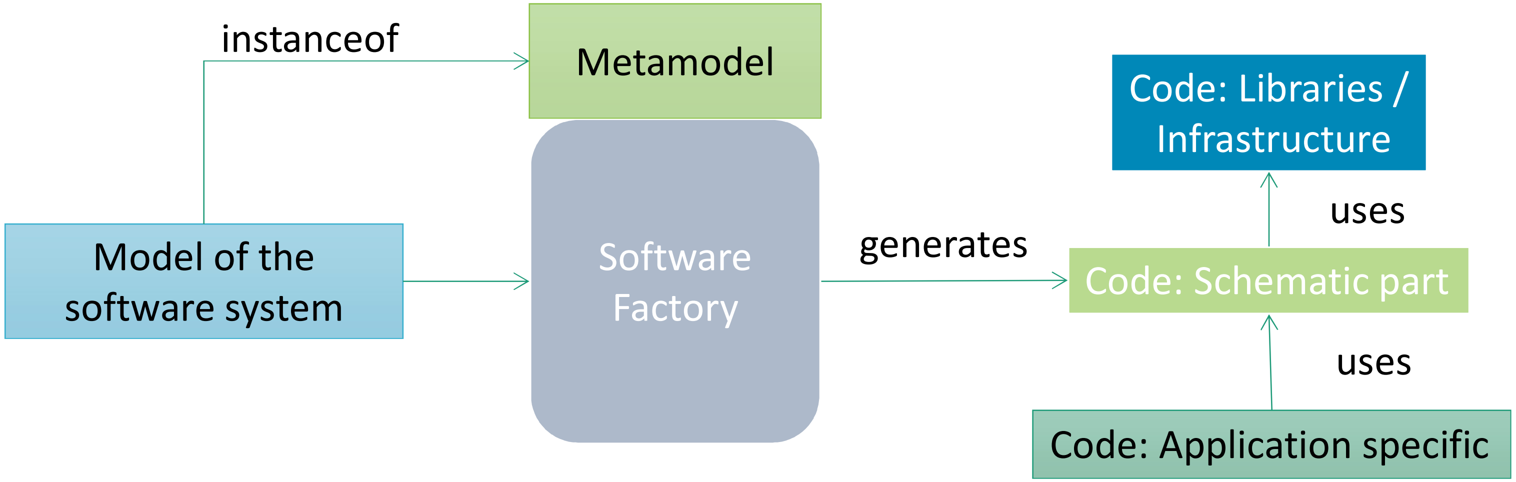
Requirements

iterate

https://en.wikipedia.org/wiki/**Requirements_analysis**

https://www.eclipse.org/**rmf**/

MDSD



instanceof

Metamodel

Model of the software system

Software Factory

generates

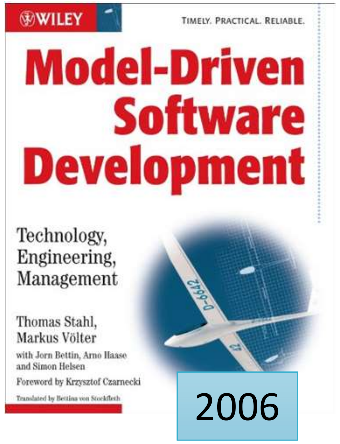
Code: Libraries / Infrastructure

uses

Code: Schematic part

uses

Code: Application specific



WILEY

TIMELY. PRACTICAL. RELIABLE.

Model-Driven Software Development

Technology, Engineering, Management

Thomas Stahl, Markus Völter

with Jorn Bettin, Arno Haase and Simon Helsen

Foreword by Krzysztof Czarnecki

Translated by Bettina von Stockfleth

2006

- Model-Driven Software Development (MDSD) puts **analysis and design models** on par with code.
- Models do not constitute documentation, but are considered equal to code, as their implementation is automated.
- The goal of the book is to convince you, the reader, that MDSD is a practicable method today, and that it is superior to conventional development methods in many cases.

