INTRODUCTION TO CONCEPTUAL MODELING

BY PANKAJ KAMTHAN

1. INTRODUCTION

It is a great misfortune in software development that the word 'model' has become so devalued. In common usage it means no more than 'description'.
— Michael A. Jackson

This document explores certain basic elements of modeling, especially conceptual modeling.

CONVENTION/NOTATION

In this document, the term 'building', 'constructing', 'crafting', 'creating', 'diagramming', 'drawing', 'illustrating', and 'modeling' are considered synonymous, and are used interchangeably[1], unless otherwise stated.

2. DEFINITION OF MODEL

The **model theory** laid out a number of characteristics inherent to any model [Stachowiak, 1973; Seidl, Scholz, Huemer, Kappel, 2015, Preface; Weinert, 2016, Chapter 4].

[1] The term 'visualizing' is **not** synonymous with any of these terms. Indeed, 'visualization' is about forming a **mental image** of a concept, and not necessarily about a graphical form. For example, a visually-disabled person could 'visualize', and the visualization may not correspond to any known graphical form.

THING

Definition [Thing]. An **entity (object), property (attribute),** or **phenomenon (event)** in the world.

For example, a thing could be a process, a product, or even people[2].

MODEL

There are a number of possible definitions of a model [Kleppe, Warmer, Bast, 2003; Seidewitz, 2003; Booch, Rumbaugh, Jacobson, 2005], including the following:

Definition [Model]. A simplified description of a thing from a particular viewpoint.

Definition [Model]. A simplified representation of a thing from a particular viewpoint.

Definition [Model]. A simplification of reality.

These definitions are **relatively weak** for a number of reasons, including that a thing is treated as **second-class**[3], or that a model is **mistakenly equated** to a description or representation [Garrido, 2016, Section 1.3; Weinert, 2016, Section 4.3].

The foregoing definitions can be improved and expressed as the following:

Definition [Model]. A simplification, with respect to some goal, of a thing.

For practical purposes, a model is a set of **statements** about some system under study [O'Keefe, 2010].

[2] In human-computer interaction, in general, and usability engineering, in particular, 'modeling of people' is commonly referred to as **user modeling.**

[3] It should be possible to model a thing other than reality. Furthermore, it is not necessary that any simplification of a thing should qualify as a model. In other words, the simplification must have some purpose. The presence of a goal makes that purpose explicit.

REMARKS

- The term ‘model’ is derived from the Latin “modulus” and Italian “modello”.

- There are misconceptions about conceptual model. In particular, a conceptual model is **not** a design metaphor [Henderson, Johnson, 2012, Section 3.2].

3. CHARACTERISTICS OF A MODEL

The **Stachowiak’s Model Theory** [Schalles, 2013, Section 2.1] suggests that a ‘candidate’ must satisfy the following criteria for it to be considered as a model:

1. **Mapping Criterion:** There must be an ‘original’ that is mapped to the model.

2. **Reduction Criterion:** The model must exhibit some, but not necessarily all, properties of the original.

3. **Pragmatic Criterion:** The model must be able to replace the original for some purpose, that is, the model is useful.

4. HISTORY AND EVOLUTION OF CONCEPTUAL MODELING

The topic of conceptual modeling has a long and interesting history, going back to late **1960s** and early **1970s** [Schalles, 2013, Section 2.2].

In the late **1960s**, the **programming language Simula** introduced a **model for a computer program** through many of the concepts of object-orientation—object, class and subclass, and method—considered a commonplace in current object-oriented programming languages.

In the early **1970s**, the **relational data model** separated logical organization of data from its physical organization, and paved the way for the theory of relational databases.

5. EXAMPLES OF MODELS

The following are common examples of models in science and engineering, some of which are discussed in detail subsequently:

- Map of Metro Route
- Blueprint of Airport Architecture
- Computer Simulation of a Moving Car
- Illustration of Assembling Furniture
- Magnetic Resonance Imaging (MRI) Scan of Human Brain
- A Finite State Automaton for a Computer Program
- An Upper Ontology of the Software Measurement Domain
- $f(x) = \mathrm{r}x(1 - x)$, $0 \leq x \leq 1$, $0 < \mathrm{r} \leq 4$

5.1. MODELS IN SCIENCE

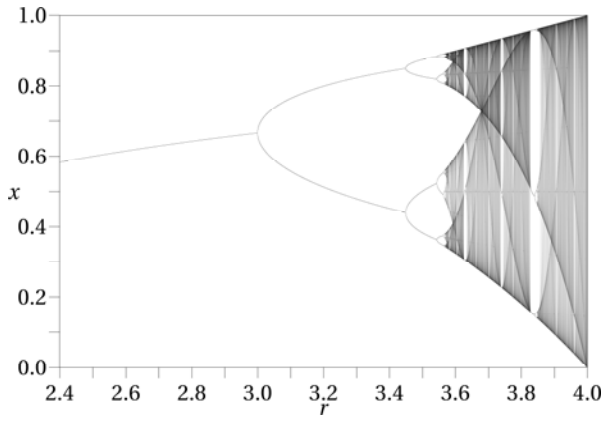
It is recognized that **mathematical models** [Bryant, Sangwin, 2008; Gustafsson, 2011, Chapter 2; Bellagamba, 2012, Chapter 2; Banerjee, 2014; Giordano, Fox, Horton, 2014] play a central role in science and engineering.

EPIDEMIOLOGY

For example, the equation

$$f(x) = \mathrm{r}x(1 - x),\ 0 \leq x \leq 1,\ 0 < \mathrm{r} \leq 4,$$

commonly known as the **logistic map**, is a discrete-time conceptual model for a variety of phenomena, including **population growth** (an area in epidemiology) [Banerjee, 2014, Section 2.4.1].



(Source: [Kneusel, 2015, Section 3.10].)

ECOLOGY

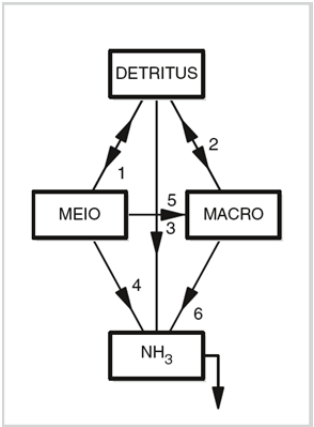
ECOLOGICAL MAP

A map is a model in which the relative position, names, and so on, of countries are emphasized, while details, such as the position of houses, agricultural information are ignored. It is a model, because it is simpler than the real world, and it is in a form that can be interpreted relatively easily and understood readily, even by non-ecologists [Soetaert, Herman, 2009].

(Source: [Soetaert, Herman, 2009].)

SEDIMENT ECOSYSTEM

In a conceptual model of a sediment ecosystem, the complexity of the food network has been collapsed into a few functional groups.



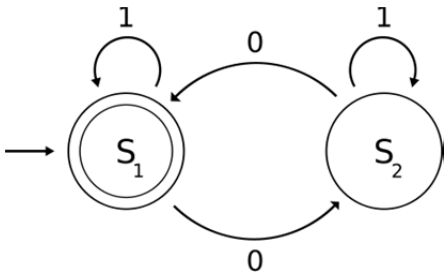
(Source: [Soetaert, Herman, 2009].)

5.2. MODELS IN COMPUTER SCIENCE

There are models of a number of things in computer science, such as sequential computers, parallel computers, and programming languages.

THEORY OF COMPUTATION

An **automaton** is a model of the behavior (specifically, states and transitions between states) of a computer program.



(Source: Google Images.)

PSEUDORANDOM NUMBERS

The equation

$$x_{n+1} = (a \times x_n + b) \pmod{c}$$

where c is the modulus, a is the multiplier, and b is the increment, is one of the earliest and most established method for **generating pseudorandom numbers** on a digital computer. In other words, the previous equation 'models' randomness in a digital computing environment.

Let $a = 1$, $b = 7$, and $c = 10$, and $x_n = 118$. Then, x_{n+1} is the remainder upon dividing $1 \times 118 + 7$ by 10, that is, $x_{n+1} = 5$.

In this method, the 'seeds' can be selected which generate patterns that eventually cycle. However, the length of the cycle is so large that the pattern does not repeat itself on large computers for most applications [Giordano, Fox, Horton, 2014, Section 5.2].

5.3. MODELS IN 'CONVENTIONAL' ENGINEERING (INDUSTRIAL ENGINEERING)

The use of models has a long and interesting history in industrial engineering [Embley, Thalheim, 2011; Henderson, Johnson, 2012].

BAKING BREAD

Figure 1 shows a conceptual model for a process for **baking bread**.

Figure 1. A model of the process for baking bread with a bread-making machine. (Source: [Snoeck, 2014].)

COMPUTER-AIDED DESIGN

In building engineering, there is a notion of **blueprint**[4]. It has been pointed out [Mehmood, Cherfi, 2009] that conceptual models serve as blueprints for information systems.

The use of **Computer-Aided Design (CAD)** has, in many situations, replaced classical blueprints by digital displays. For example, there are **computer simulations of a moving car** in automobile engineering.

(Source: Google Images.)

[4] URL: http://www.the-blueprints.com/ .

5.4. MODELS IN SOFTWARE ENGINEERING

REFERENCE MODEL

A reference model is a "model of something that embodies the basic goal or idea of something and can then be **looked at as a reference** for **various purposes**" [Wikipedia].

A **reference model in software engineering** consists of an interlinked **set of clearly defined concepts** produced by an expert or body of experts in order to encourage clear communication. This set could, for example, be represented by a **domain-specific ontology** [Becker, Delfmann, 2007].

For example, the **Capability Maturity Model for Software (SW-CMM)** is a reference model for performing **software process appraisals**.

For example, the **GREENSOFT Model** is a reference model for **"green and sustainable software"**.

6. META-MODEL

A meta-model is a model of a model.

For example, there are meta-models, such as the **Meta Object Facility (MOF)**, for defining modeling languages, such as the **Unified Modeling Language (UML)**.

7. MOTIVATION FOR MODELING

The unsuccessful software projects fail in their own unique ways, but all successful projects are alike in many ways. There are many elements that contribute to successful software [...]; one [of which] is the use of **modeling**.
— Grady Booch, Ivar Jacobson, and James Rumbaugh

There are a number of **reasons** for modeling:

- Communicate with People
- Document Decisions
- Furnish Abstractions
- Understand Domain
- Provide Structure
- To Explore and To Create
- Discover Errors

- Manage Risk
- Follow Trend

COMMUNICATE WITH PEOPLE

A model could be used to communicate with a stakeholder, such as with the client, a user, or a team member [Brambilla, Cabot, Wimmer, 2012, Section 1.2; Reinhartz-Berger, Sturm, Clark, Cohen, Bettin, 2013, Page ix].

This, in surveys, has been pointed out as the **most common reason** for conceptual modeling [Davies, Green, Rosemann, Indulska, Gallo, 2006, Table 10].

DOCUMENT DECISIONS

A model could be used to make the **knowledge of decisions, including agreements, explicit**.

In fact, almost every salient activity in software engineering requires decision making. A decision that remains **implicit** is lost or forgotten.

For example, consider a garden or patio or terrace chair (such as an Adirondack). The **juxtaposition of different parts** in assembling a chair is a decision that may need to be documented. (This is especially important in those cases where the shapes of the parts are rather similar, and therefore can be confused.)

(Source: Google Images.)

FURNISH ABSTRACTIONS

No problem can be solved from the same level of consciousness that created it.
— Albert Einstein

A model could be used to furnish abstractions so as to **manage complexity** [Selic, 2003; Fairbanks, 2010, Chapter 6; Sangwan, 2015, Section 1.2; Seidl, Scholz, Huemer, Kappel, 2015, Preface].

The software systems of today have become increasingly **complex** (structurally and/or behaviorally). (For example, consider the evolution of an operating system or a graphics editor.)

In general, interesting and useful software systems have a **natural tendency** to become more complex over time [Booch, Rumbaugh, Jacobson, 2005].

Figure 2 shows the **proportion of software in a fighter aircraft** over a time period of 60 years.

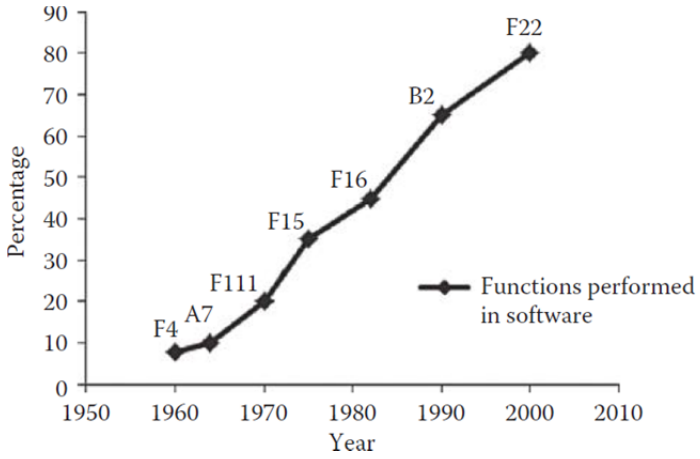


Figure 2. The proportion of avionics software in a US fighter aircraft. (Source: [Sangwan, 2015].)

