Model Driven Development

Paulo Vieira

24 de Janeiro de 2022

Burj Khalifa Project

- Multi-use development tower
  - Residential
  - Hotel
  - Commercial
  - Office
  - Entertainment
  - Shopping
  - Leisure
  - Parking facilities

Burj Khalifa Project

- Tallest structure ever built by man
- The tower is 828 meters tall
- Total floor area of 460,000 square meters
- 162 floors above grade
- 3 basement levels

[Abdelrazaq, 2010]

Burj Khalifa Project

- One of the most important design criteria

“Mitigating and taming the dynamic wind effects”

How?

"All machinery is derived from nature (…) Let us but consider the connected revolutions of the sun, the moon, and the five planets, without the revolution of which, due to mechanism, we should not have had the alternation of day and night, nor the ripening of fruits. Thus, when our ancestors had seen that this was so, they took their models from nature, and by imitating them were led on by divine facts, until they perfected the contrivances which are so serviceable in our life (…)"

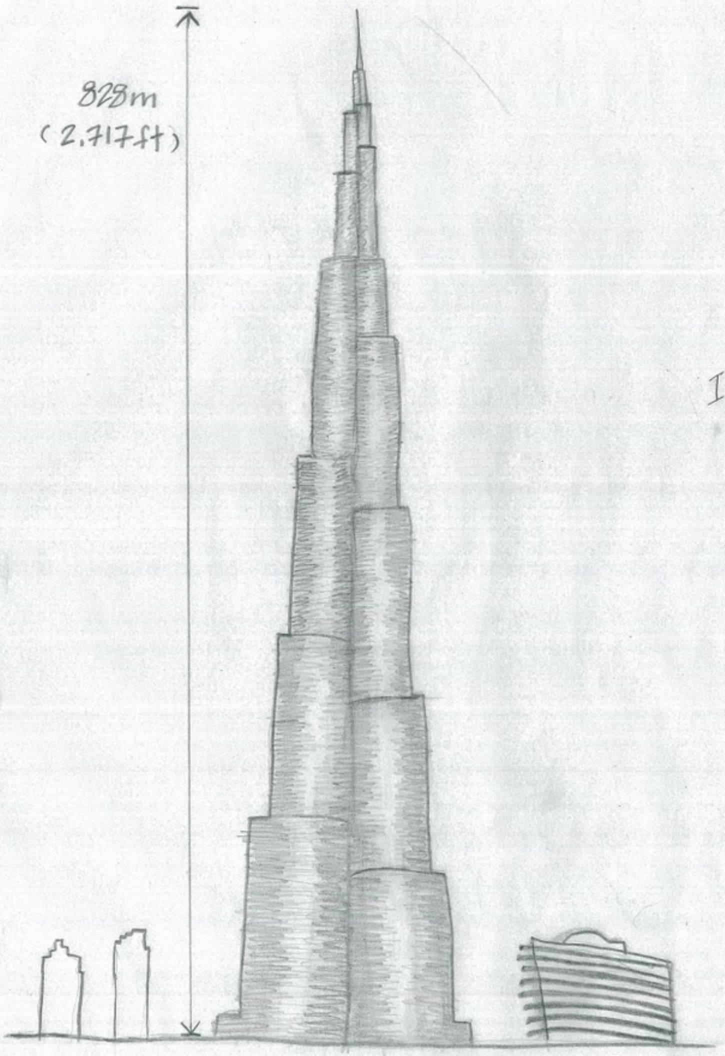
https://www.gutenberg.org/files/20239/20239-h/20239-h.htm

Burj Khalifa Project

- Design derived from geometries of the desert flower *Hymenocallis*

SJ 12 Research on Concept & Design of a building

BURJ KHALIFA / BURJ DUBAI



BEST Tall Building Middle East & Africa award, June 2010

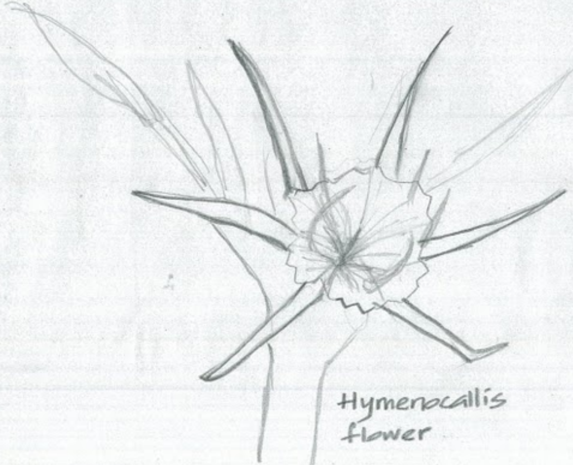
DESIGN

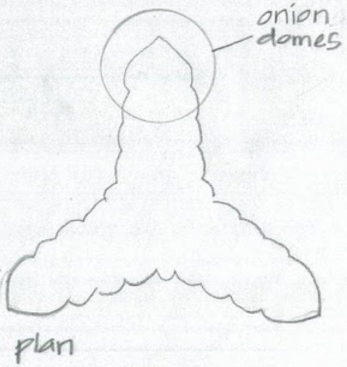
Derived from patterning systems embodied in Islamic Architecture.

Incorporates culture and historical elements particular to the region.

CONCEPT

The Y-shape plan or the triple lobed footprint of the building was inspired by the desert flower Hymenocallis.





http://dyonethamcw.blogspot.com/2011/01/sj-12-research-on-concept-and-design-of.html

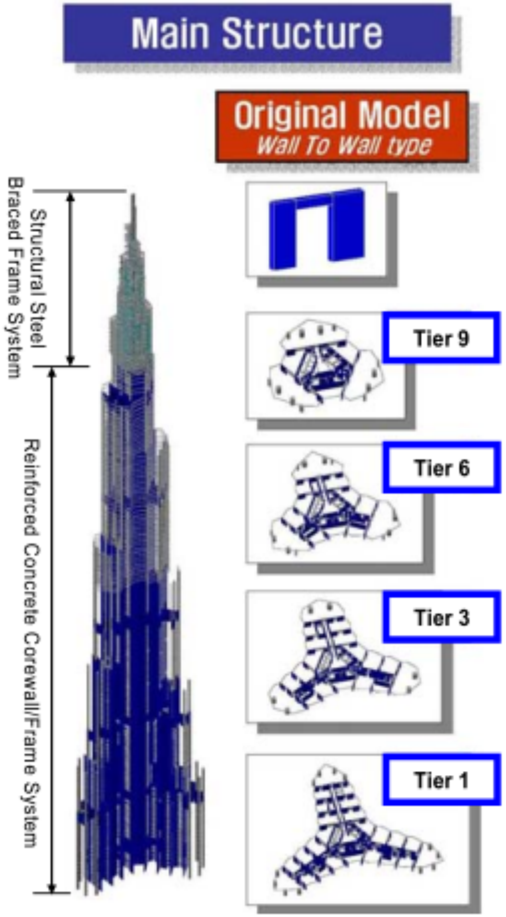
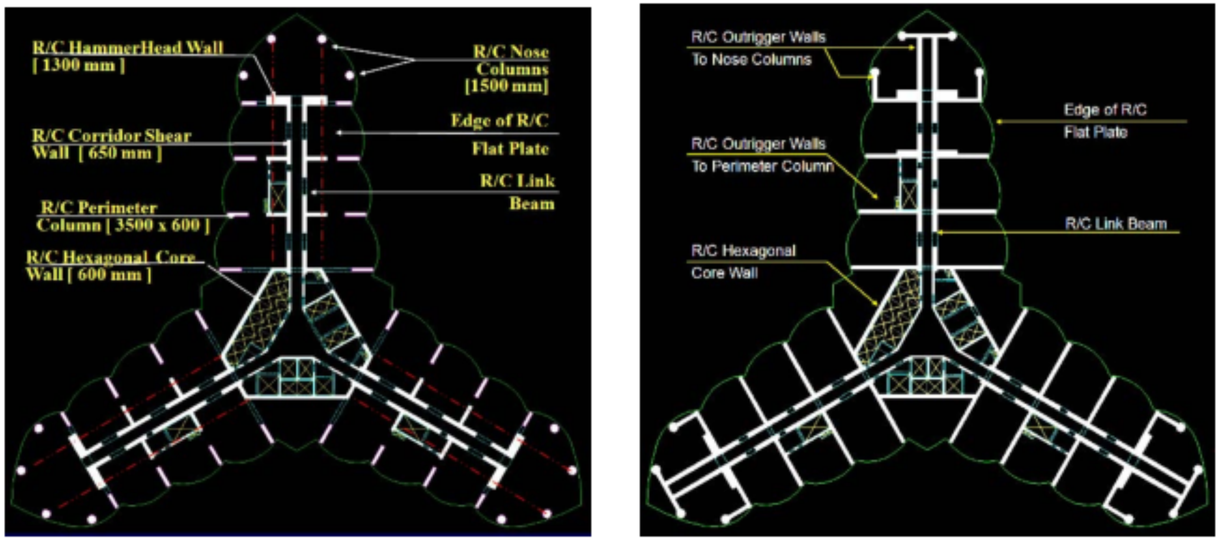

Main Structure
Original Model
Wall To Wall type
Structural Steel Braced Frame System
Reinforced Concrete Corewall/Frame System
Tier 9
Tier 6
Tier 3
Tier 1