Main roles

There are four main roles in the development process of software

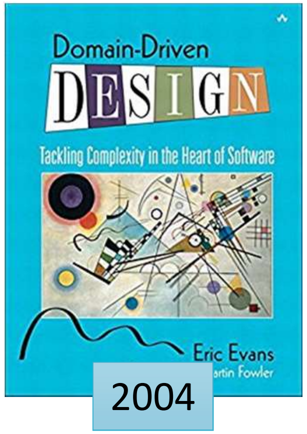
– **Domain expert**

– **Designer**

– **Software developer**

– **End user**

DOMAINS


2004

(pg. xxi) Some software design factors are technological. Yet, the *most significant complexity of many application is not techcnical*

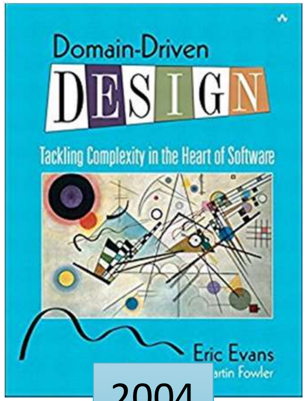
- For most software projects, the **primary focus** should be on **domain** and **domain logic**.
- Complex domain designs should be based on **models**.

Developers are often insulated from the domain experts. If a developer does not understand a concept, it is likely the implementation will not accurately reflect the domain.

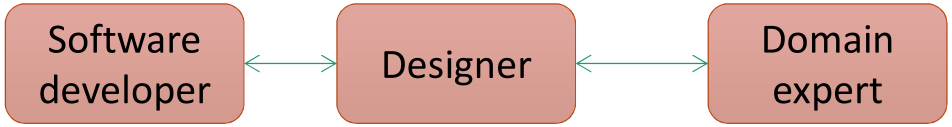
To **facilitate communications** between domain experts, designers and developers:

- Establish a **common language** (UBIQUITOUS LANGUAGE)
- **Iterating a single model** to relfect sharing understanding across domain experts, , designers and developers.

DDD



2004



Software developer

Designer

Domain expert

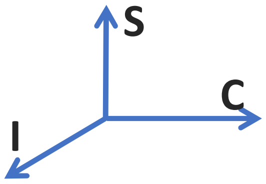
- Developers are insulated from the domain experts. If a developer does not understand a concept, it is likely the implementation will not accurately reflect the domain.
- Developers without solid design principles will produce a code that is hard to understand or change – the oppostive of agility. (pg. xxiij)

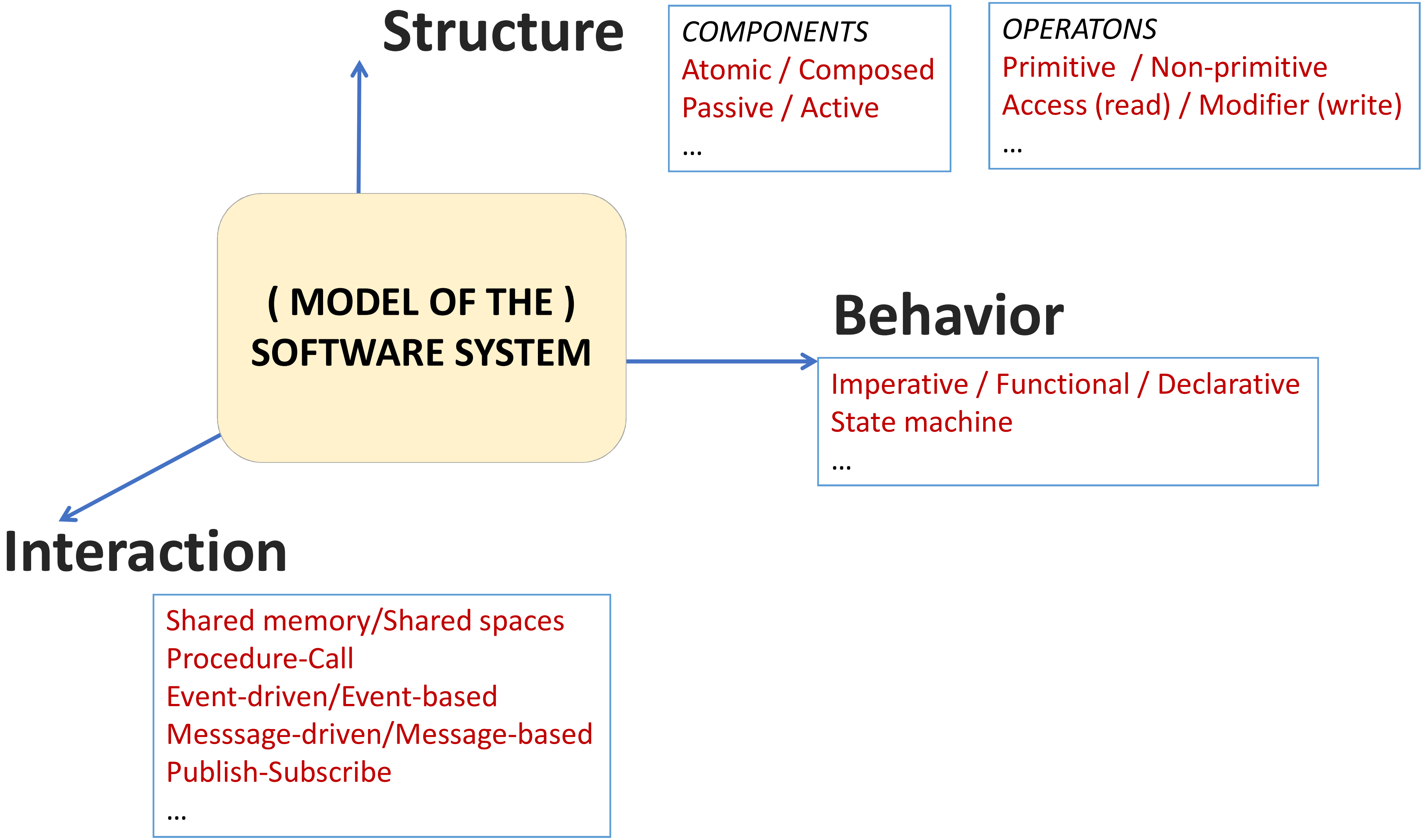
Domain model: a rigorously organized and selective abstraction of the knowledge in a domain's expert head (pg. 3).
One model should underlie implementation, design and team communications (pg. 41)
The model is the backbone of a language used by all team members (pg.4, 26).