However, there are inherent **limits** to the **human ability** to understand complexity.

The **commutative diagram** [Fairbanks, 2010, Figure 6.1] given in Figure 3 illustrates a relation between problems and abstraction: "**simple problems can be solved directly** (indicated by grey arrow), while **complex problems are solved by using abstractions**" (indicated by the long way around the diagram).

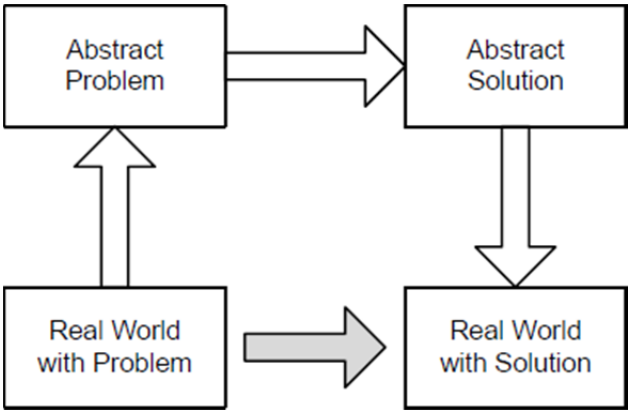


Figure 3. The role of abstraction in a methodology for problem solving.

UNDERSTAND DOMAIN

A model could be used to understand **the problem or the solution domain** better [Booch, Rumbaugh, Jacobson, 2005; Larman, 2005; Reinhartz-Berger, Sturm, Clark, Cohen, Bettin, 2013, Page ix].

Every argument made during a technical communication rests on an intimate understanding of the underlying domain. Therefore, developing an understanding of the domain is crucial.

Indeed, it has been stated that "the key to **successful requirements engineering** is being able to model: to **model the business, the system, and its users**" [Graham, 2008, Page 85].

PROVIDE STRUCTURE

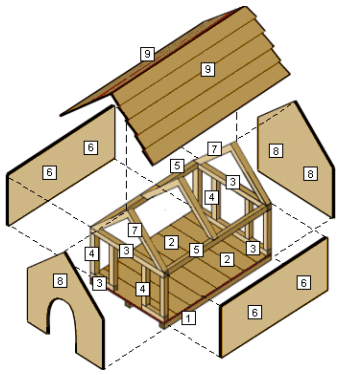
A model could be used to provide structure for **problem solving**.

For example, a process model usually includes steps that include activities to be carried out and artifacts to be produced. The collection of steps and activities, expressed in some form, lend structure to the process.

TO EXPLORE AND TO CREATE

Two things are infinite: the universe and human stupidity, and I am not sure about the universe.
— Albert Einstein

A model of a software system can also be used to create **other artifacts** related to a software system, such as **project plan, cost estimates, licensing conditions**, and so on.



(Source: Google Images.)

The following is an adaptation from [Booch, Rumbaugh, Jacobson, 2005]:

> If you want to build a dog house, you can pretty much start with a pile of lumber, some nails, and a few basic tools such as a hammer, saw, and tape measure.
>
> If you want to build a high-rise office building, it would be **infinitely stupid** for you to start with a pile of lumber, some nails, and a few basic tools.
>
> If you really want to build the software equivalent of a house or a high-rise, the problem is more than just a matter of writing lots of software. In fact, the trick is […] in figuring out how to **write less software**.

In general, for a single problem, there can be multiple solutions. A model could be used to **explore multiple solutions**. In doing so, models open avenues for **experimentation, creative thinking, and innovation** at relatively little cost. For example, a **prototype** is a kind of model used for this purpose. For another example, use case models can be used for software project estimation.

DISCOVER ERRORS

A model could be used to find errors and/or omissions **before** implementing the software system (and, evidently, before the system is in operation). This advocates **'prevention is better than (or, at least as significant as) cure'**.

MANAGE RISK

A model could be used to manage the risk of mistakes [Booch, Rumbaugh, Jacobson, 2005], and thereby **reduce the likelihood** of **software project failures**.

A software project faces several **risks**[5], inherent or otherwise. To err is human, and people make mistakes.

FOLLOW TREND

In the past couple of decades or so, modeling has become increasingly relevant in software engineering, as well as in human-computer interaction [Embley, Thalheim, 2011; Henderson, Johnson, 2012].

Indeed, it has been pointed out that modeling is one of the trends of the software engineering for the 21st century [Brambilla, Fraternali, 2015, Page xiii]. It has become common to see development approaches prefixed with phrases like '**model-driven**' such as **Model-Driven Architecture (MDA)**, **Model-Driven Software Development (MDSD)**, **Model-Driven Engineering (MDE)**, and so on.

8. THE MODEL ECOSYSTEM

A model does not exist in isolation. Indeed, a model is part of an **ecosystem**. There are a number of concerns in the ecosystem of a model, a subset of which is given in Table 1.

Model	Thing Stakeholder Quality Modeling Language Tool Medium Modality Process Related Models/Documents . . .

Table 1. A collection of things in a model ecosystem.

[5] For the sake of this document, a **risk** is a potentially adverse, and usually uncertain, event, with a positive probability of occurrence that may impair the development process or the quality of the product.

The terms **‘simplified’** and **‘viewpoint’** are the essence of a model.

9. CLASSIFICATION OF MODELS

There are multiple dimensions of classifying models, including the following:

- Physical/Conceptual
- Descriptive/Prescriptive
- Generative/Reflective

9.1. PHYSICAL VERSUS CONCEPTUAL

A model could be **physical** or **conceptual**. This categorization is **dichotomous**.

PHYSICAL MODEL

A physical model is a model of thing that has a physical (or material) existence.

It is common in ‘conventional’ engineering disciplines, such as civil, electrical, and mechanical engineering, to have physical models.

For example, a **toy replica** of a ship is a physical model of a **ship**.

(Source: Google Images.)

CONCEPTUAL MODEL

A conceptual model is a model of thing, where the thing can have a physical existence, or have a virtual (abstract or mental) existence based on some reality.

In the development of virtual products, such as **software systems**, the interest is in **conceptual models**. For example, the **Waterfall Model** is a conceptual model of a (linear) **software process**, and a **persona** is a conceptual model of a user of a specific **interactive system**.

For example, a **map** of the **route of a subway system** is a conceptual model of the physical **route of a subway system**.

(Source: Google Images.)

9.2. DESCRIPTIVE VERSUS PRESCRIPTIVE

A model could be **descriptive** or **prescriptive**. This categorization is dichotomous.

DESCRIPTIVE MODEL

A model of a thing that **already exists** is a descriptive model. For example, a **(problem) domain model** or a **user model** (such as a **persona**) is a descriptive model.

A descriptive model makes **indicative statements**. For example, a statement of the type " … X is Y …" is an indicative statement.

PRESCRIPTIVE MODEL

A model of a thing that **does not (yet) exist** is a prescriptive model. For example, a **use case model** is a prescriptive model.

A prescriptive model makes **putative statements**. For example, a statement of the type " … X should do Y …" is a putative statement.

9.3. GENERATIVE VERSUS REFLECTIVE

A model could be **generative** or **reflective**. This categorization is **not** dichotomous.

GENERATIVE MODEL

A model could be used to generate the **executables** of a software system.

For example, this is the case with visual programming languages and, to a certain extent, with the **Unified Modeling Language (UML)** [Booch, Rumbaugh, Jacobson, 2005].

REFLECTIVE MODEL

A model may be a reflection (such as an **orthographic projection**) of a software system. For example, this is the case with **models of software architecture**.

Figure 4 gives several **projections** of **stairs**.

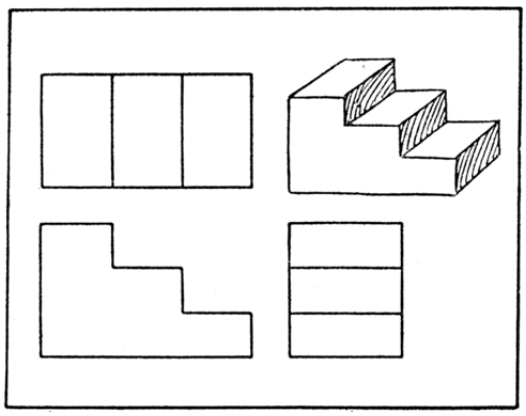


Figure 4. A collection of projections of stairs, some of which are orthographic. (Source: Google Images.)

10. GOAL OF A MODEL

A model must have a **goal** (or a purpose [Daniels, 2002; Lankhorst, 2013, Chapter 6]).

The goal of a model is inspired by **motivations for modeling** and **needs of the stakeholders** of the software project.

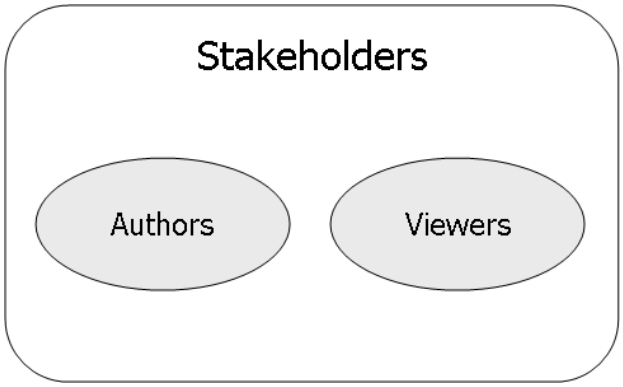
Table 2 illustrates certain goals in a typical software project and corresponding conceptual model.

Goal	Type of Model
Understand Value of Software System to an Organization	Business Model
Understand Problem Domain	Problem Domain Model
Understand User	User Model
Understand Interaction between a User and the Software System	Use Case Model

Table 2. A collection of goals and corresponding types of conceptual models.

11. STAKEHOLDERS OF A MODEL

There are a number of **stakeholders** of a model [Lankhorst, 2013, Chapter 6], including an **author** and a **viewer**. (For the sake of this document, both author and viewer are **human**.)



AUTHOR

A model must have an author. An author of a model is a **producer** of the model.

In a **collaborative modeling** environment, such as Pair Modeling, there are multiple authors of the same model.

The **goal** of a model is determined by the author(s).

VIEWER

A model must have a viewer. A viewer of a model is a **consumer** of the model.

For example, a **project manager** is a viewer of a process model or a quality model, and a **requirements engineer** is a viewer for a domain model or a use case model.

The **goal** of a model must be **aligned** with the needs of some viewer. The **usefulness** of a model is determined by a viewer. The **Return-On-Investment (ROI)** of modeling is determined, in part, by the type of viewer and, to a certain extent, by the number of viewers.

12. THE THING TO MODEL MAPPING

It is evident that, for a given thing, there is **no unique model**. Therefore, there are no 'right' or 'wrong' models; there are only **'purposeful' and 'purposeless'** models.

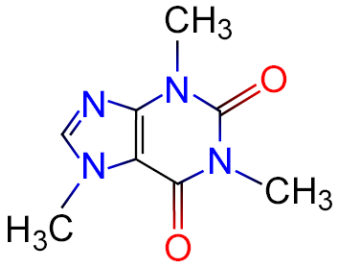
For example, consider a **coffee table** T. Then, a physical model M_1 for a **carpenter** is likely different from a conceptual model M_2 for a three-dimensional (3D) **graphics designer**.

(Source: Google Images.)

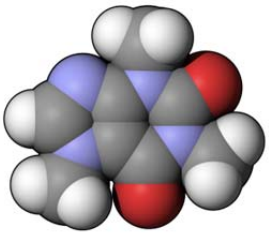
For example, the **caffeine molecule** can have multiple different models, in multiple different forms.

1,3,7-trimethyl-1H-purine-2,6(3H,7H)-dione

IUPAC Nomenclature



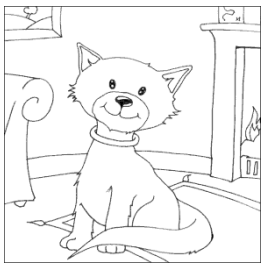
Chemical Formula



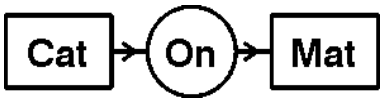
3D Rendering

For example, the **event** 'cat on a mat' can have three different models.

Photograph



Sketch



Conceptual Graph

(Source: Google Images.)