FIGURE 2 – LATERAL LOAD SYSTEM


R/C HammerHead Wall [1300 mm]
R/C Nose Columns [1500 mm]
R/C Corridor Shear Wall [650 mm]
Edge of R/C Flat Plate
R/C Link Beam
R/C Perimeter Column [3500 x 600]
R/C Hexagonal Core Wall [600 mm]
R/C Outrigger Walls To Nose Columns
Edge of R/C Flat Plate
R/C Outrigger Walls To Perimeter Column
R/C Link Beam
R/C Hexagonal Core Wall

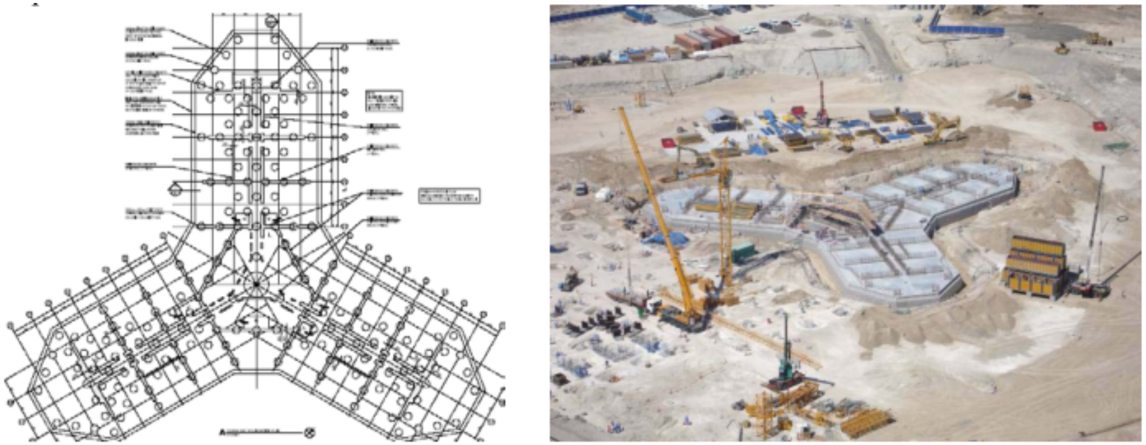
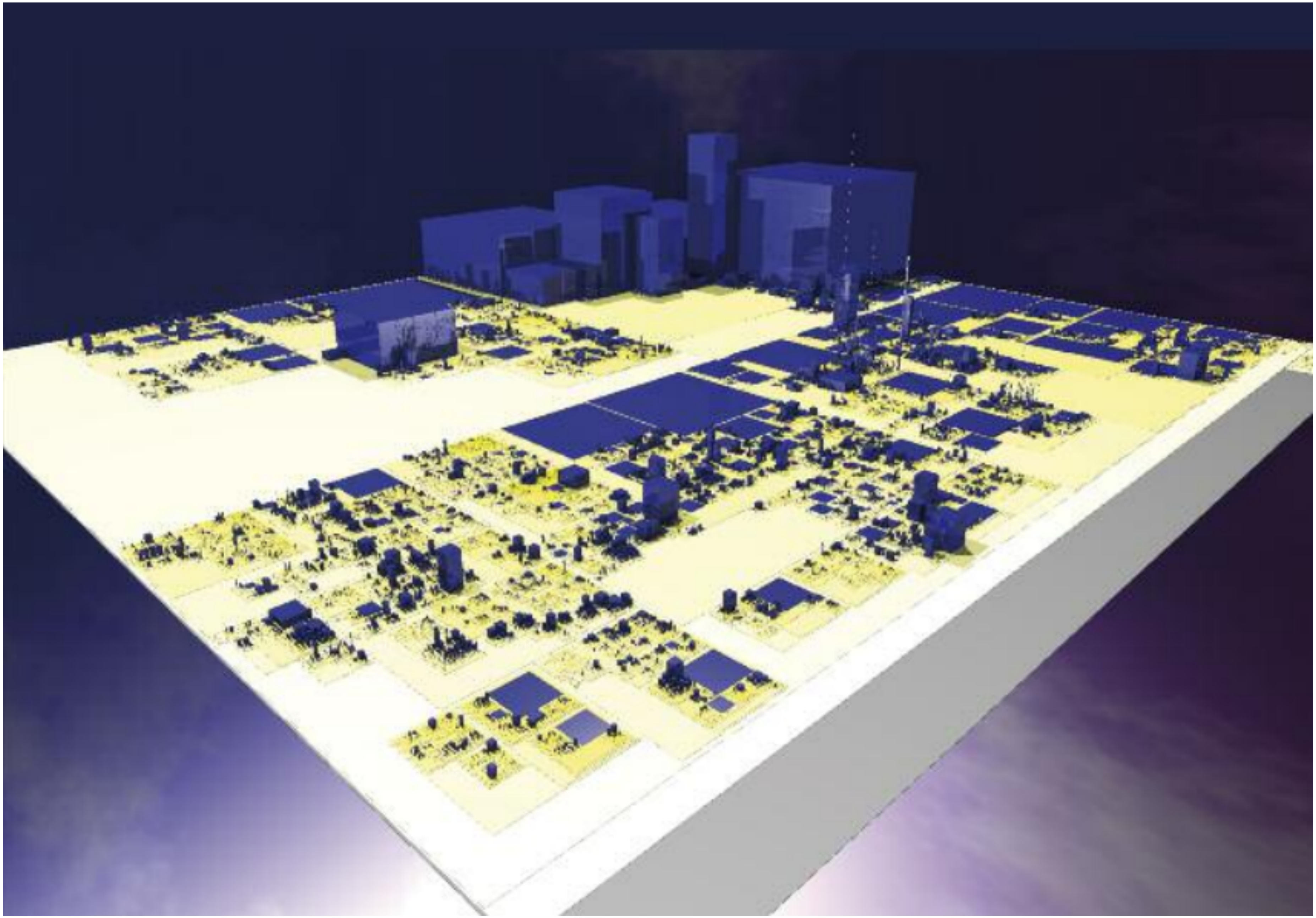


FIGURE 5 – PILE SUPPORTED RAFT FOUNDATION AND COMPLETED RAFT CONSTRUCTION

Engineering Model Characteristics

- Abstraction
  - Deal with complexity
  - Systematic gathering of knowledge
- Understandability
- Accuracy
- Predictiveness
- Inexpensive

- Allow uncertainty/risk reduction



Models and abstraction

- No one would imagine constructing an edifice as complex as a bridge or an automobile without first constructing a variety of specialised system models

- Models help us understand a complex problem and its potential solutions through abstraction

[Selic, 2003]

Models and abstraction

- In software engineering, models are not frequent and normally play a secondary role

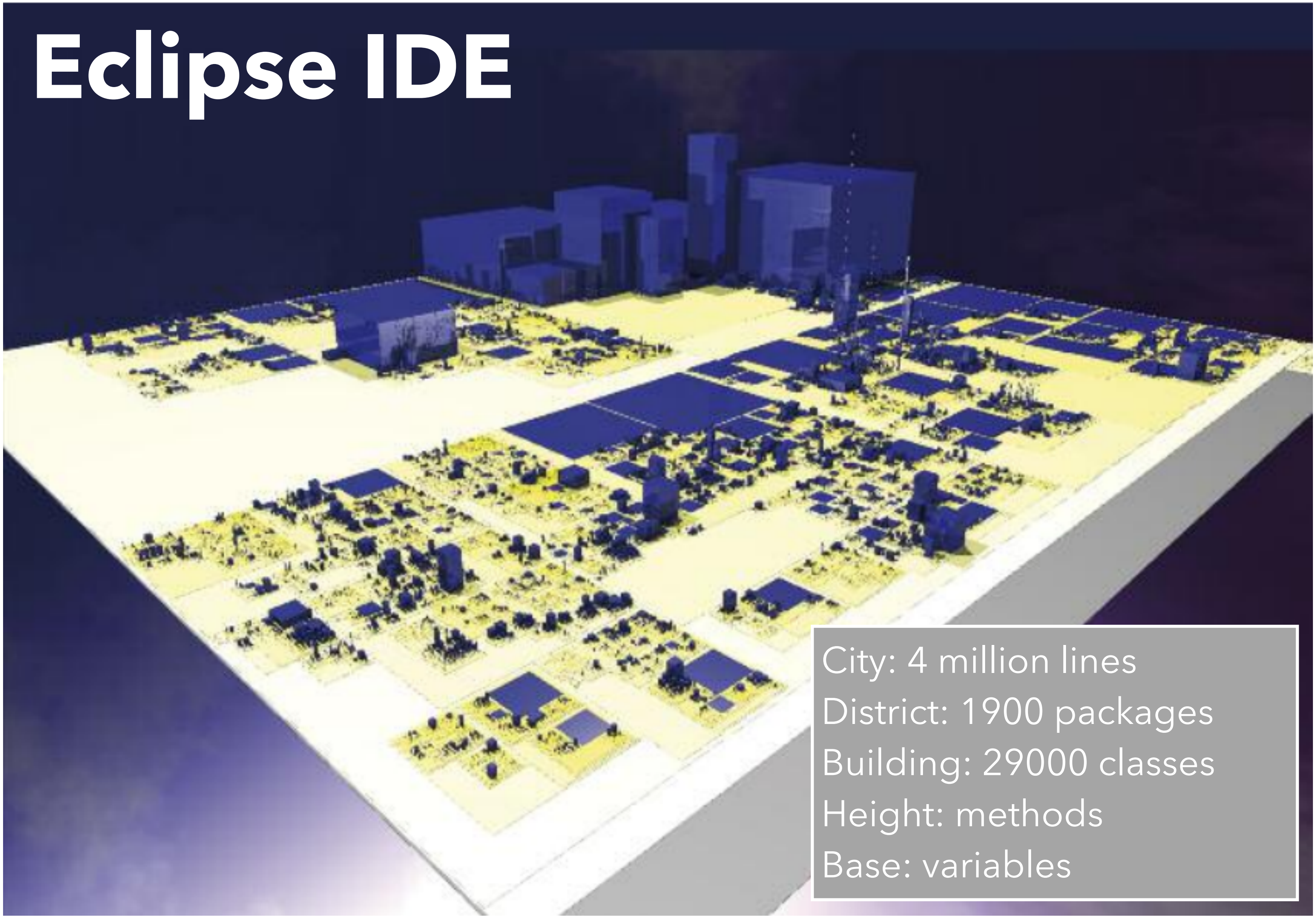
- The potential benefits of using models are significantly greater in software than in any other engineering discipline

Software Engineering

"The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software."