- iterate a single model to reflect a shared understanding across domain experts, designers and developers
- establisha common language, i.e. a UBIQUITOUS LANGUAGE (pg. 24)

Code as a design document does have its limits (pg. 38). A document should explain the concepts of the model and must be involved in project activities (pg.39)

Effective domain modelers are knowledge crunchers. Continuous learning takes place between domain experts, designers and developers (pg. 15)
(OO) MODEL-DRIVEN DESIGN pg. 47


S
C
I



(MODEL OF THE) SOFTWARE SYSTEM

Structure

COMPONENTS
Atomic / Composed
Passive / Active
…

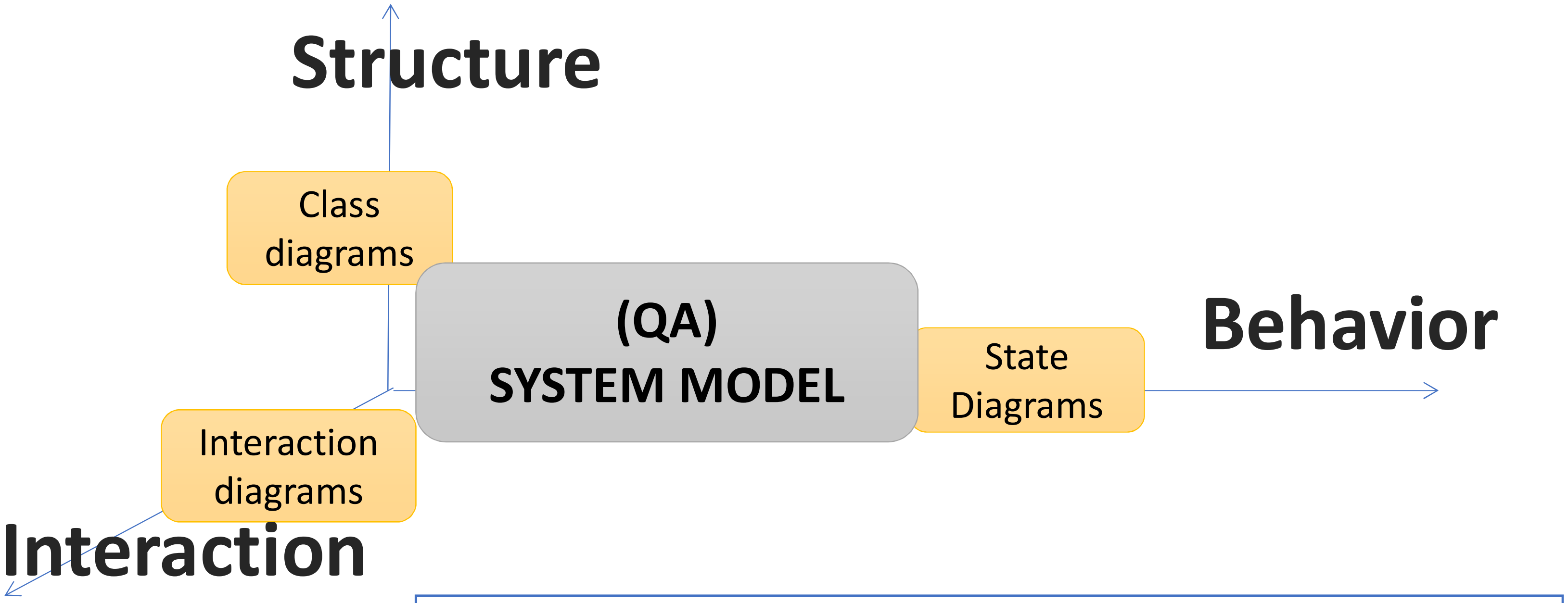
OPERATONS
Primitive / Non-primitive
Access (read) / Modifier (write)
…