13. VIEWS IN SOFTWARE MODELING

There are several approaches for modeling a software system. The currently two most common ways for modeling software systems are from an **algorithmic perspective** and from an **object-oriented perspective**.

The classical view of software development takes an algorithmic perspective. In this approach, the main building block of all software systems is the **procedure or function**. It has been shown in experience that the resulting software system tends to be **brittle and unmaintainable**.

The contemporary view of software development takes an object-oriented perspective. In this approach, the main building block of all software systems is the **object or class**. It has been shown in experience that the resulting software system tends to be **robust and maintainable**.

14. CONCEPTUAL MODELS AS BOUNDARY OBJECTS

In sociology, a **boundary object** describes information used in different ways, by different communities.

A conceptual model is usually a boundary object. (In general, most software project artifacts are boundary objects.) It is therefore important that a conceptual model be represented and presented in a way so that it is able to satisfy different communities.

15. ON VIEWPOINTS AND VIEWS OF A MODEL

> Since the measuring device has to be constructed by the observer [...] we have to remember that what we observe is **not nature in itself**, but **nature exposed** to our method of questioning.
> — Werner Karl Heisenberg

A viewer, based on some goal, is interested in a certain view of a thing and 'looks' at that thing from a certain viewpoint, as shown in Figure 5.

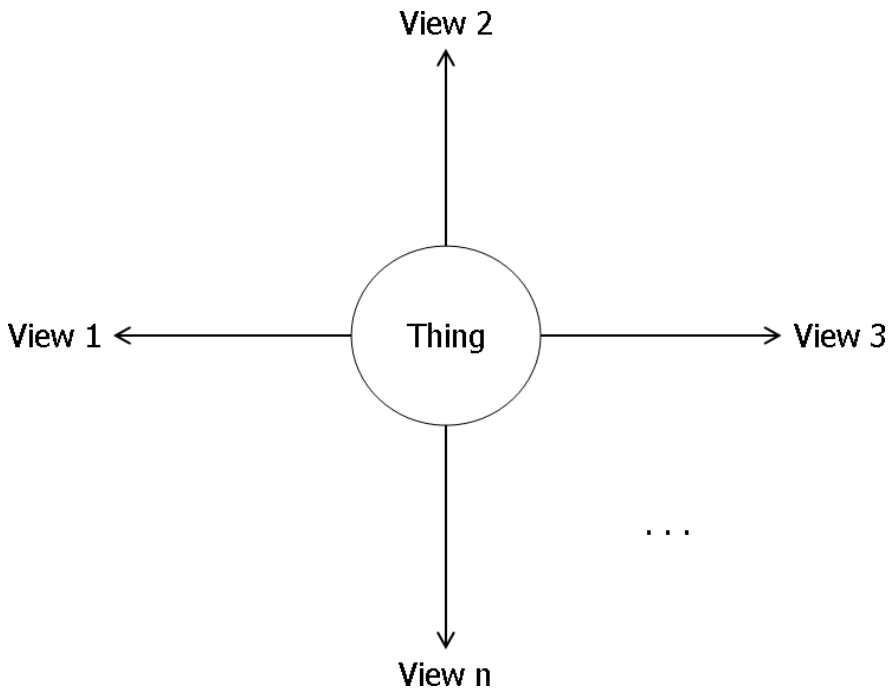


Figure 5. A thing can have multiple views, from different viewpoints.

A view corresponds to viewpoint.

It is with respect to the goal of a model that the viewpoint and view of the thing is chosen.

For example, a viewer could be the **Owner (Entrepreneur) or Market Specialist**, the **goal** could be **Reduce Time-to-Market**, and **viewpoint** is **Business**.

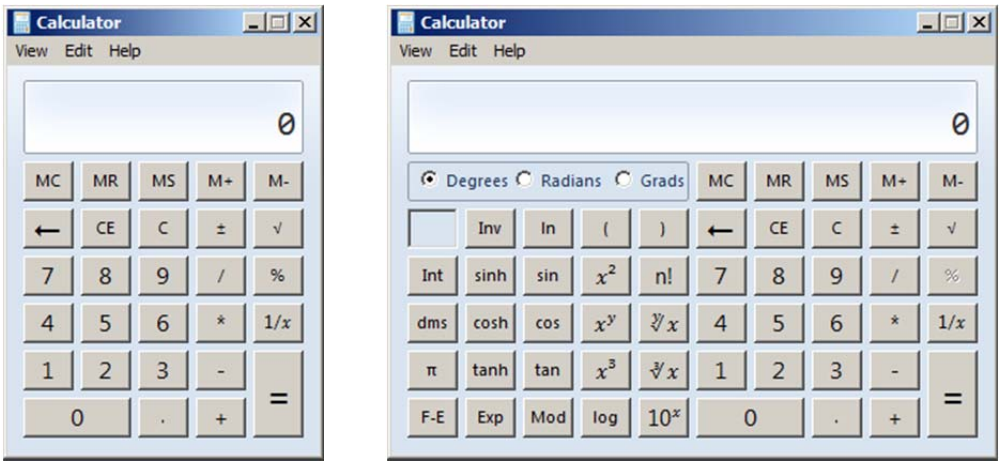
In general, the mapping between the **set of things** and the **set of models** is **many-to-many**. A thing can have several models, and, conversely, a model can correspond to several things. Indeed, each model may belong to a different level of abstraction [Daniels, 2002].

15.1. ONE THING, MANY VIEWS

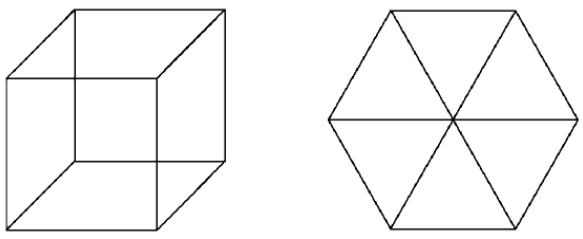
For the **same thing**, it is possible to have **multiple, different views**.

For a number of reasons, there is a need to have different views of the same thing. For example, different views may be suitable for different stakeholders.

For example, a calculator has **at least two different views** from two different viewpoints.

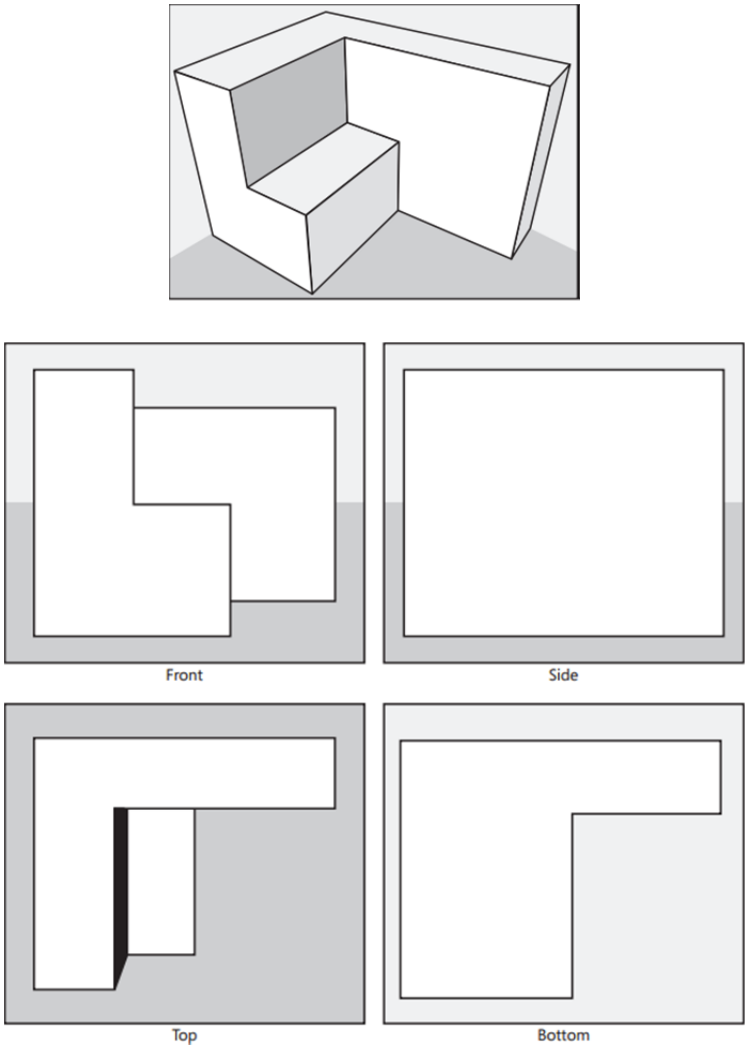


For example, a hollow cube has **at least two different views** from two different viewpoints [Gordon, 2004, Page 50].



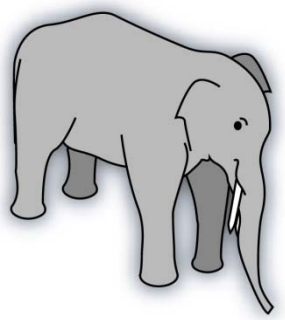
(Source: Google Images.)

For example, a three-dimensional structure shown below has **at least four different views** from four different viewpoints [Beatty, Chen, 2012, Page 16].



(Source: [Beatty, Chen, 2012].)

For example, an elephant can have **at least six different views** (depending on the viewpoints of six **blind** men): Fan, Rope, Snake, Spear, Tree, and Wall.



(Source: Google Images.)

REMARKS

For the same thing, there can be **multiple possible models**, each from a **different viewpoint**.

15.2. MANY THINGS, ONE VIEW

For two or more **different things**, it is possible to have the **same view**.

For example, two different software systems can have the same structure (say, software architecture).

There are essentially an **unlimited** number of views of an arbitrary given thing. It is **not** necessary that **all** views of a thing are useful. It is the goal of a model that determines the usefulness of a view.

REMARKS

- It is possible to have **same views from different viewpoints**. However, it is **not** possible to have different views from the **same** viewpoint.

- It follows from the previous discussion that, in general, the mapping between the **set of things** and the **set of views** is **many-to-many**.

16. ABSTRACTION IN CONCEPTUAL MODELING

The purpose of **abstraction** is highlighting the necessary aspects and suppressing the details. This suppression of information can be carried out using the software engineering **principle of abstraction** [Ghezzi, Jazayeri, Mandrioli, 2003].

The means to achieving the **'simplified'** in modeling is **abstraction** [Daniels, 2002; Selic, 2003; Lieberman, 2007; Boca, Bowen, Siddiqi, 2010, Chapter 1; Fairbanks, 2010, Chapter 6; Lankhorst, 2013, Chapter 6; Seidl, Scholz, Huemer, Kappel, 2015, Preface].

For example, consider the **brain** of a human being. Then the aspects necessary for a **psychologist** are different than those for a **neurosurgeon**. In case of a psychologist, the behavioral aspects of the brain such as **cognition and memory** are highlighted, and the structural details of the brain are suppressed.

(Source: Google Images.)

For example, as shown in Figure 6, a UML Class Diagram can be expressed at **three different levels** of abstraction, where each level has a different purpose:

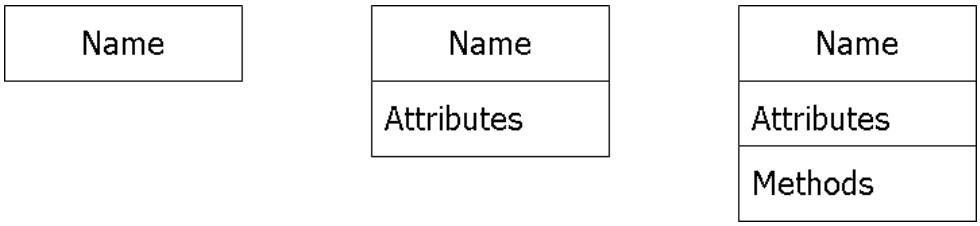


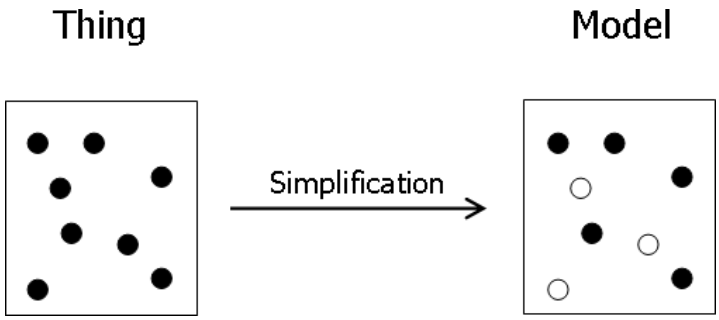
Figure 6. A UML Class Diagram at different levels of abstraction.

17. ON 'SIMPLIFICATIONS' OF A MODEL

If a picture is worth a thousand words then a model is often worth 1024 lines of code when applied in the right circumstances.
— Karl Wiegers

It is with respect to the goal of a model that the simplification of the thing takes place.