[IEEE, 1990]

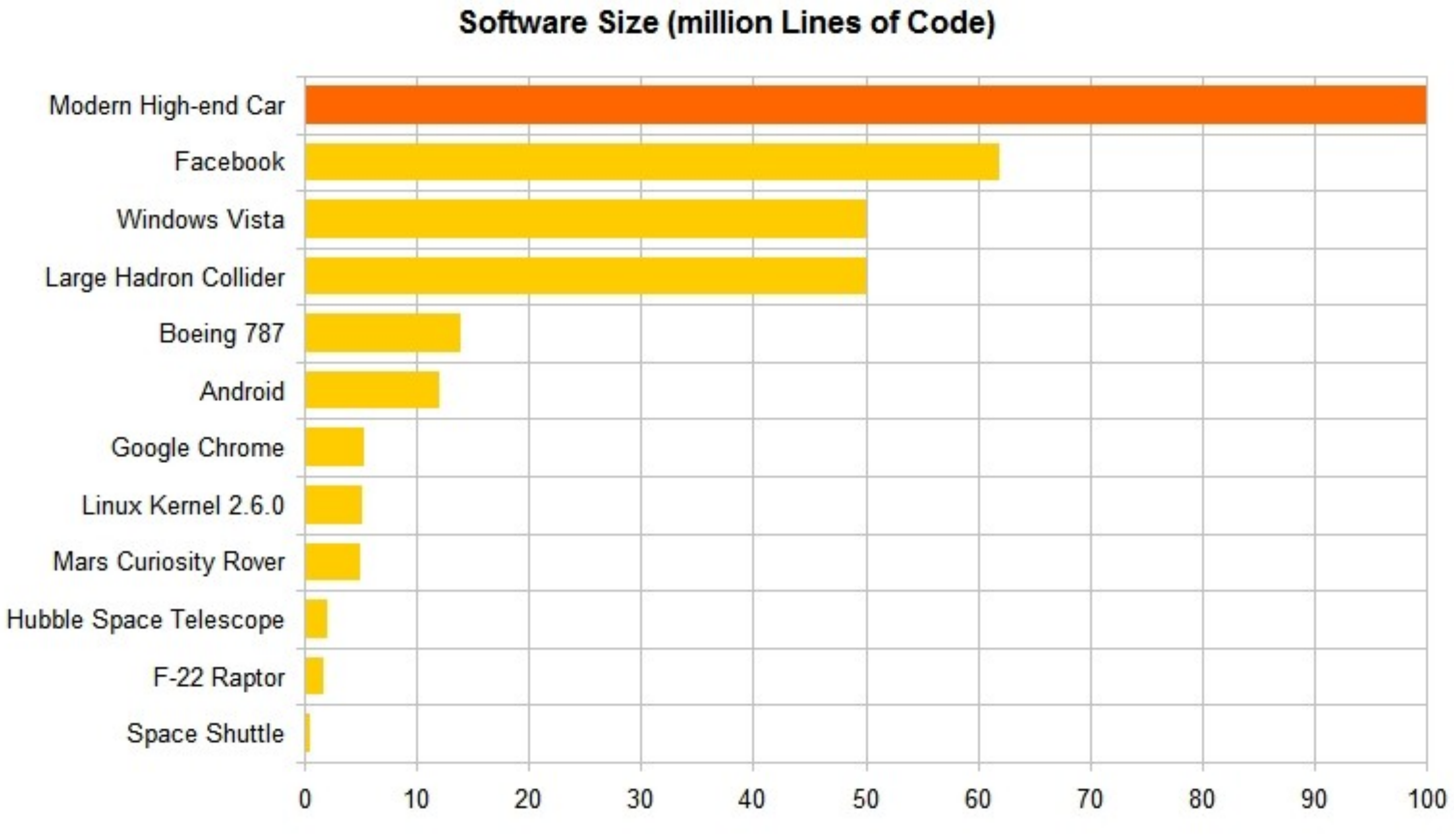
Eclipse IDE



City: 4 million lines
District: 1900 packages
Building: 29000 classes
Height: methods
Base: variables

https://ercim-news.ercim.eu/images/stories/EN88/EN88-web.pdf

Software size



https://www.linkedin.com/pulse/20140626152045-3625632-car-software-100m-lines-of-code-and-counting

Why?

- Software engineering is in the unfortunate position of being a new and relatively immature branch of engineering of which much is expected

- Software users and developers are demanding systems whose complexities exceed our ability to build them
  - Due to the relative ease of writing code - there is no metal to bend or heavy material to move
  - Compelled by relentless market pressures

Why?

- Designing large software systems is one of the big technical challenges of our time
- The scope and complexity of software have now reached dimensions that push all established approaches and methods for its development to its limits
- Solution: an appropriate abstraction leading to decreased complexity and improved controllability of software systems.

> “there are still many questions that need to be answered by research”

[Rumpe, 2016]

Model-Driven Development (MDD)

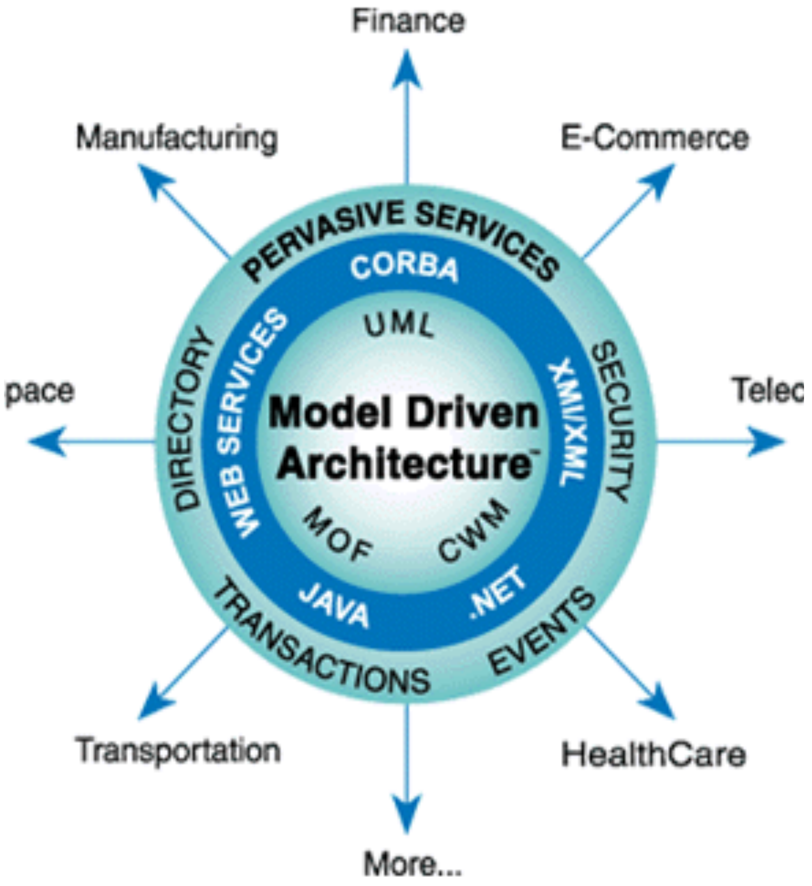
- Software development’s primary focus and products are models
    - Instead of computer programs

- Express models using concepts much closer to problem domain
    - Much less bound to underlying implementation technology

Model-Driven Architecture

- Provided by the Object Management Group (OMG)
    - OMG is a consortium of software vendors and users from industry, government, and academia.