Behavior

Imperative / Functional / Declarative
State machine
…

Interaction

Shared memory/Shared spaces
Procedure-Call
Event-driven/Event-based
Messsage-driven/Message-based
Publish-Subscribe
…



Enterprise Application Software

First generation

Client-server.

Business model: software licensing

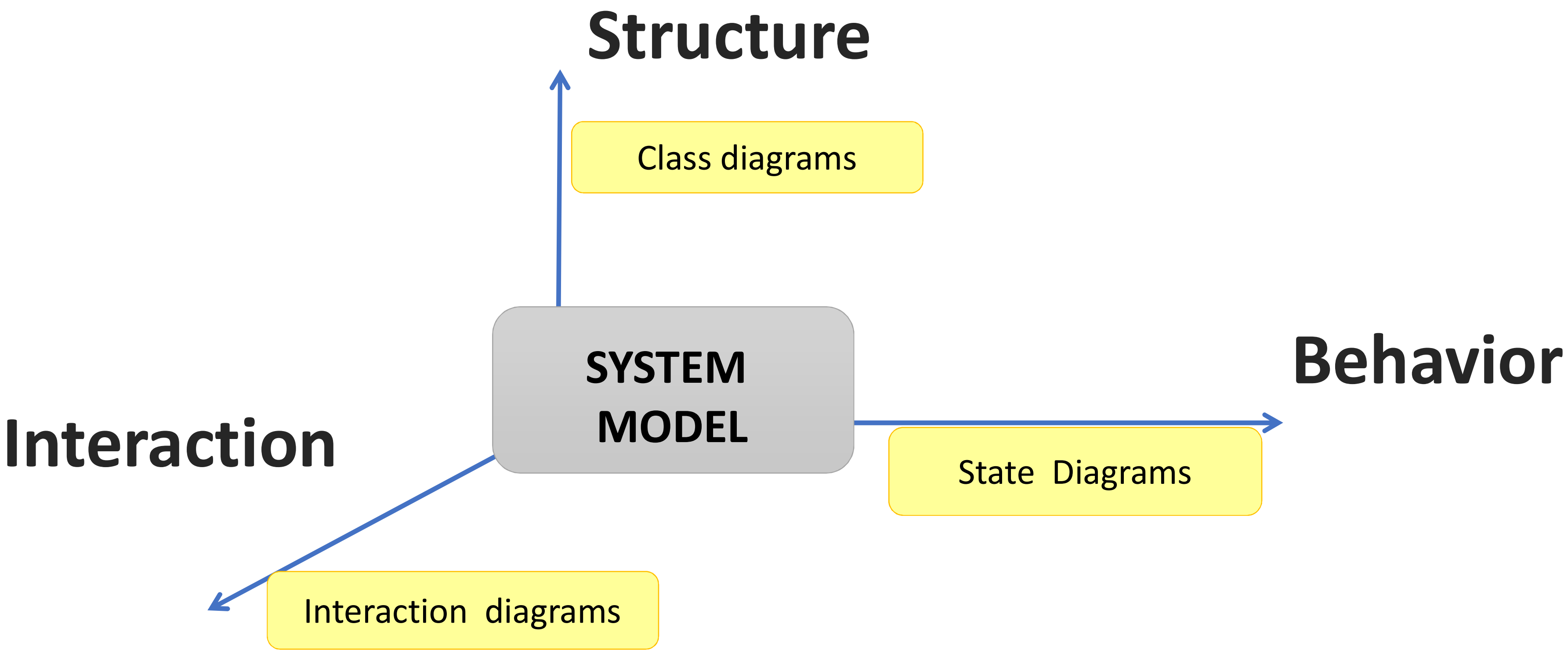
Second generation

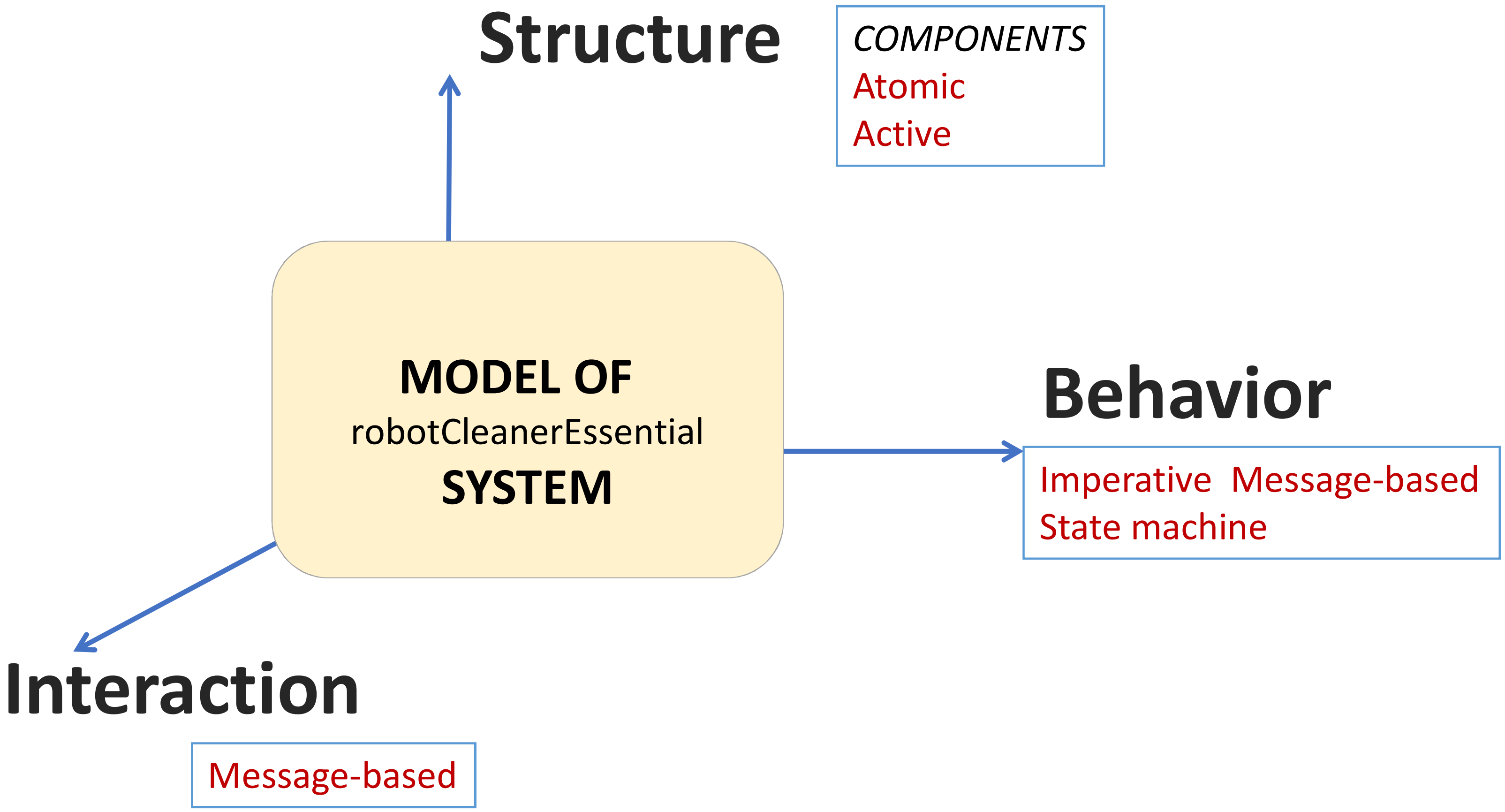
Software as a service (SaaS)

Business model: software purchasing

Third generation

From Internet of people (IoP) to Internet of Things (IoT)


Structure
Class diagrams
SYSTEM MODEL
Behavior
State Diagrams
Interaction
Interaction diagrams



Structure

COMPONENTS
Atomic
Active

MODEL OF
robotCleanerEssential
SYSTEM

Behavior

Imperative Message-based
State machine

Interaction

Message-based