Let T be a thing, and let M be a model for T. A simplification of T is attained by removing (or ignoring) elements of T. In other words, there must be certain elements that are in T but not in M.



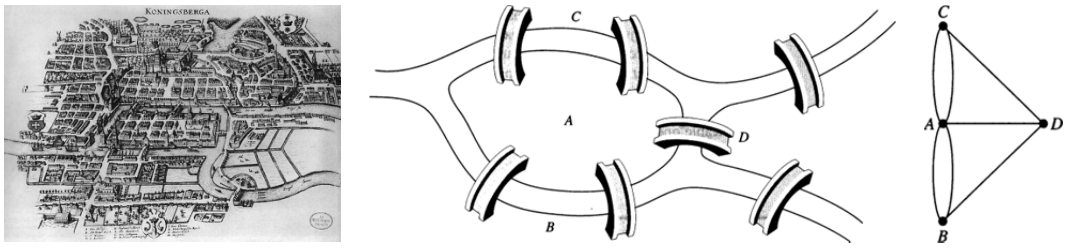
REMARKS

- The notion of simplification is aligned with the one of the practices of **Agile Modeling** [Ambler, 2002], namely to **Depict Models Simply**, and one of the UML 2.0 style guideline, namely **Show Only What You Have to Show** [Ambler, 2005, Guideline 14].

- The desired **simplification** is achieved through **abstraction** [Fernandes, Machado, 2016, Section 8.3].

EXAMPLE

This is about the **Königsberg Bridge Problem**, shown in Figure 7. It has applications in computer networks.



(a) (b) (c)

Figure 7. (a) A part of the map of the Town of Königsberg with the Seven Bridges, (a) A topographical model of the Seven Bridges of Königsberg, and (b) A multigraph model of the Seven Bridges of the Königsberg. (Source: [Rosen, 2012].)

The city of **Königsberg** in Prussia (now Kaliningrad, Russia) was set on both sides of the Pregel River, and included two large islands which were connected to each other and the mainland by seven bridges. **The problem was to find a walk through the city that would cross each bridge once and only once.**

The islands could not be reached by any route other than the bridges, and every bridge must have been crossed completely every time. (For example, one could not walk halfway onto the bridge, and then turn around and later cross the other half from the other side. In other words, the walk need not start and end at the same spot.) It was proven by **Leonhard Euler in 1735** that **the problem has no solution**. (In terms of graph theory, it means that that the graph does not have an **Euler Path**.)

The **choice of route inside each land mass is irrelevant**. The only important feature of a route is the sequence of bridges crossed. This allows one to **reformulate the problem** in **abstract** terms (specifically, in terms of **graph theory**), **eliminating all features except the list of land masses and the bridges connecting them**. In terms of graph theory, one replaces **each land mass with an abstract "vertex"**, and **each bridge with an abstract connection, an "edge"**, which only serves to record which pair of vertices (land masses) is connected by that bridge. The resulting mathematical structure is called a graph.

It is important to note that only the connection information is relevant, and so the shape of pictorial representations of a graph may be distorted in any way, without changing the graph itself. Furthermore, only the existence (or absence) of an edge between each pair of nodes is significant. For example, it does not matter whether the edges drawn are straight or curved, or whether one node is to the left or right of another.

17.1. AN ANALOGY OF CONCEPTUAL MODEL TO CARICATURE

A caricature is a description of a person using **exaggeration of some characteristics** and **oversimplification of others** [Wikipedia].

18. MODALITY OF A MODEL

> **Yin and yang** can be thought of as **complementary** (instead of opposing) forces interacting to form a **holistic dynamic system** (that is, the whole is greater than the sum of parts). In Taoist metaphysics, **good-bad distinctions** and other **dichotomous moral judgments** are **perceptual**, not real; so, yin-yang is an indivisible whole.
>
> Everything has both yin and yang aspects. (For example, shadow cannot exist without light.)
>
> Either of the two major aspects may manifest more strongly in a particular object, depending on the criterion of the observation. (Source: Wikipedia.)

All our knowledge begins with the **senses**, proceeds then to the **understanding**, and ends with **reason**.
— Immanuel Kant

The modality of information refers to the way an idea is expressed or perceived. There are a number of possible modalities, including **aural and visual** (specifically, textual, symbolic, or graphical), or tactile, each with their own advantages and limitations.

The description of a conceptual model is in some modality. This modality could, for example, be visual.

In general, modalities are **complementary** rather than competing [Moody, 2009, Section 4.7]. In particular, the textual and graphical representations are **complementary** rather than competing.

TEXT VERSUS GRAPHICS

It can be noted that a graphical representation provides **one view** of information[6]. In doing so, it will, by necessity, **suppress** certain information. The information suppressed in a graphical representation can be highlighted in a textual representation.

[6] For example, the caption associated with a figure is expected to make the view of information to be communicated by the author to the viewer explicit.

For example, text is limitlessly expressive, and graphic is relatively more compact. Indeed, to provide a complete model, a **heterogeneous combination** of modalities may be necessary. For example, a graphic may need **legend and caption**, both of which are expressed in text.

REMARKS

There is **guidance on diagramming** available [Tufte, 1990; Ambler, 2005; Booch, 2011].

19. REPRESENTATION AND PRESENTATION OF A MODEL

The representation of a model is relevant to **machines**. The presentation of a model is relevant to **humans**.

The **separation** between **representation and presentation** of a model is important for a variety of reasons, including **productivity, reusability, and maintainability**.

The Extensible Markup Language (XML) can be used to represent a model. For example, **XML Metadata Interchange (XMI)**, a markup language based on XML, is the serialization format for UML.

20. MEDIUM OF A MODEL

A model resides in some medium. The medium of a model could be **physical** or **virtual**.

Let T be a thing, and M be a model for T. Then, T and M could be in a physical medium or in a virtual medium. This leads to 2×2 possibilities.

In other words, both T and M could be in a physical medium, both T and M could be in a virtual medium, T could be in a physical medium and M could be in a virtual medium, or T could be in a virtual medium and M could be in a physical medium.

For example, let T be a software system. Then, M could be on a **paper**, on a **black or whiteboard**, or on a **tablet**.

(Source: Google Images.)

21. DEFINITION OF CONCEPTUAL MODELING

An effective practice of modelling must seek to create a bridge that is strong at both ends: it must find **abstractions of reality** that are adequate for the purpose in hand and its **context**; and it must express and **convey** those abstractions in ways that serve the **goal** of human understanding as well as possible.
— Michael A. Jackson

There are a number of possible definitions of conceptual modeling, including the following:

Definition [Conceptual Modeling] [Borgida, Chaudhri, Giorgini, Yu, 2009]. The activity of formally describing some aspects of the physical and social world around us for purposes of understanding and communication.

REMARKS

- The foregoing definition can be improved in a number of ways.
- A successful approach to modeling requires a combination of art, science, and engineering.
- It could be noted that modeling is more than drawing [Seidl, Scholz, Huemer, Kappel, 2015, Preface].

22. THE PROCESS OF CONCEPTUAL MODELING

There is **no generic process** of conceptual modeling that applies to all conceptual models [Lankhorst, 2013, Chapter 6]. The process of conceptual modeling, including the activities therein, will depend on the type of conceptual model.

There are processes for specific kinds of models. For example, a **pattern-oriented use case modeling process** is given in [Kamthan, 2009; Kamthan, Shahmir, 2009].

22.1. PROPERTIES OF A PROCESS OF CONCEPTUAL MODELING

The process of conceptual modeling is carried out **iteratively and incrementally**.

The process of conceptual modeling could be carried out **individually** or **collaboratively**.

The process of conceptual modeling must be **feasible**, that is, the benefits of creating a model should outweigh the costs. In particular, it must be cheaper to create a model than to create the thing being modeled [Selic, 2003].

REMARKS

There are a number of possible **approaches for conceptual modeling**, each with different focus. A comparison of some approaches is available[7].

23. AGILE MODELING

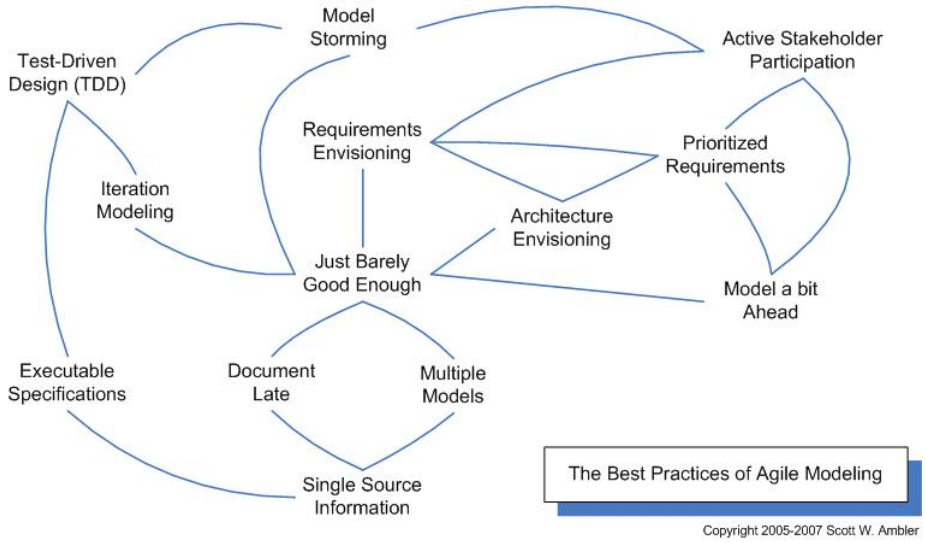
Agile Modeling[8] [Ambler, 2002] is a practice-based methodology for effective modeling of software systems. It consists of a collection of **values**, **principles**, and **practices** for modeling software. The target of Agile Modeling is **agile software projects**.

Agile Modeling is meant to be **tailored and integrated** into other, full-fledged methodology, such as **Extreme Programming (XP)** [Beck, Andres, 2005] or the Rational Unified Process (RUP) [Kruchten, 2004].

An agile model is a "**conceptual model that fulfills its purpose and no more**; is understandable to its intended audience; is simple; sufficiently accurate, consistent, and detailed; and investment in its creation and maintenance provides positive value to your project". The focus of Agile Modeling is the **economy of a model**.

Thus, Agile Modeling is aligned with **"Perfect" is the Enemy of "Good Enough"**, one of the **97 Things Every Software Architect Should Know** [Monson-Haefel, 2009].

[7] URL: http://www.agilemodeling.com/essays/modelingApproaches.htm .
[8] URL: http://www.agilemodeling.com/ .



(Source: Scott Ambler.)

23.1. VALUES OF AGILE MODELING

The values of Agile Modeling, inspired by the values of Extreme Programming (XP), are: **Communication, Simplicity, Feedback, Courage, and Humility**.

23.2. PRINCIPLES OF AGILE MODELING

Core Principles	Supplementary Principles
• Assume Simplicity • Embrace Change • Enabling the Next Effort is Your Secondary Goal • Incremental Change [+] • Maximize Stakeholder ROI [+] • Model With a Purpose [+] • Multiple Models [+] • Quality Work • Rapid Feedback • Working Software Is Your Primary Goal • Travel Light	• Content is More Important Than Representation [+] • Open and Honest Communication

A [+] indicates **positive emphasis** and a [−] indicates **cautionary emphasis**.

23.3. PRACTICES OF AGILE MODELING

Core Practices	Supplementary Practices
• Active Stakeholder Participation • Apply the Right Artifact(s) • Collective Ownership [+] • Create Several Models in Parallel • Create Simple Content • Depict Models Simply • Display Models Publicly [−] • Iterate to Another Artifact • Model in Small Increments [+] • Model With Others [+] • Prove it With Code • Single Source Information • Use the Simplest Tools [+]	• Apply Modeling Standards • Apply Patterns Gently • Discard Temporary Models [−] • Formalize Contract Models • Update Only When It Hurts

24. LITERATE MODELING

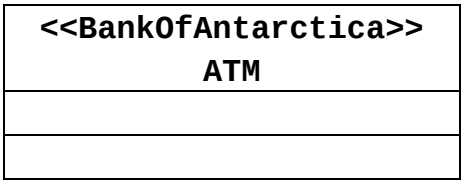
The goal to make computer programs more communicable led to a **style of programming** known as **Literate Programming**[9].

The goal to make conceptual models for software systems **more communicable** has led to the introduction of a **style of modeling** called as Literate Modeling [Arlow, Neustadt, 2004, Chapter 3]. The **communicability of a model** is the focus of Literate Modeling. Literate Modeling has been embraced in the **guidelines for conceptual modeling** [Lankhorst, 2013, Chapter 6].