- Conceptual framework for defining a set of standards in support of MDD



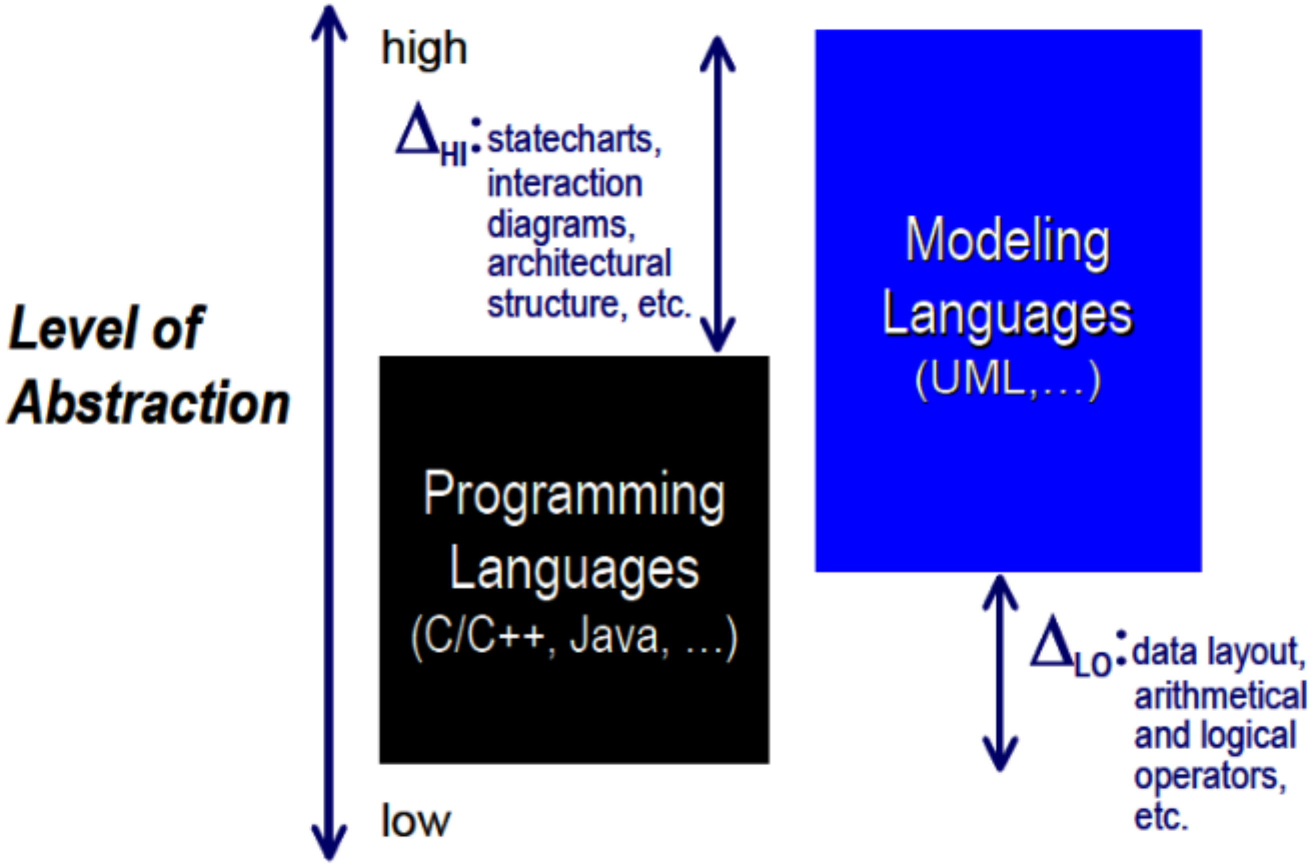
https://www.omg.org/mda/index.htm

Model-Driven Architecture

- Standardisation provides a significant impetus for further progress because:
  - it codifies best practices
  - enables and encourages reuse
  - facilitates interworking between complementary tools
  - encourages specialisation, which leads to more sophisticated and more potent tools

- A key MDA standard is the Unified Modelling Language (UML)

Unified Modelling Language (UML)



high
Δ_{HI}: statecharts, interaction diagrams, architectural structure, etc.
Level of Abstraction
Programming Languages (C/C++, Java, ...)
Modeling Languages (UML,...)
Δ_{LO}: data layout, arithmetical and logical operators, etc.
low

Why UML?

- To effectively model a system, we need a language with which the model can be described
- With a formal modelling language, the language is **abstract** yet just as **precise** as a programming language

[Miles, 2006]

Why UML?

- A modelling language can be anything that contains
    - A way of expressing the model (**notation**)
    - A description of what the notation means (**meta-model**)

[Miles, 2006]

Why UML?

- Every approach has advantages and disadvantages.
- Six main advantages of UML:
    1. It’s a formal language
    2. It’s concise
    3. It’s comprehensive
    4. It’s scaleable
    5. It’s built on lessons learned
    6. It’s the **standard**

[Miles, 2006]

“Degrees” of UML

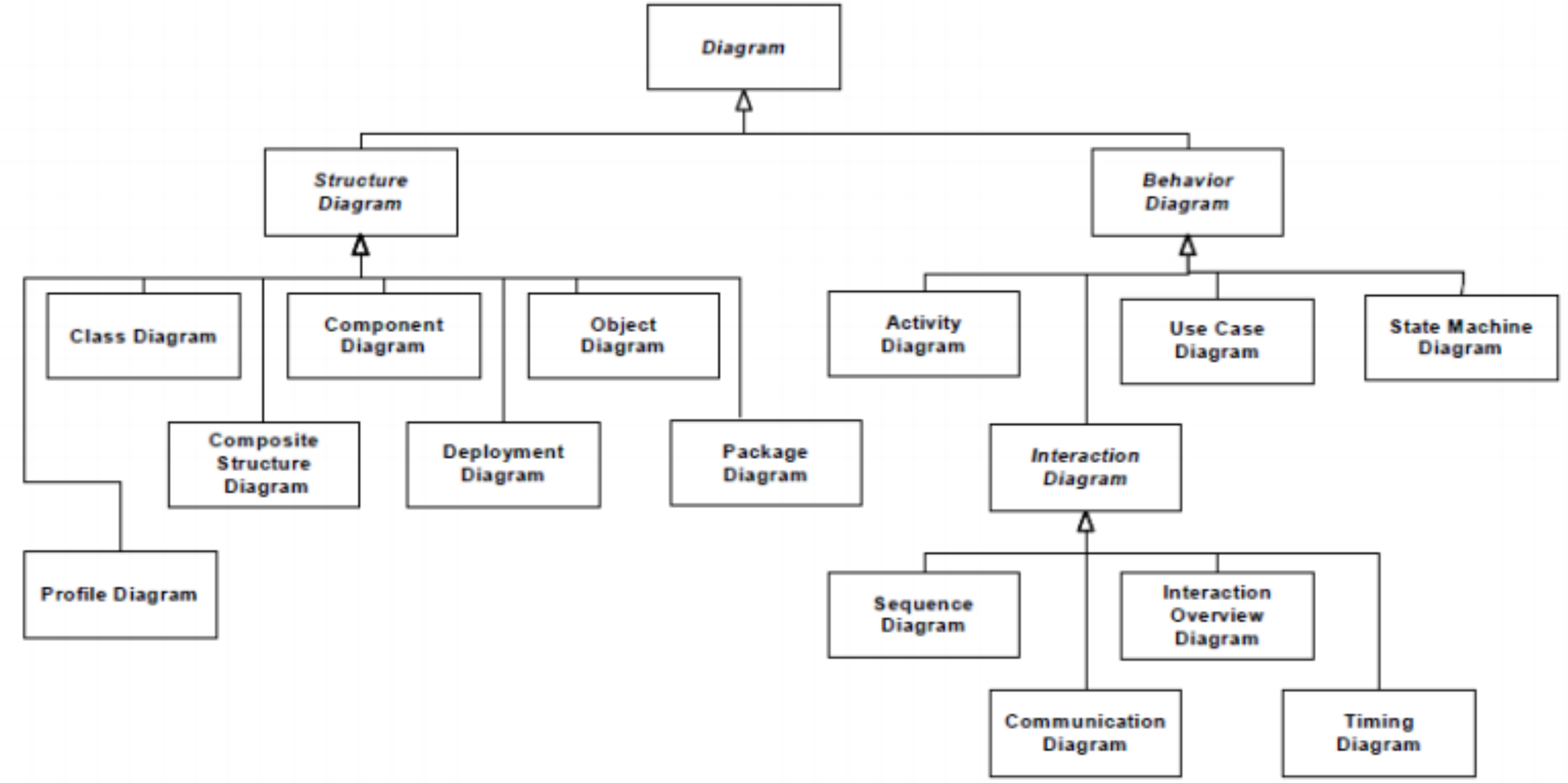
- UML as a sketch
  - Make brief sketches to convey key points

- UML as a blueprint
  - Detailed specification of a system

- UML as a programming language
  - Every aspect of a system is modelled
  - UML model to executable code

[Miles, 2006]

UML Diagrams



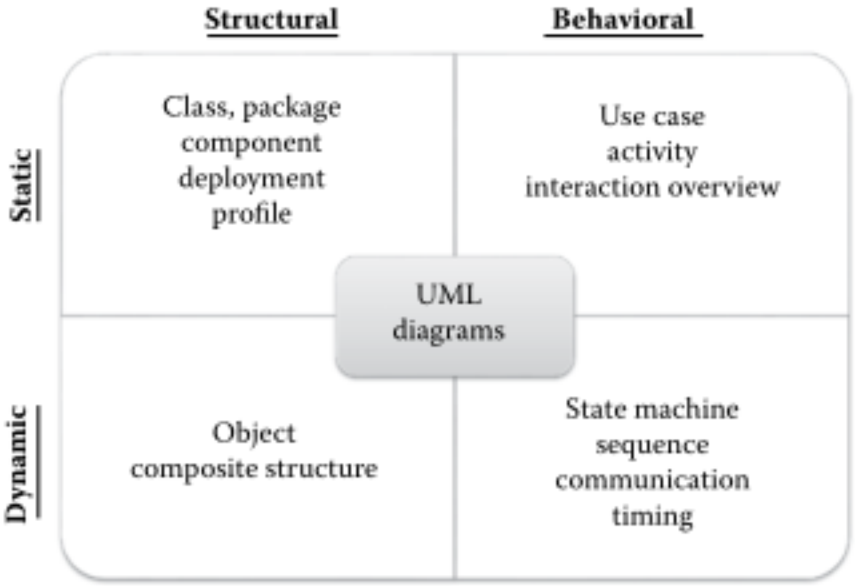
https://www.omg.org/spec/UML/2.5/PDF

Nature of UML Diagrams

- Structural versus behavioural
  - The structural aspect of a diagram illustrates the way a system is organised
    - e.g., how classes relate to each other in a class diagram
  - The behavioural aspect models the flow of the system.
    - e.g., shows the way in which a user interacts with the system—through a use case or an activity diagram
- Static versus dynamic
  - Depicts the time dependency of the model
    - A diagram with no concept of time or movement is static
    - One that shows changes in time is considered dynamic