In Literate Modeling, **documentation is integral** to a conceptual model. This is possible in UML by an **appropriate use of stereotypes and tagged values**, standard or otherwise, in preference to using the UML Note construct.

[9] In **Literate Programming**, a computer program is written in natural language as a piece of literature interspersed with snippets of macros, thereby reflecting the logic and thought processes of the programmer rather than of the computer. In Literate Programming, internal documentation is **integral** to a computer program. This is a departure from conventional programming style where internal documentation (comments) is sprinkled at appropriate places in a program.

For example, the following attempts to make the type of **Automated Teller Machine (ATM)** clearer by including a **non-standard stereotype**:



REMARKS

It is evident that non-standard stereotypes may not be understood by **all**, and therefore must be defined and made available to the stakeholders of the model. Indeed, the appropriate place for doing that is the **project glossary**.

25. PAIR MODELING

In general, the conceptual modeling process is **independent** of the number of people involved. However, there are cases where there is a need for **collaboration** and **agreement** during the construction of conceptual models [Brambilla, Cabot, Wimmer, 2012, Chapter 10]. For example, this is the case in domain modeling.

In Pair Modeling [Kamthan, 2005; Kamthan, 2008], two people collaborate in the construction of a conceptual model.

The **relationship and communication between authors of a model** is the focus of Pair Modeling.

Let P1 and P2 be two people involved in the conceptual modeling process. Then, the roles of P1 and P2 are **different**. P1 is primary (the **pilot**) and P2 is secondary (the **navigator**). For example, P1 sketches and P2 provides input (in form of questions, guidelines, and so on) or informally inspects the conceptual model. This is illustrated in Figure 8.

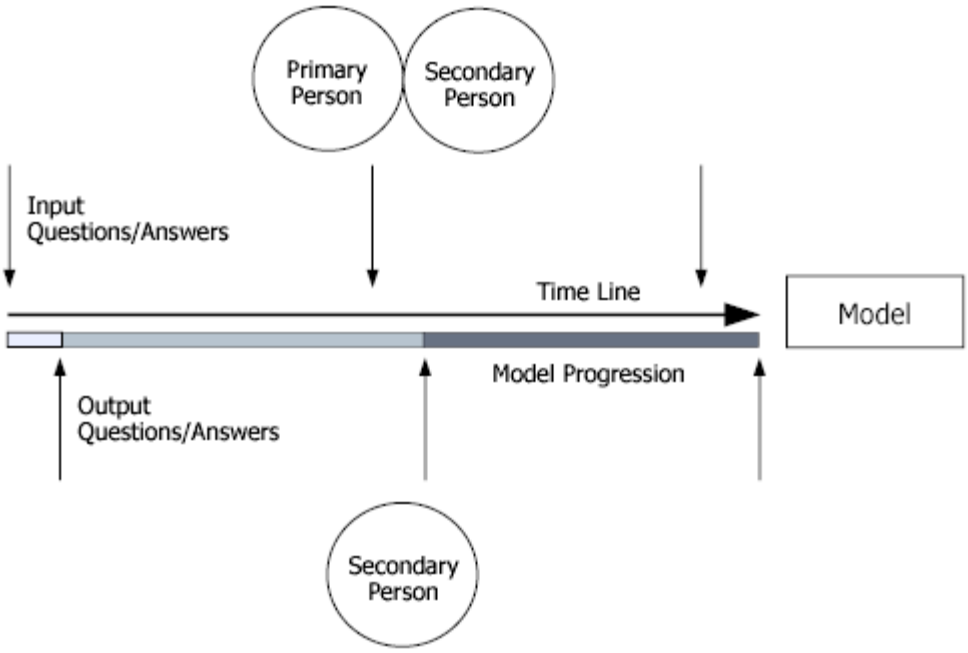


Figure 8. The activities over time in Pair Modeling.

REMARKS

Pair Modeling and Agile Modeling are **related** in some aspects. For example, Pair Modeling assumes the Agile Modeling practice named **Model With Others**.

26. SITUATING CONCEPTUAL MODELING IN A SOFTWARE PROCESS

In this document, it is assumed that conceptual modeling is being conducted as **part** of software development. In such a case, the process of conceptual modeling is a **sub-process** of the software development process.

There are certain software processes that **explicitly** support conceptual modeling. These include the **Unified Process (UP)** [Jacobson, Booch, Rumbaugh, 1999], in general, and its customizations, namely the **Rational Unified Process (RUP)** [Kruchten, 2004] (commercial) and **OpenUP** [Kroll, MacIsaac, 2006] (open source), in particular, and the **ICONIX Process** [Rosenberg, Stephens, Collins-Cope, 2005].

It is known [Rolland, Prakash, 2000] that conceptual modeling and requirements engineering are related intimately.

27. MODELING MATURITY

It is **not** automatic that all organizations have the same level of commitment to modeling.

The level of maturity of an organization can be defined in terms of their modeling practices [Warmer, Kleppe, 2003]. The different levels of modeling maturity are given in Table 3.

Maturity Level	Activity/Artifact
Level 0	No Specification
Level 1	Textual
Level 2	Text with Diagrams
Level 3	Models with Text (Heterogeneous Combination of Natural Language Text and Graphics.)
Level 4	Precise Models (Use of Object Constraint Language (OCL).)
Level 5	Models Only

Table 3. A collection of modeling maturity levels.

REMARKS

- The idea of modeling maturity is motivated by the **Capability Maturity Model for Software (CMM or SW-CMM)** [Paulk, Weber, Curtis, Chrissis, 1995] and by the **Model-Driven Architecture (MDA)** [Kleppe, Warmer, Bast, 2003; Mellor, Scott, Uhl, Weise, 2004]. The purpose is to reduce the 'gap' between models and source code, and to move from manual to automatic generation of source code.

- The proposal is an individual/isolated effort, not a standard. There is a bent towards the use of UML and OCL.

28. PRINCIPLES OF MODELING

There are a number of principles of conceptual modeling that can make models **communicable as well as useful** [Booch, Rumbaugh, Jacobson, 2005; Lankhorst, 2013, Chapter 6].

Principle 1. The choice of what models to create has a profound influence on how a problem is [approached] and how a solution is shaped.

The selection of models profoundly impacts the world-view, and each world-view leads to a different kind of software system, with different costs and benefits. For example, the view of a relational database engineer or an object-oriented programmer is **different**, and so is the choice of the models.

Principle 2. Every model may be expressed at different levels of precision.

For example, in user interface design, there is a need for models at different levels of fidelity.

Principle 3. The best models are connected to reality.

For example, a mathematical model of an aircraft that assumes only ideal conditions and perfect manufacturing can **mask some potentially fatal characteristics** of the real aircraft [Booch, Rumbaugh, Jacobson, 2005].

Indeed, by definition, all models need to simplify reality; the trick is ensuring that the simplifications do not mask any important details.

Principle 4. No single model or view is sufficient.

Every nontrivial [software] system is best approached through a small set of nearly independent models with multiple viewpoints.

For example, there are multiple views of a given software architecture and, depending on the domain, one view may be more important than the others.

Principle 5. Keep abstraction levels consistent.

Principle 6. Keep related models consistent.

29. LANGUAGES FOR CONCEPTUAL MODELING

The choice of language **depends** on the problem.
— Brian W. Kernighan and Philip J. Plauger

The description of a model is expressed in some language. This language may be classified in different ways: it may be (1) **general-purpose** or **domain-specific**, or (2) **informal** (natural) or **formal** (mathematical). If formal, the language could be partially or completely formal.

There has been **no single language** that satisfies **all possible needs** of conceptual modeling.

In the past few decades, there have been a number of proposals of languages for modeling the structure and behavior of software systems [Jorgensen, 2010; Brambilla, Cabot, Wimmer, 2012, Chapter 6; Wazlawick, 2014]. This proliferation led to a variety of **issues** such as threat to **communicability** among humans and threat to **interoperability** among machines.

There are efforts to **unify** different languages for conceptual modeling.

The **Unified Modeling Language (UML)** [Rumbaugh, Jacobson, Booch, 2005] is a **generic, graphical, moderately expressive, semi-formal, standard** language with a medium-to-high learning curve. UML is aimed towards modeling **object-oriented systems**.

UML is a result of **convergence** of a number of different languages for conceptual modeling. UML has evolved over the past decade.

SELECTION

The selection of a modeling language can vary on a number of criteria including **applicability** (that is, suitability for purpose), **expressivity** (of information), **formality (mathematical 'rigor'[10])**, **learnability** (by stakeholders), **modality** (of notation), and **support** (say, through standardization, adoption, presence of tools, and so on).

[10] For example, a scheme commonly used to denote a formality spectrum is: informal (non-mathematical syntax and non-mathematical semantics), semi-formal (mathematical syntax and non-mathematical semantics), and formal (mathematical syntax and mathematical semantics).

UML is increasingly being accepted as the **de facto language** for modeling in software engineering [Spinellis, 2010].

NOTATION

A language has a **primary notation** and a **secondary notation**.

The **primary notation** of a language is responsible for a **representation**. The **secondary notation** [Petre, 1995] is responsible for a **presentation**.

GRAPHICAL MODELING LANGUAGES IN CONTEXT

There are both advantages and limitations of using graphical languages for conceptual modeling [Reed, 2010; Kop, Mayr, 2011].

ADVANTAGES

- **Sensory Modality.** It is known that vision is the primary sense of humans. Usually, many people perceive, explore, and understand the world around them through their eyes since they are children.

- **History.** It is known that pictures (such as, cave paintings), graphical symbols (such as, cave drawings), statues, and totem-poles were among the first models by humans.

- **Expressivity.** It is known that graphical representations can provide a spatial overview of an object, an event, or a problem.

- **Brevity.** It is known that picture can often provide a summary of text in a compact form. In particular, pictures are useful in expressing multiple, complex relationships. It is often said that a **"picture is worth a thousand words"**. For example, such a picture could be a **blueprint**.

LIMITATIONS

- **Notations.** The semantics of a picture may not be evident. A reader of a graphical model must have the knowledge of the notation used, and the skills to read and interpret the picture. For example, that one is able to see a cave painting does not mean the one can understand it (entirely).

- **Scale.** If there are many concepts being simultaneously depicted in a picture, then the picture loses its ability to provide an overview. For example, a UML Use Case Diagram may show 10 actors and 50 use cases, or a UML Class Diagram may have 100 classes with all the details of operations and properties of each class. In such a case, a reader of the picture may also not know which concepts to focus upon. (This problem can somewhat be circumvented by those conceptual modelers that can present the picture are **different levels of abstraction** (that is, by showing certain kinds information while hiding other kinds of information).

30. TOOLS FOR CONCEPTUAL MODELING

In general, modeling can be carried out manually or with the assistance of a tool[11], or both. For the sake of this document, a tool specifically used for modeling is called a **modeler**[12].

There are both **advantages and limitations** of deploying a modeler for the purpose of modeling.

ADVANTAGES

The advantages of deploying a modeler for the purpose of modeling include opportunities for distributed **collaborative authoring** of the model; **automation** (syntactic checking; to a certain extent semantic checking, such as, consistency checking); **manipulation** (copy-and-paste; search-and-replace; and so on); potential for **reuse of modeling language constructs**; **dissemination** of the model; **generation of source code**; and so on.

LIMITATIONS

The limitations of deploying a modeler for the purpose of modeling include the presence of **learning curve of the modeler** and the potential for becoming **habitual of the modeler** (and, therefore, having a **dependency on a specific modeler**).

It is better to consider a modeler as an **assistant** to, rather than a replacement of, a human. For example, a modeler does not have a capability to check for all desirable quality attributes, and therefore human intervention is necessary.

---

[11] URL: http://agilemodeling.com/essays/simpleTools.htm .

[12] In certain cases, especially when the model has a graphic modality, an illustrator can be used as a tool. However, an illustrator is **not** a modeler. In general, a modeling tool and a drawing tool are **different** [Brambilla, Fraternali, 2015, Section 2.3.1].

SELECTION

The **selection** of a modeler is an **important** consideration. This is because, at the very least, there are **implications**, including the potential for **inadvertent side-effects**, arising from a **commitment** to any modeler.

In general, a modeling tool can vary on a number of **dimensions** including cost ($s or not), openness (availability of source code or not), degree of support of a modeling language (100% support or not), usability, and so on.

31. MANAGEMENT OF CONCEPTUAL MODELS

The concerns of managing conceptual models include archiving/retrieving conceptual models, associating version information with conceptual models, and assuring quality of conceptual models [Brambilla, Cabot, Wimmer, 2012, Chapter 10].

32. QUALITY OF CONCEPTUAL MODELS

Essentially, all models are wrong, but some are useful.
— George E. P. Box

A good model guides your thinking, a bad one warps it.
— Brian Marick

There is no 'inherently' good or bad (conceptual) model [Lankhorst, 2013, Chapter 6]. However, some (conceptual) models are more useful than others.

There are different possible views of quality [Wong, 2006]. From the ISO/IEC 9126-1:2001 Standard, the following definition of the quality of a domain model could be derived.

Definition [Quality of Conceptual Model]. The totality of characteristics of a conceptual model that bear on its ability to satisfy **stated and implied** needs.

REMARKS

The issue of the quality of conceptual models has been of interest in recent years [Lindland, Sindre, Sølvberg, 1994; Selic, 2003; Kamthan, 2004; Bolloju, Leung, 2006; Mohagheghi, Dehlen, Neple, 2009; Moody, 2005; Moody, 2009; Nelson, Monarchi, 2007; Nelson, Poels, Genero, Piattini, 2012; Lankhorst, 2013, Chapter 6; Snoeck, 2014; Seidl, Scholz, Huemer, Kappel, 2015, Section 1.2].

33.1. MOTIVATION

The introduction and use of conceptual models in software engineering has its indirections, including **dependency**.

In particular, a conceptual model is depended upon by both **humans** and, in some cases, **other software artifacts**. If a conceptual model does not meet the desirable expectations of quality, then the potential of modeling can not be completely realized.

For example, if there are errors in the problem domain model, then there will be errors or 'gaps' in the understanding of the problem domain by the team members.

For example, if there are errors in one design model, then those errors can seep into other design models and/or implementation models.

33.2. UNDERSTANDING THE NOTION OF QUALITY IN CONCEPTUAL MODELS

It is known that quality is a multi-dimensional concept.

There are a number of dimensions of the quality of a conceptual model [IEEE, 2014, Chapter 9; Seidl, Scholz, Huemer, Kappel, 2015, Preface]: **lexical** (or empirical) quality, **syntactic** quality, **semantic** quality, **pragmatic** quality, and **social** quality, as illustrated, in part, in Figure 9.

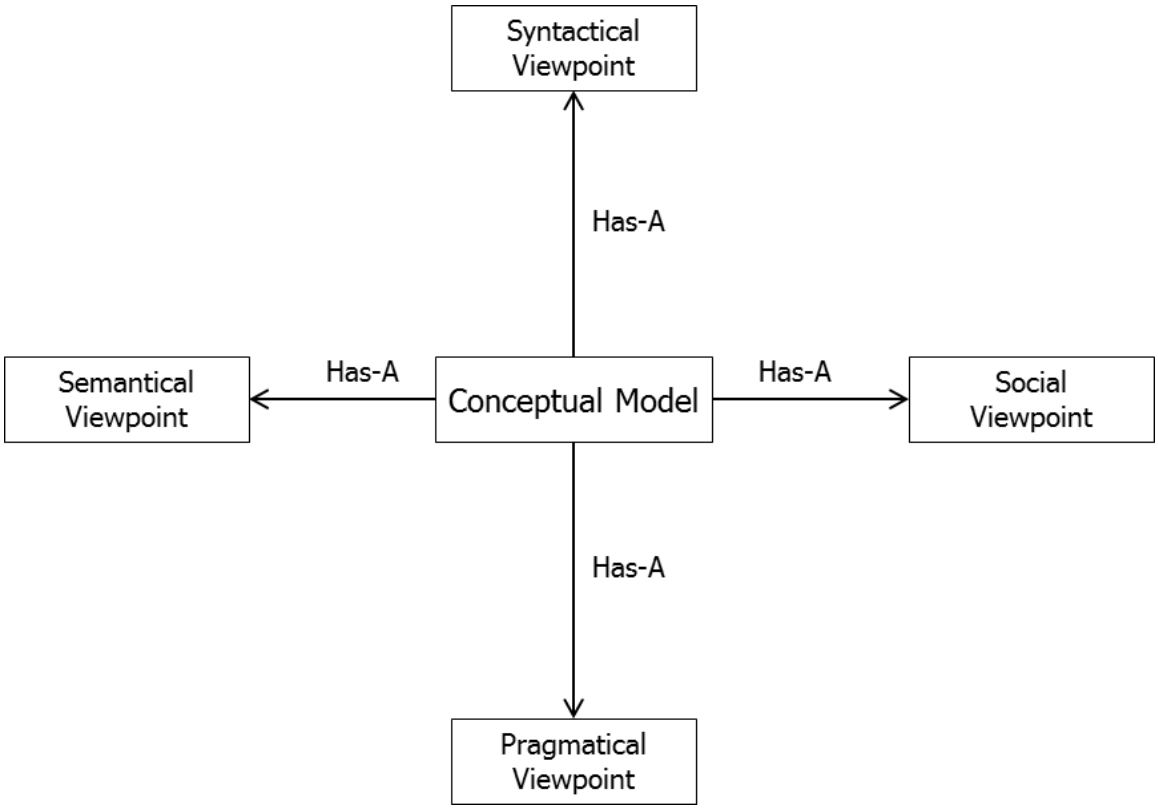


Figure 9. A collection of viewpoints from which the issue of quality of a conceptual model can be assessed.

For example, expressiveness is a lexical quality, correctness is a syntactic quality, consistency is a semantic quality, accessibility is a pragmatic quality, and credibility is a social quality.

EXAMPLE

There are two **semantic errors** in the UML Class Diagram shown in Figure 10 [Nelson, Poels, Genero, Piattini, 2012]: (1) a book is always on loan, and (2) a book can be simultaneously loaned to more than one library member.

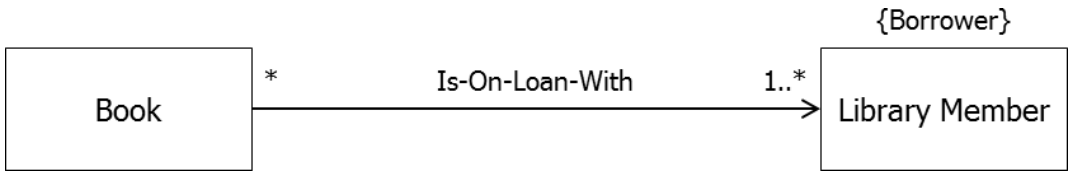


Figure 10. A UML Class Diagram with semantic quality issues.

34. ORGANIZING ELEMENTS OF A MODEL

In case the modality of a model is visual, the issue of **visual organization** arises. The visual organization of elements of a model is relevant to the **pragmatic quality** of that model.

It is important that such organization aim to **reduce visual complexity**. This can be achieved by [Lankhorst, 2013, Chapter 6]: (1) limiting the number of types of concepts and relations, and (2) limiting the number of concepts and relations that are visible, in a model.

35. PERCEPTION IN CONCEPTUAL MODELING

The presentation of a model depends intrinsically on **human perception**. The human perception can be affected by **context**.

For example, in the following table, 'I' and '1' can be visually interchanged.

I	2	3
J		
K		

It is therefore important to be careful in the **selection and use of characters**, **individually** and **collectively**.

For example, if oriented differently (say, by rotating 180 degrees), a depression in the ground may appear as elevation.

(Source: [Stone, 2013, Page 26].)

The studies in **Gestalt**[13] **Psychology** suggest **Laws of Organization**[14] of visual elements [Lidwell, Holden, Butler, 2003; Metzger, 2006; Johnson, 2010; Mazza, 2009; Lidwell, Holden, Butler, 2010; Leeuwenberg, Helm, 2013; Johnson, 2014].

These 'laws' are considered especially important in **user-centered design** [Lowdermilk, 2013, Chapter 7] and **information visualization** [Mazza, 2009, Section 3.5; Ware, 2013, Chapter 6].

In Gestalt Psychology, there are **six** main **factors** that determine how humans group things according to visual perception:

1. Proximity
2. Similarity
3. Closure
4. Symmetry
5. Uniform Destiny (Common Fate)
6. Continuity

GESTALT LAW OF PROXIMITY

Statement: The elements that are spatially or temporally close are perceived as a collective.

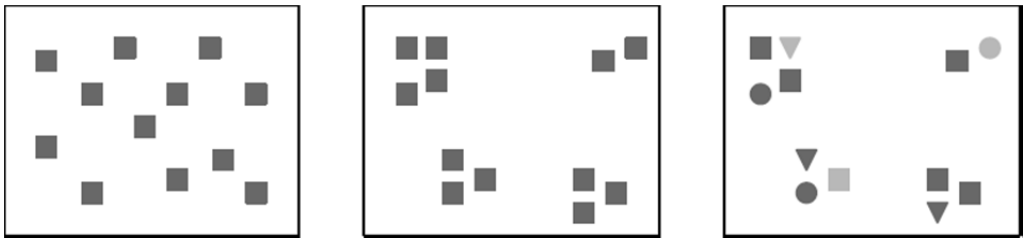
The law of proximity also applies to colors. Therefore, the color of objects in a model can be applied to indicate relations between objects.

EXAMPLE

EXAMPLE

In the image on the left, the squares are placed without proximity, and therefore are perceived as 12 separate elements. In the central image, the squares form four distinguishable groups. Even if the shapes or colors of the objects are different, they will appear as a group, as can be seen in the image on the right.

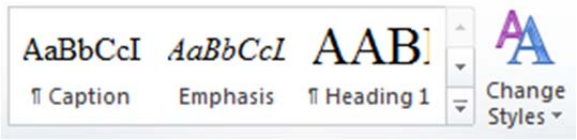
[13] Gestalt is a German word that translates in English to configuration or shape.
[14] These Laws of Organization are inspired by the Law of Prägnanz. Prägnanz is a German word that translates in English to pithiness or succinctness.



(Source: [Mazza, 2009].)

EXAMPLE

This law has been applied in Microsoft Office 2010:



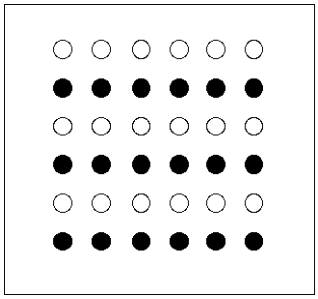
UML

The classifiers in UML that are relatively closer to each other may be perceived as being related differently than those that are relatively further.

GESTALT LAW OF SIMILARITY

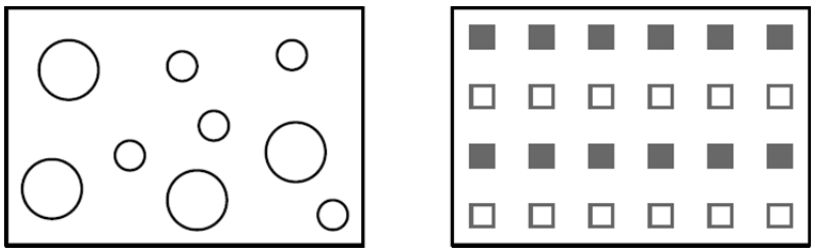
Statement: The elements that are similar in some way (say, color, opacity, orientation, shape, or size) are perceived as a collective.

EXAMPLE



EXAMPLE

In the image on the left, the objects of two distinct sizes seem to belong to the same group. In the image on the right, the filled and empty squares are associated naturally, and show alternating rows of filled and empty squares.



(Source: [Mazza, 2009].)

UML

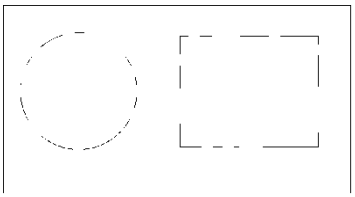
The **classifiers** in UML that have the same size or color may be perceived as being related differently than those that are not. Let there be two classifiers in UML of the same type, but of different sizes. Then one classifier may be seen as relatively more significant than the other, based solely on their sizes.

GESTALT LAW OF CLOSURE

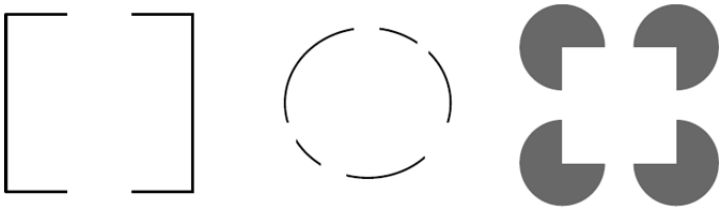
Statement: The elements that have an 'incomplete' shape may be perceived as regular. For example, an unrelated collection of curves placed in a certain configuration may be perceived as a circle.

EXAMPLE

EXAMPLE

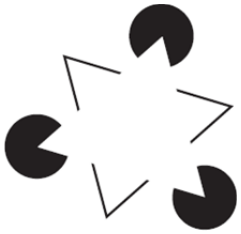


EXAMPLE



(Source: [Mazza, 2009].)

EXAMPLE



(Source: [Johnson, 2014].)