[Unhelkar, 2017]

Nature of UML Diagrams



	Structural	Behavioral
Static	Class, package component deployment profile	Use case activity interaction overview
Dynamic	Object composite structure	State machine sequence communication timing

[Unhelkar, 2017]

Kruchten's "4+1" view model

- Breaks a model into a set of views
- Each view captures a specific aspect of a system

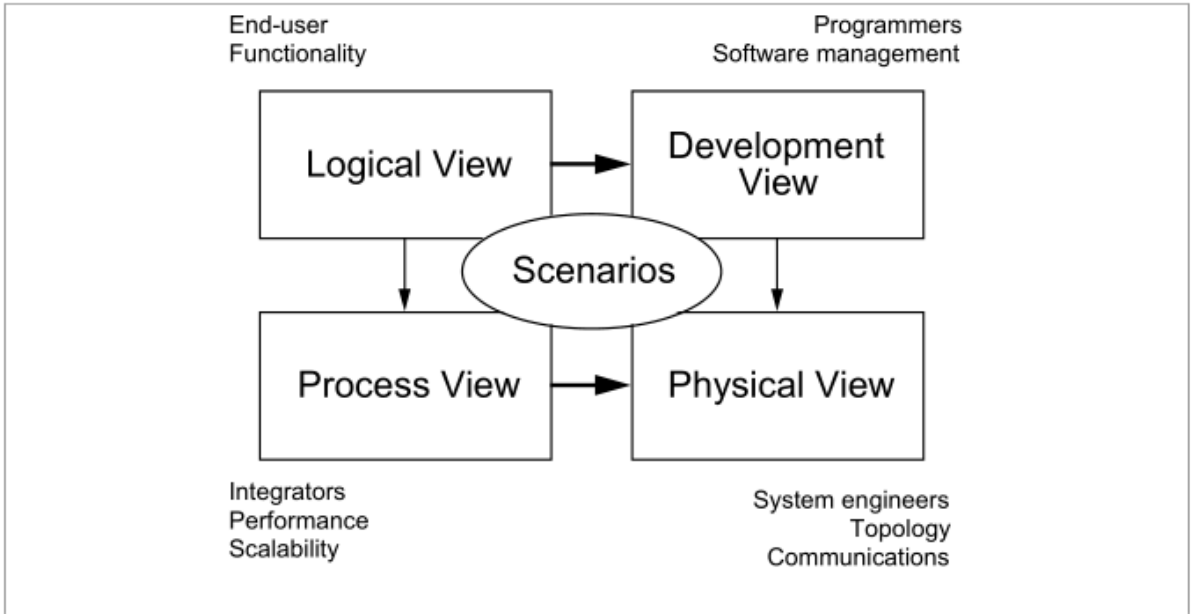


Figure 1 — The "4+1" view model

[Krutchen, 1995]

Logical View

The Object-Oriented Decomposition

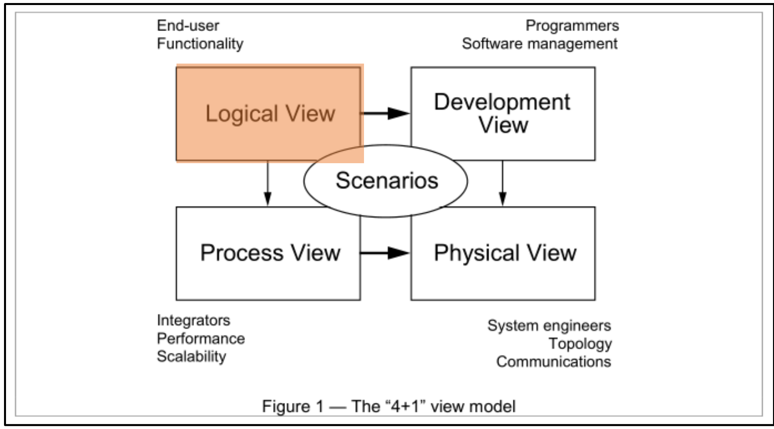


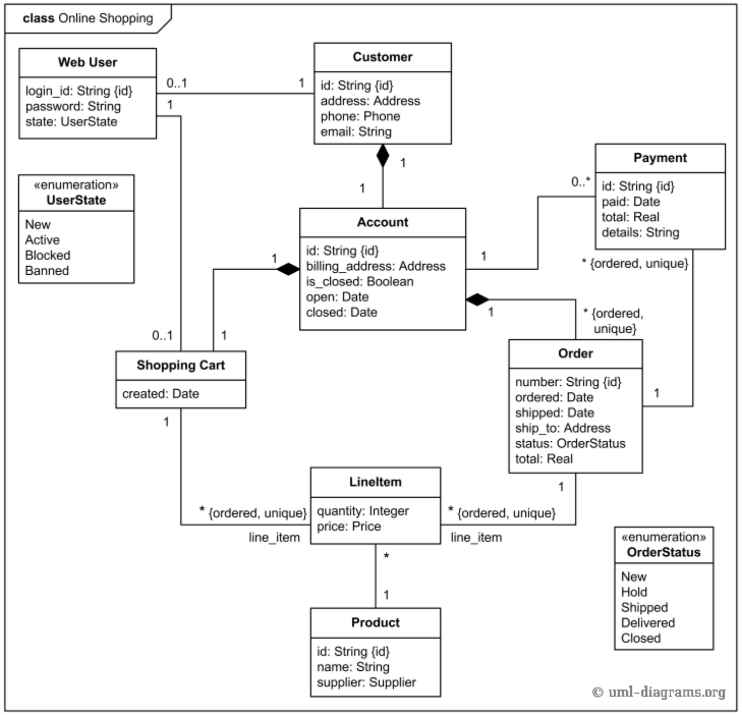
Figure 1 — The "4+1" view model

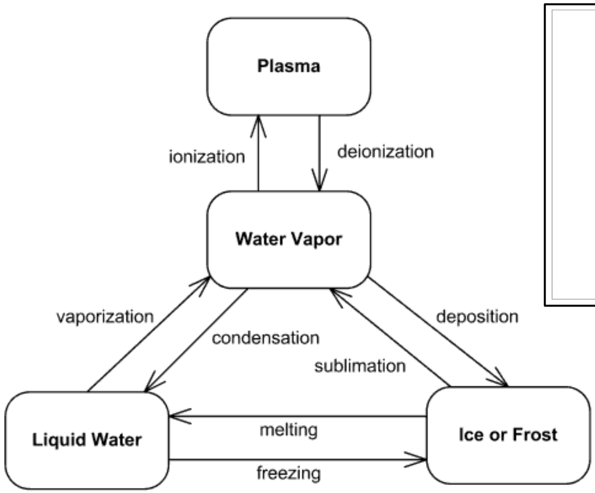
- Supports the **functional requirements**
  - Which services should the system provide to its users

- Abstract description of system's parts
- Model the parts of the system
- How the parts interact with each other

- Typically:
  - class,
  - object,
  - state machine, and
  - interaction diagrams

[Miles, 2006]; [Krutchen, 1995]





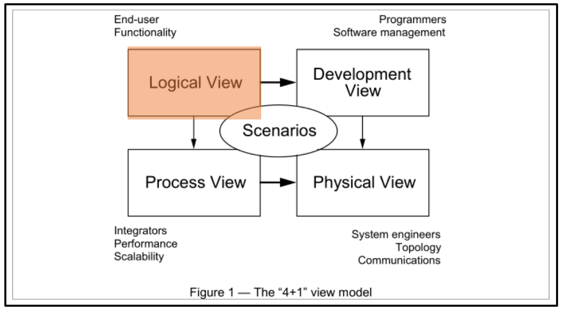
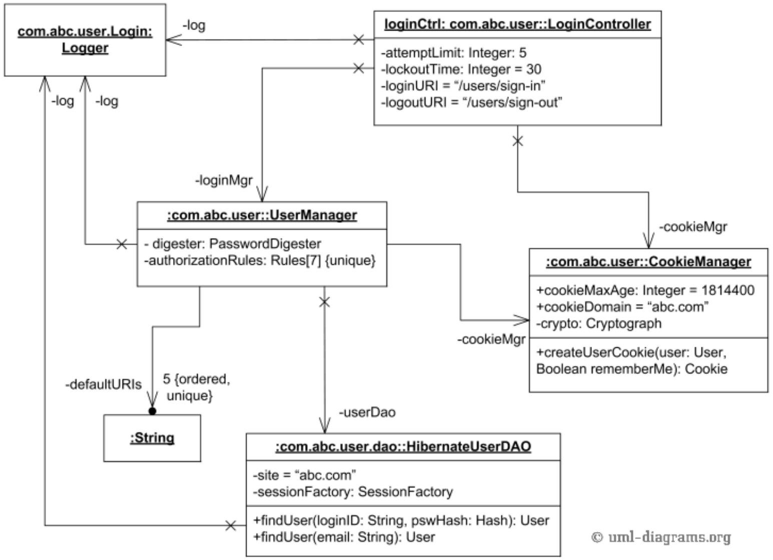
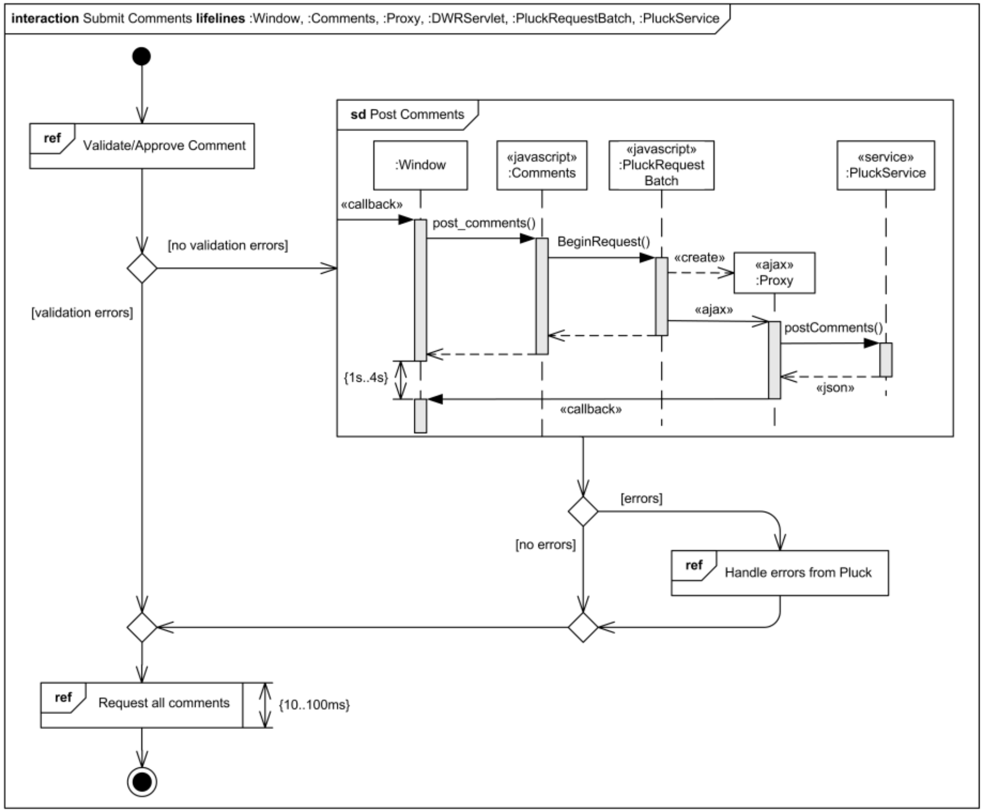


Figure 1 — The "4+1" view model


com.abc.user.Login: Logger
-log
loginCtrl: com.abc.user::LoginController
-attemptLimit: Integer: 5
-lockoutTime: Integer = 30
-loginURI = "/users/sign-in"
-logoutURI = "/users/sign-out"
-log
-log
-loginMgr
-cookieMgr
:com.abc.user::UserManager
- digester: PasswordDigester
-authorizationRules: Rules[7] {unique}
:com.abc.user::CookieManager
+cookieMaxAge: Integer = 1814400
+cookieDomain = "abc.com"
-crypto: Cryptograph
+createUserCookie(user: User, Boolean rememberMe): Cookie
-cookieMgr
-defaultURIs
5 {ordered, unique}
:String
-userDao
:com.abc.user.dao::HibernateUserDAO
-site = "abc.com"
-sessionFactory: SessionFactory
+findUser(loginID: String, pswHash: Hash): User
+findUser(email: String): User
© uml-diagrams.org



https://www.uml-diagrams.org

Process View

The Process Decomposition

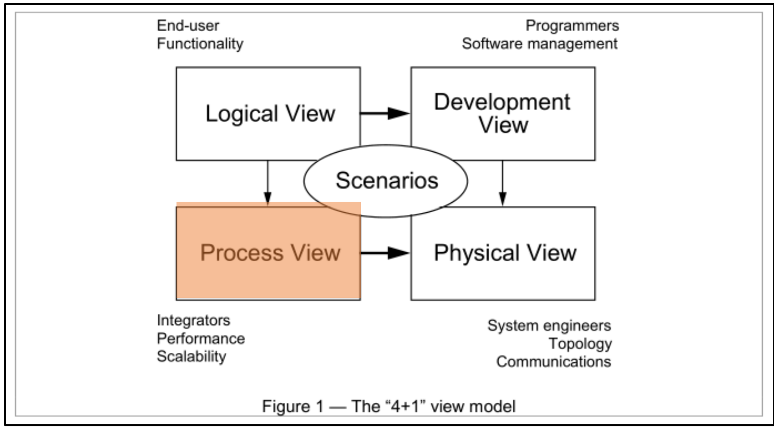


Figure 1 — The "4+1" view model

- Describes the processes within a system
- Takes into account some non-functional requirements
    - e.g., performance and availability
- Address concurrency and distribution, system's integrity, and fault-tolerance
- Helpful to visualize what must happen within a system

- Typically:
    - Activity diagrams

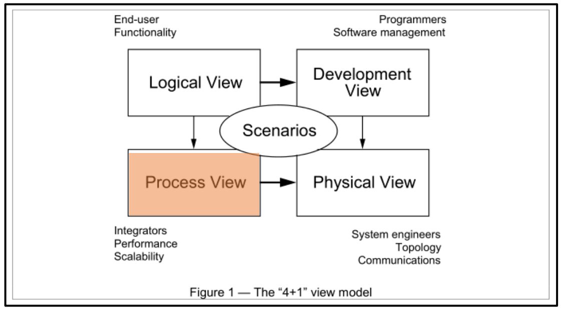
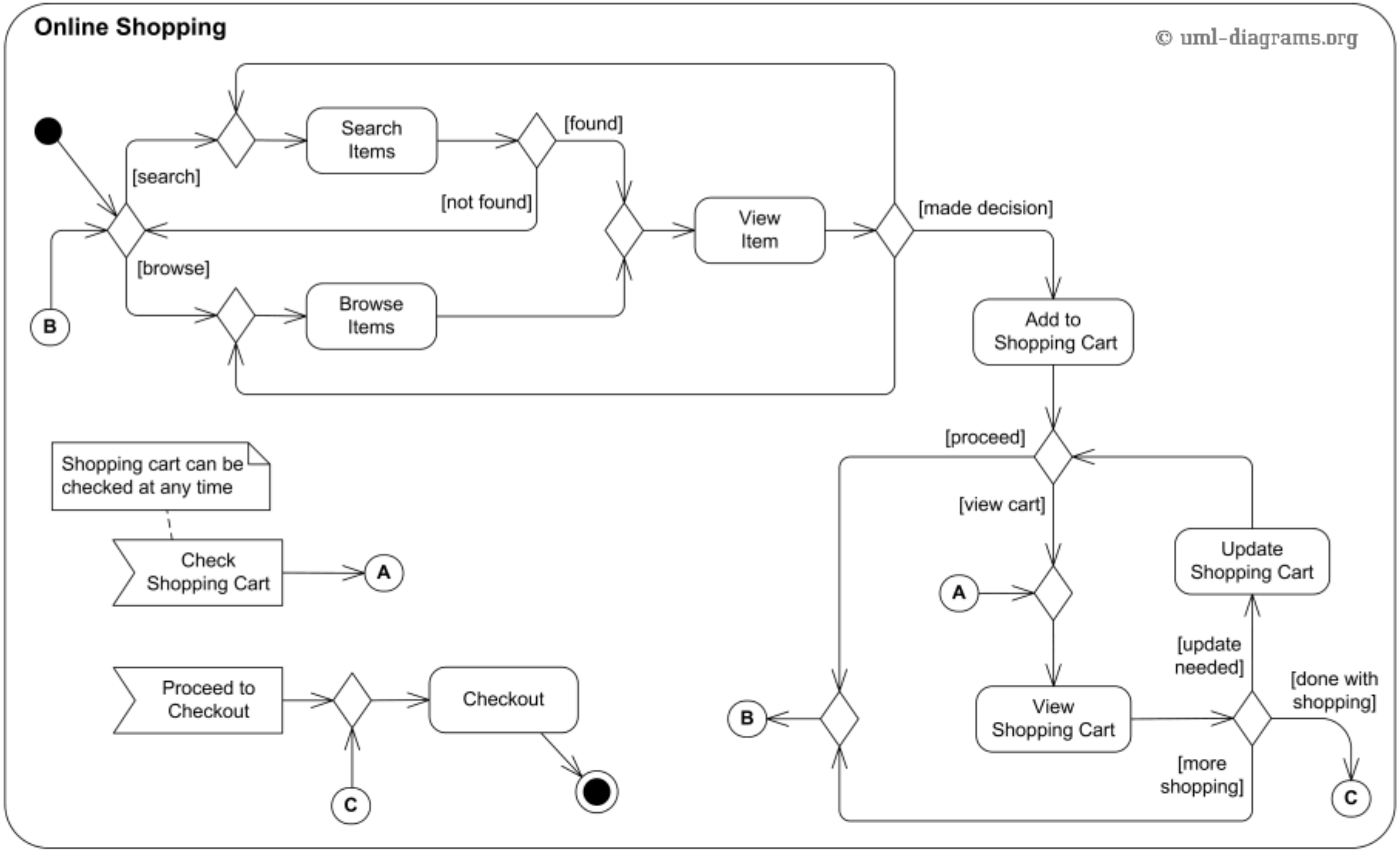