EXAMPLE

This law can be used to identify product insignia:

(Source: Google Images.)

EXAMPLE

This law can be used to identify organization logos:

(Source: Google Images.)

EXAMPLE

(Source: Google Images.)

UML

A dependency relationship in UML is different from an association relationship, and the two must be distinguishable in a UML diagram type.

GESTALT LAW OF SYMMETRY

Statement: The elements that are symmetric are perceived as a collective, even if they are far apart.

A **symmetry of an object in the plane** is a rigid motion of the plane that leaves the object apparently unchanged [Tapp, 2012].

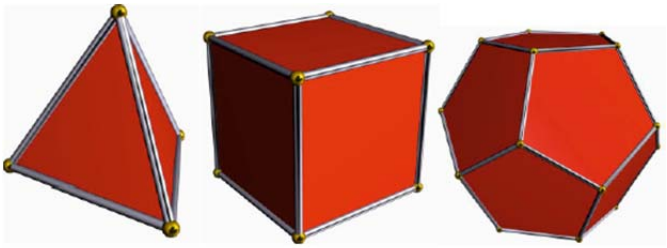
There are **different kinds of symmetries** pertaining to conceptual modeling (including **reflection, rotational, translational, and scale** symmetry).

EXAMPLE

The following object has **reflection (or mirror) symmetry**:

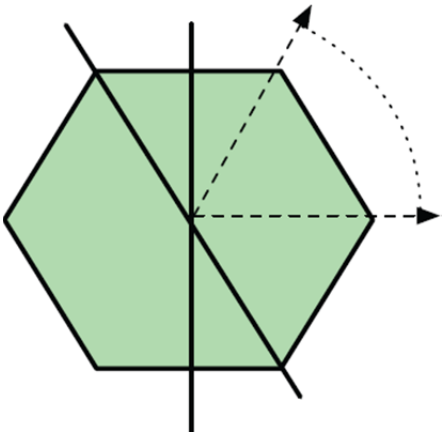
EXAMPLE

The following objects have **scale symmetry**:



EXAMPLE

The following object has **two kinds of reflection symmetry**, as well as a **rotational symmetry** [Chu-Carroll, 2013, Page 171]:



(Source: [Chu-Carroll, 2013, Page 171].)

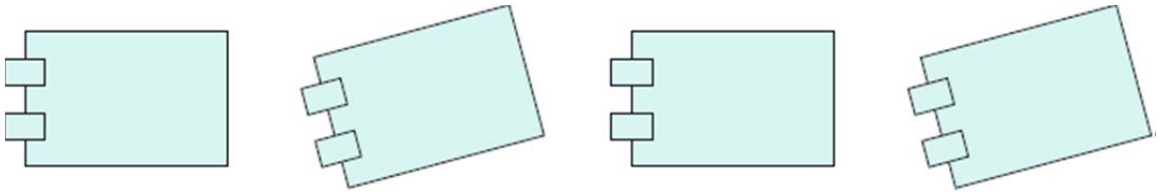
UML

An appropriate use of symmetry can help **readability and evocativeness** of a model.

GESTALT LAW OF UNIFORM DESTINY (COMMON FATE)

Statement: The elements that are moving in the same direction, or function in a similar manner, are perceived as a collective.

EXAMPLE



(Source: [Lankhorst, 2013, Chapter 6].)

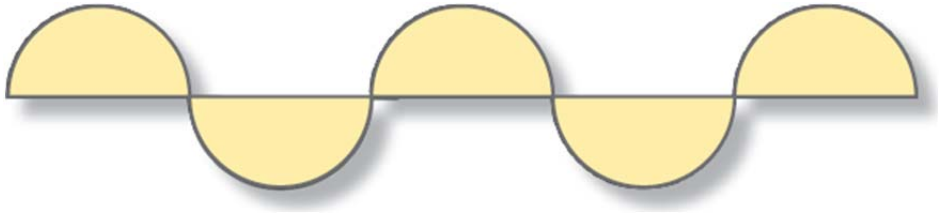
UML

This is relevant in UML to the situations that involve **directionality** in relationships. These, for example, include UML Activity Diagram, UML State Machine Diagram, UML Communication Diagram, UML Interaction Overview Diagram, UML Sequence Diagram, UML Timing Diagram, and UML Use Case Diagram. For example, in UML Sequence Diagram, **stimuli** are perceived as one collective and **responses** (to those stimuli) are perceived as another collective.

GESTALT LAW OF CONTINUITY

Statement: The elements that have the same modal pattern are perceived as a collective. (The modality could, for example, be visual, aural, or tactile.)

EXAMPLE



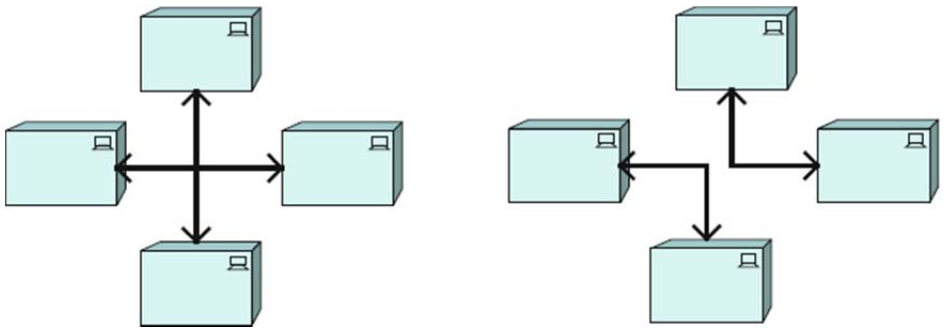
(Source: [Benyon, Turner, Turner, 2005].)

EXAMPLE

The image on the left is perceived as two triangles interrupted by a horizontal line. The image in the center is perceived as an X sign. In the image on the right, the viewer's eye naturally follows the curved line, although the curved line is interrupted and joined to another line.

(Source: [Mazza, 2009].)

EXAMPLE



(Source: [Lankhorst, 2013, Chapter 6].)

UML

This is relevant to the notion of classifiers in UML.

36. COMMUNITY SUPPORT FOR CONCEPTUAL MODELING

Develop Conceptual Models is one of the recommended **software engineering quality practices** [Kandt, 2006].

The **Guide to the Software Engineering Body of Knowledge (SWEBOK)**[15] "describes the sum of knowledge within the profession of software engineering".

SWEBOK continues to **evolve**. SWEBOK2 is an international standard [ISO/IEC, 2005]. The latest version of SWEBOK is SWEBOK3 [IEEE, 2014]. The purpose of the **SWEBOK3** is "to describe what portion of the Body of Knowledge is generally accepted, to organize that portion, and to provide a topical access to it".

SWEBOK has a number of Knowledge Areas (KAs). There is mention of conceptual modeling in the **Requirements Analysis sub-area** of the **Software Requirements Knowledge Area** of the SWEBOK3 [IEEE, 2014, Chapter 1], as illustrated in Figure 11.

[15] URL: http://www.swebok.org/ .

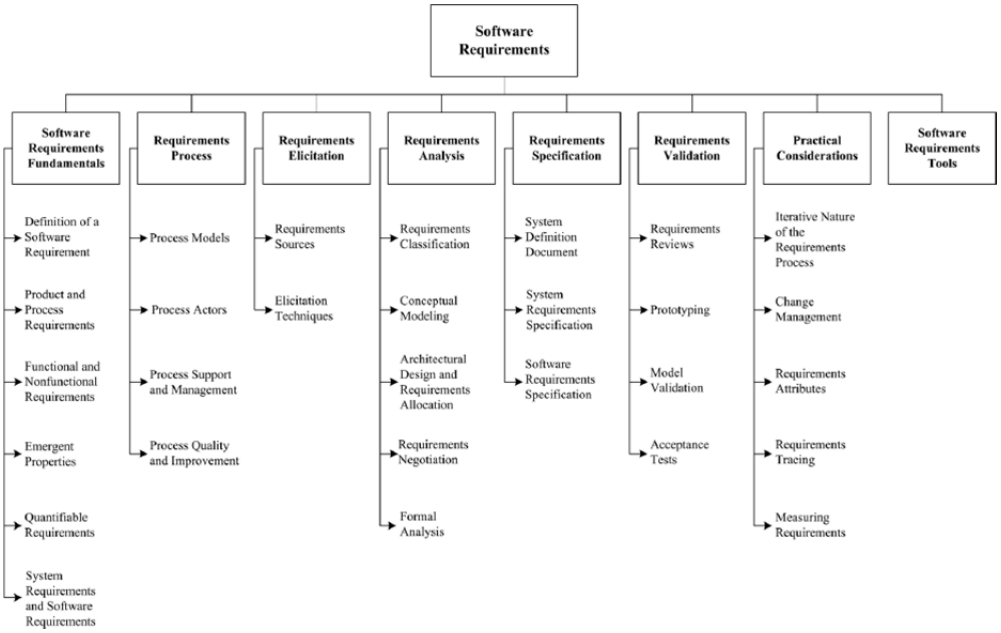


Figure 11. Software Requirements is a Knowledge Area in the Guide to the Software Engineering Body of Knowledge (SWEBOK). (Source: SWEBOK3 [IEEE, 2014].)

The role of modeling, in general, is emphasized by the **Software Engineering Models and Methods** KA, as shown in Figure 12.

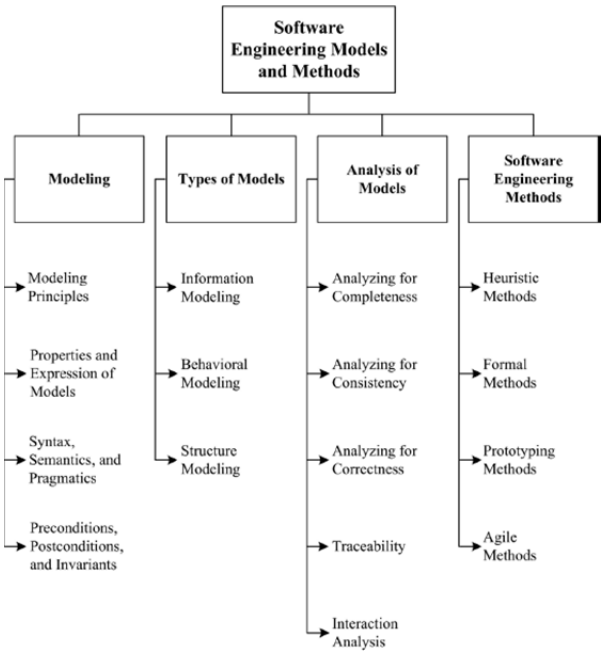


Figure 12. Modeling is a sub-Knowledge Area in the Guide to the Software Engineering Body of Knowledge (SWEBOK). (Source: SWEBOK3 [IEEE, 2014].)

There are a number of means to assess the **state-of-the practice**, and surveys are one of them. The objective in using survey results for some purpose should be **insight**, not numbers.

The results of a number of **surveys** over the years have shown that conceptual modeling is an accepted practice in a variety of software projects [Davies, Green, Rosemann, Indulska, Gallo, 2006].

The industrial adoption and practice of conceptual modeling languages and notations varies [Davies, Green, Rosemann, Indulska, Gallo, 2006].

In particular, the results of one survey[16] are shown in Figure 13. (SBMT stands for Software Based Modeling Tool.)

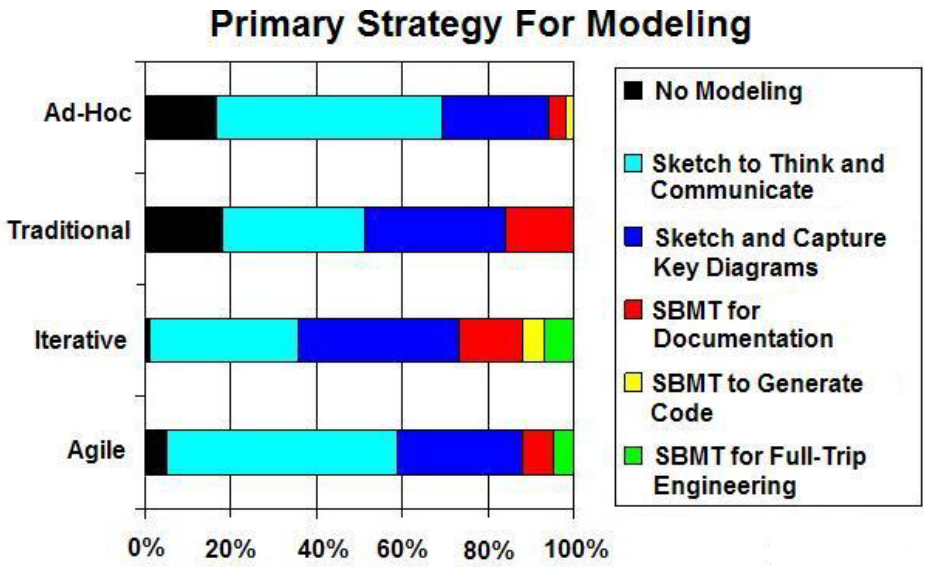


Figure 13. The results of a (conceptual) modeling survey. (Source: The 2008 Modeling and Documentation Survey Results[17].)