Figure 1 — The "4+1" view model



https://www.uml-diagrams.org

Development View

Subsystem decomposition

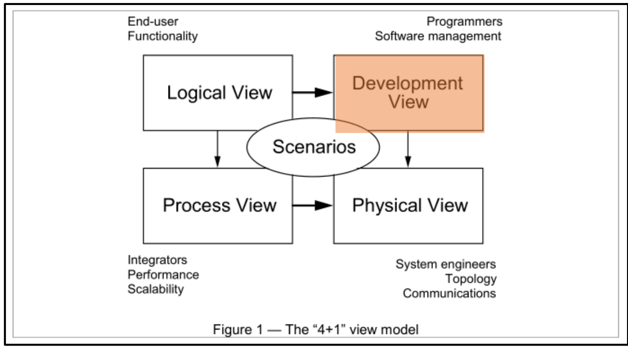
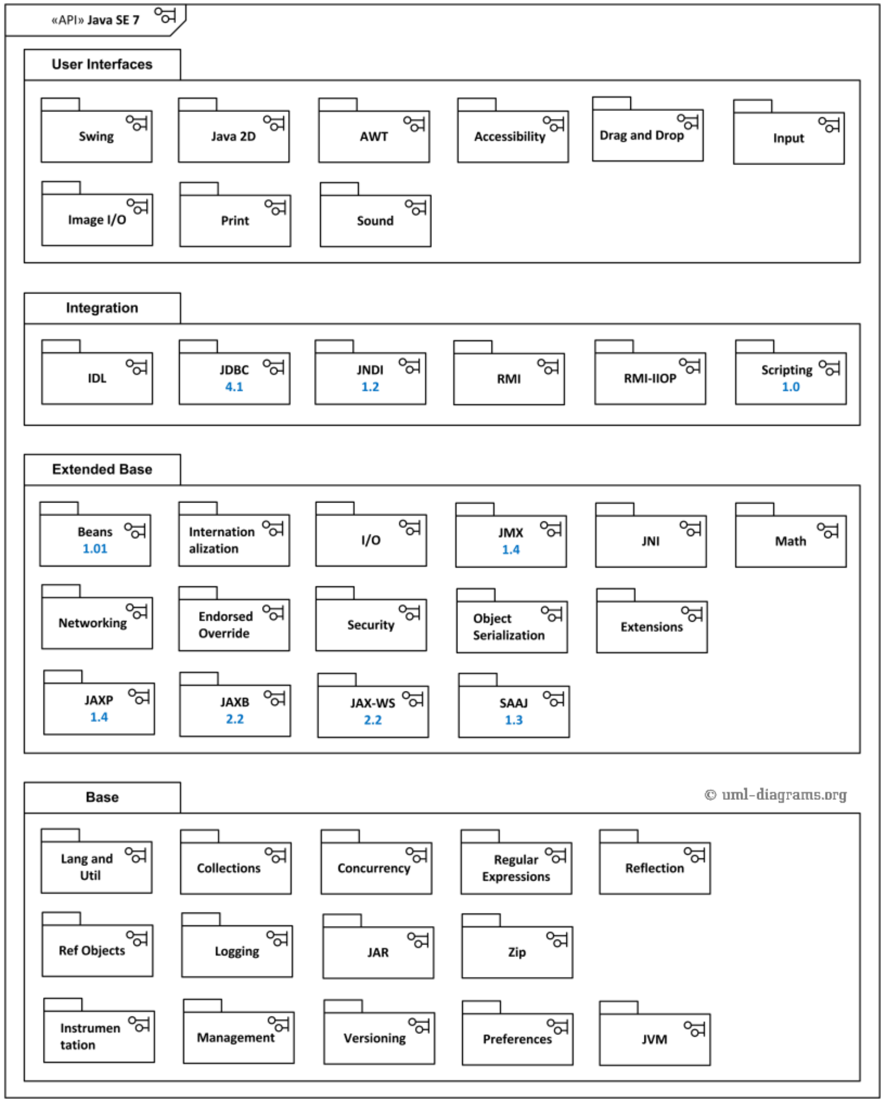


Figure 1 — The "4+1" view model

- Describes how the systems' parts are organised into modules and components
  - Subsystems are organised in a hierarchy of layers
  - Each layer provides a narrow and well-defined interface to the layers above it
- Useful to manage the layers within the systems' architecture

- Typically:
  - Package
  - Component



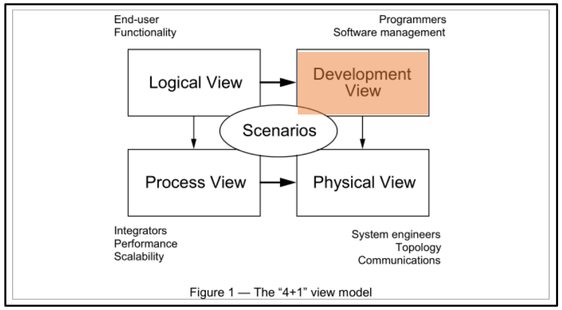
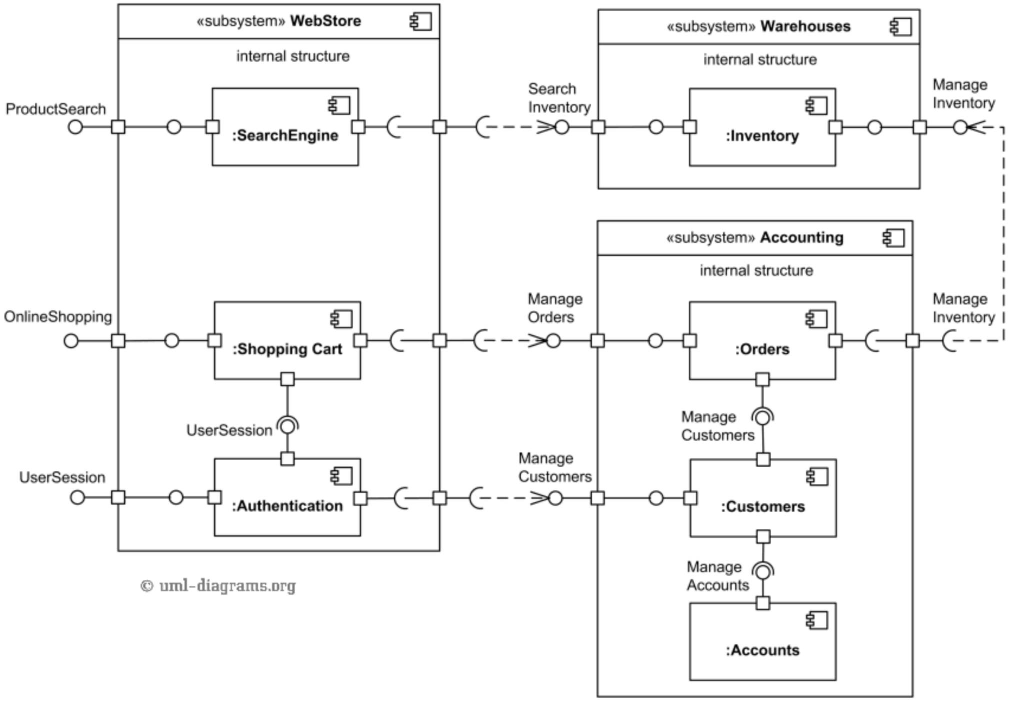


Figure 1 — The "4+1" view model



https://www.uml-diagrams.org

Physical View

Mapping the software to the hardware

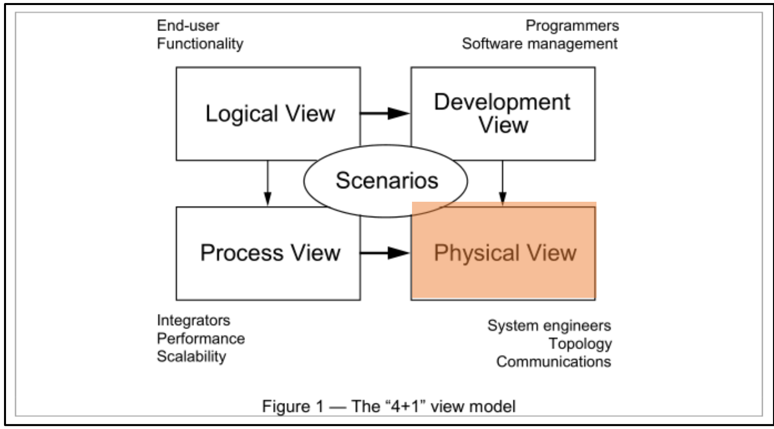


Figure 1 — The "4+1" view model

- Takes **non-functional requirements** into account primarily
  - E.g., availability, reliability, performance, and scalability
- How the abstract parts map into the final deployed system
- Typically:
  - Deployment diagrams

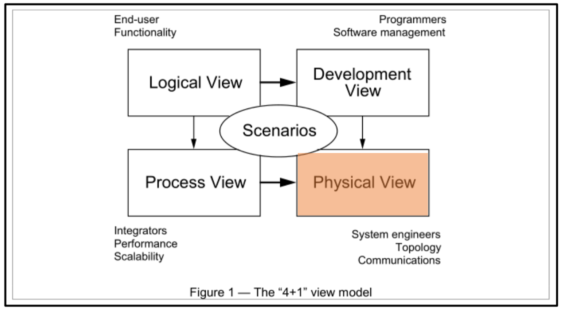
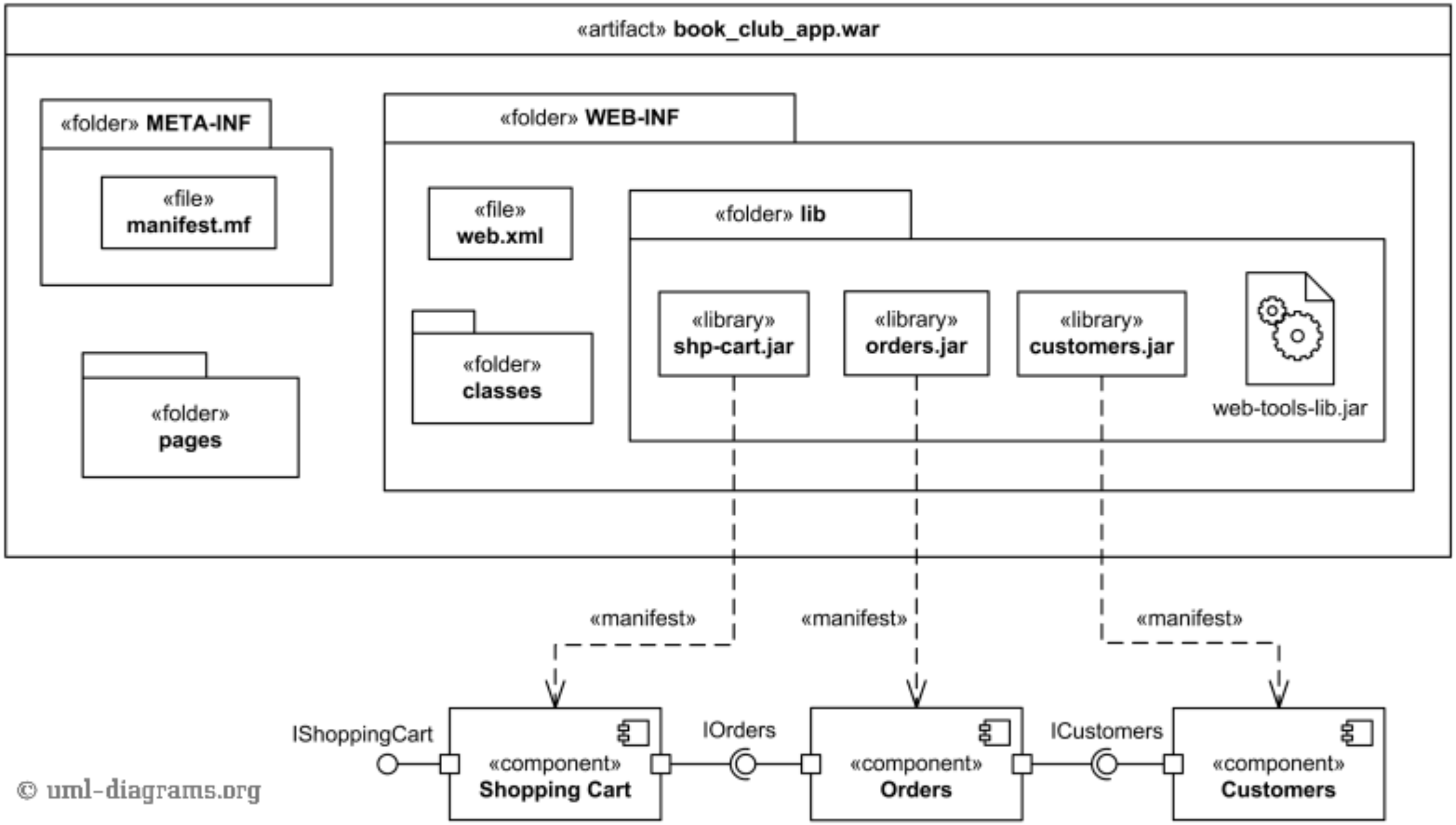


Figure 1 — The "4+1" view model



https://www.uml-diagrams.org

Scenarios

Putting it all together

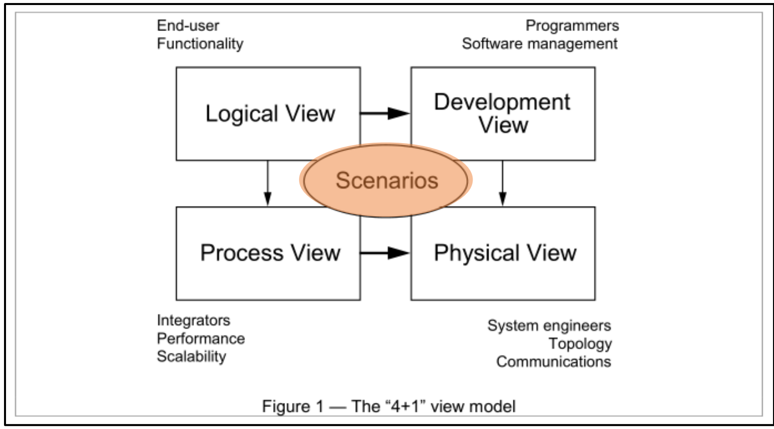
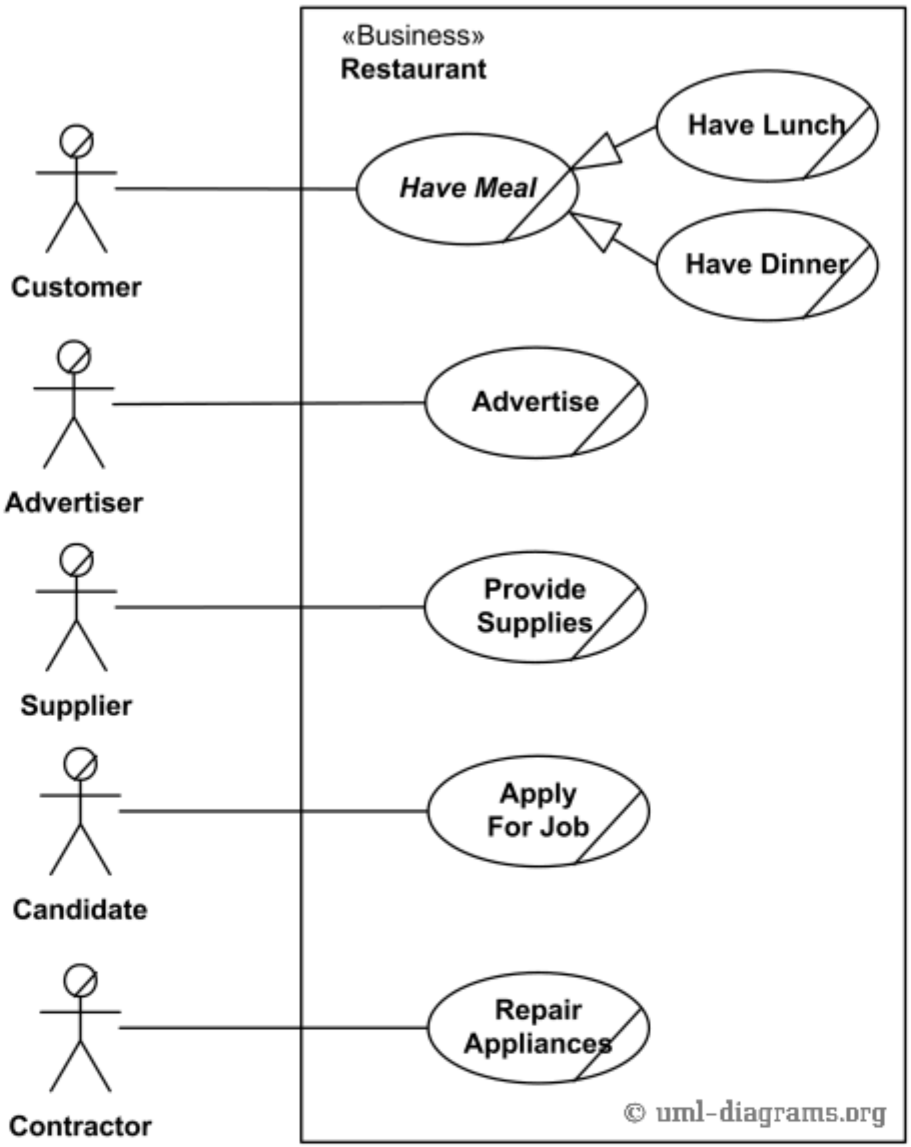


Figure 1 — The "4+1" view model

- Describe the functionality of the system being modelled from the outside world's perspective
- Guides all the other views
- Instances of **general use cases**
- Abstraction of the most important requirements

- Typically:
  - Use case
  - Descriptions
  - Overview



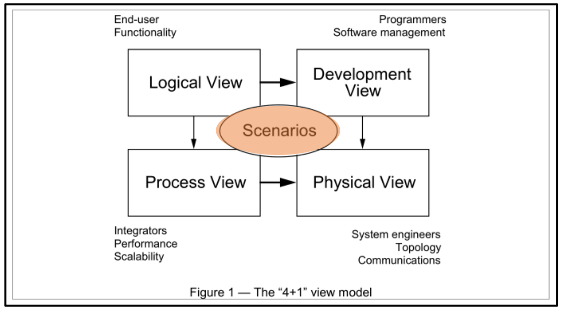


Figure 1 — The "4+1" view model

https://www.uml-diagrams.org

References

- Abdelrazaq, Ahmad. (2010). "Design and Construction Planning of the Burj Khalifa, Dubai, UAE"
- Kruchten, P. B. (1995), "The 4+1 View Model of architecture" in IEEE Software, vol. 12, no. 6, pp. 4 2-50
- IEEE 610.12-1990 (1990), “Standard Glossary of Software Engineering Terminology”.
- Miles, Russ and Hamilton, Kim. (2006). "Learning UML 2.0". O’Reilly Media, Inc.
- Pressman, Roger. (2009) ”Software Engineering: A Practitioner’s Approach” (7th. ed.). McGraw-Hill, Inc., USA.
- Rumpe, Bernhard, (2016) “Modeling with UML: Language, Concepts, Methods”, Springer International Publishing Switzerland
- Selic, Bran. (2003) "The Pragmatics of Model-Driven Development."
  IEEE Software 20, 5 (September 2003), pp. 19–25.
- Unhelkar, Bhuvan. (2017) “Software Engineering with UML” (1st. ed.). Auerbach Publications, USA.