[16] URL: http://www.ambysoft.com/surveys/modelingDocumentation2008.html .
[17] URL: http://www.ambysoft.com/surveys/modelingDocumentation2008.html .

OBSERVATIONS

- It has been concluded that conceptual modeling is an **accepted practice** in software projects, **irrespective** of the adopted software development methodology.

- The teams following an agile methodology or carrying out iterative development are **more likely to model** than others.

ACKNOWLEDGEMENT

The contexts in which some of the conceptual and physical models occur are inspired by the work of Peter Grogono. The inclusion in this document of an image from an external source is only for non-commercial educational purposes, and its use is hereby acknowledged.

REFERENCES

[Ambler, 2002] Agile Modeling. By S. W. Ambler. John Wiley and Sons. 2002.

[Ambler, 2004] The Object Primer. By S. W. Ambler. Third Edition. Cambridge University Press. 2004.

[Ambler, 2005] The Elements of UML 2.0 Style. By S. W. Ambler. Cambridge University Press. 2005.

[Arlow, Neustadt, 2004] Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML. By J. Arlow, I. Neustadt. Addison-Wesley. 2004.

[Banerjee, 2014] Mathematical Modeling: Models, Analysis and Applications. By S. Banerjee. CRC Press. 2014.

[Beatty, Chen, 2012] Visual Models for Software Requirements. By J. Beatty, A. Chen. Microsoft Press. 2012.

[Beck, Andres, 2005] Extreme Programming Explained: Embrace Change. By K. Beck, C. Andres. Second Edition. Addison-Wesley. 2005.

[Becker, Delfmann, 2007] Reference Modeling: Efficient Information Systems Design Through Reuse of Information Models. By J. Becker, P. Delfmann (Editors). Physica-Verlag. 2007.

[Bellagamba, 2012] Systems Engineering and Architecting: Creating Formal Requirements. By L. Bellagamba. CRC Press. 2012.

[Benyon, Turner, Turner, 2005] Designing Interactive Systems: People, Activities, Contexts, Technologies. By D. Benyon, P. Turner, S. Turner. Addison-Wesley. 2005.

[Blilie, 2007] The Promise and Limits of Computer Modeling. By C. Blilie. World Scientific. 2007.

[Boca, Bowen, Siddiqi, 2010] Formal Methods: State of the Art and New Directions. By P. P. Boca, J. P. Bowen, J. I. Siddiqi (Editors). Springer-Verlag. 2010.

[Bolloju, Leung, 2006] Assisting Novice Analysts in Developing Quality Conceptual Models with UML. By N. Bolloju, F. S. K. Leung. Communications of the ACM. Volume 49. Issue 7. 2006. Pages 108-112.

[Booch, 2011] Draw Me a Picture. By G. Booch. IEEE Software. Volume 28. Issue 1. 2011. Pages 6-7.

[Booch, Rumbaugh, Jacobson, 2005] The Unified Modeling Language User Guide. By G. Booch, I. Jacobson, J. Rumbaugh. Second Edition. Addison-Wesley. 2005.

[Borgida, Chaudhri, Giorgini, Yu, 2009] Conceptual Modeling: Foundations and Applications. By A. T. Borgida, V. K. Chaudhri, P. Giorgini, E. S. K. Yu. Springer-Verlag. 2009.

[Brambilla, Cabot, Wimmer, 2012] Model-Driven Software Engineering in Practice. By M. Brambilla, J. Cabot, M. Wimmer. Morgan and Claypool. 2012.

[Brambilla, Fraternali, 2015] Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML. By M. Brambilla, P. Fraternali. Morgan Kaufmann. 2015.

[Bryant, Sangwin, 2008] How Round Is Your Circle? Where Engineering and Mathematics Meet. By J. Bryant, C. Sangwin. Princeton University Press. 2008.

[Daniels, 2002] Modeling with a Sense of Purpose. By J. Daniels. IEEE Software. Volume 19. Issue 1. 2002. Pages 8-10.

[Davies, Green, Rosemann, Indulska, Gallo, 2006] How do Practitioners Use Conceptual Modeling in Practice? By I. Davies, P. Green, M. Rosemann, M. Indulska, S. Gallo. Data and Knowledge Engineering. Volume 58. Issue 3. 2006. Pages 358-380.

[Embley, Thalheim, 2011] Handbook of Conceptual Modeling: Theory, Practice, and Research Challenges. By D. W. Embley, B. Thalheim (Editors). Springer-Verlag. 2011.

[Fairbanks, 2010] Just Enough Software Architecture: A Risk-Driven Approach. By G. H. Fairbanks. Marshall and Brainerd. 2010.

[Fernandes, Machado, 2016] Requirements in Engineering Projects. By J. M. Fernandes, R. J. Machado. Springer International Publishing. 2016.

[Garrido, 2016] Introduction to Computational Models with Python. By J. M. Garrido. CRC Press. 2016.

[Ghezzi, Jazayeri, Mandrioli, 2003] Fundamentals of Software Engineering. By C. Ghezzi, M. Jazayeri, D. Mandrioli. Second Edition. Prentice-Hall. 2003.

[Giordano, Fox, Horton, 2014] A First Course in Mathematical Modeling. By F. R. Giordano, W. P. Fox, S. B. Horton. Fifth Edition. Cengage Learning. 2014.

[Gordon, 2004] Theories of Visual Perception. By I. E Gordon. Third Edition. Psychology Press. 2004.

[Graham, 2008] Requirements Modelling and Specification for Service Oriented Architecture. By I. Graham. John Wiley and Sons. 2008.

[Gustafsson, 2011] Fundamentals of Scientific Computing. By B. Gustafsson. Springer-Verlag. 2011.

[Henderson, Johnson, 2012] Conceptual Models: Core to Good Design. By A. Henderson, J. Johnson. Morgan and Claypool. 2012.

[IEEE, 2014] Guide to the Software Engineering Body of Knowledge (SWEBOK) Version 3.0. The Institute of Electrical and Electronics Engineers (IEEE) Computer Society. 2014.

[ISO/IEC, 2005] ISO/IEC TR 19759:2005. Guide to the Software Engineering Body of Knowledge (SWEBOK). The International Organization for Standardization (ISO)/The International Electrotechnical Commission (IEC). 2005.

[Jacobson, Booch, Rumbaugh, 1999] The Unified Software Development Process. By I. Jacobson, G. Booch, J. Rumbaugh. Addison-Wesley. 1999.

[Johnson, 2010] Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Rules. By J. Johnson. Morgan Kaufmann. 2010.

[Johnson, 2014] Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Guidelines. By J. Johnson. Second Edition. Morgan Kaufmann. 2014.

[Kamthan, 2004] A Framework for Addressing the Quality of UML Artifacts. By P. Kamthan. Studies in Communication Sciences. Volume 4. Number 2. 2004. Pages 85-114.

[Kamthan, 2005] Pair Modeling. By P. Kamthan. The 2005 Canadian University Software Engineering Conference (CUSEC 2005). Ottawa, Canada. January 14-16, 2005.

[Kamthan, 2008] Pair Modeling. By P. Kamthan. In: Encyclopedia of Networked and Virtual Organizations. G. D. Putnik, M. M. Cunha (Editors). IGI Global. 2008. Pages 1171-1178.

[Kamthan, 2009] Pattern-Oriented Use Case Modeling. By P. Kamthan. In: Encyclopedia of Information Science and Technology, Second Edition. M. Khosrow-Pour (Editor). IGI Global. 2009.

[Kamthan, Shahmir, 2009] A Pattern-Oriented Process for the Realization of Use Case Models. By P. Kamthan, N. Shahmir. The Thirteenth IBIMA Conference on Knowledge Management and Innovation in Advancing Economies. Marrakech, Morocco. November 9-10, 2009.

[Kandt, 2006] Software Engineering Quality Practices. By R. K. Kandt. Auerbach Publications. 2006.

[Kleppe, Warmer, Bast, 2003] MDA Explained: The Model Driven Architecture™: Practice and Promise. By A. Kleppe, J. Warmer, W. Bast. Addison-Wesley. 2003.

[Kneusel, 2015] Numbers and Computers. By R. T. Kneusel. Springer International Publishing. 2015.

[Kop, Mayr, 2011] Templates in Domain Modeling – A Survey. By C. Kop, H. C. Mayr. In: The Evolution of Conceptual Modeling: From a Historical Perspective towards the Future of Conceptual Modeling. R. Kaschek, L. Delcambre (Editors). Springer-Verlag. 2011. Pages 21-41.

[Kroll, MacIsaac, 2006] Agility and Discipline Made Easy: Practices from OpenUP and RUP. By P. Kroll, B. MacIsaac. Addison-Wesley. 2006.

[Kruchten, 2004] The Rational Unified Process: An Introduction. By P. Kruchten. Third Edition. Addison-Wesley. 2004.

[Lankhorst, 2013] Enterprise Architecture at Work: Modelling, Communication and Analysis. By M. Lankhorst. Third Edition. Springer-Verlag. 2013.

[Larman, 2005] Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. By C. Larman. Third Edition. Prentice-Hall. 2005.

[Leeuwenberg, Helm, 2013] Structural Information Theory: The Simplicity of Visual Form. By E. Leeuwenberg, P. A. van der Helm. Cambridge University Press. 2013.

[Lethbridge, Laganière, 2005] Object-Oriented Software Engineering: Practical Software Development using UML and Java. By T. C. Lethbridge, R. Laganière. McGraw-Hill. 2005.

[Lidwell, Holden, Butler, 2003] Universal Principles of Design. By W. Lidwell, K. Holden, J. Butler. Rockport Publishers. 2003.

[Lidwell, Holden, Butler, 2010] Universal Principles of Design. By W. Lidwell, K. Holden, J. Butler. Second Edition. Rockport Publishers. 2010.

[Lieberman, 2007] The Art of Software Modeling. By B. A. Lieberman. Auerbach Publications. 2007.

[Lindland, Sindre, Sølvberg, 1994] Understanding Quality in Conceptual Modeling. By O. I. Lindland, G. Sindre, A. Sølvberg. IEEE Software. Volume 11. Issue 2. 1994. Pages 42-49.

[Lowdermilk, 2013] User-Centered Design. By T. Lowdermilk. O'Reilly Media. 2013.

[Mazza, 2009] Introduction to Information Visualization. By R. Mazza. Springer-Verlag. 2009.

[Mehmood, Cherfi, 2009] Evaluating the Functionality of Conceptual Models. By K. Mehmood, S. Si-Said Cherfi. The Twenty Eighth International Conference on Conceptual Modeling (ER 2009). Gramado, Brazil. November 9-12, 2009.

[Mellor, Scott, Uhl, Weise, 2004] MDA Distilled: Principles of Model-Driven Architecture. By S. J. Mellor, K. Scott, A. Uhl, D. Weise. Addison-Wesley. 2004.

[Metzger, 2006] Laws of Seeing. By W. Metzger. The MIT Press. 2006.

[Mohagheghi, Dehlen, Neple, 2009] Definitions and Approaches to Model Quality in Model-Based Software Development - A Review of Literature. By P. Mohagheghi, V. Dehlen, T. Neple. Information and Software Technology. Volume 51. Issue 12. 2009. Pages 1646-1669.

[Monson-Haefel, 2009] 97 Things Every Software Architect Should Know: Collective Wisdom from the Experts. By R. Monson-Haefel (Editor). O'Reilly Media. 2009.

[Moody, 2005] Theoretical and Practical Issues in Evaluating the Quality of Conceptual Models: Current State and Future Directions. By D. L. Moody. Data and Knowledge Engineering. Volume 55. Issue 3. 2005. Pages 243-276.

[Moody, 2009] The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. By D. L. Moody. IEEE Transactions on Software Engineering. Volume 35. Issue 6. 2009. Pages 756-779.

[Nelson, Monarchi, 2007] Ensuring the Quality of Conceptual Representations. By H. J. Nelson, D. E. Monarchi. Software Quality Journal. Volume 15. Number 2. 2007. Pages 213-233.

[Nelson, Poels, Genero, Piattini, 2012] A Conceptual Modeling Quality Framework. By H. J. Nelson, G. Poels, M. Genero, M. Piattini. Software Quality Journal. Volume 20. Number 1. 2012. Pages 201-228.

[O'Keefe, 2010] The Meaning of UML Models. By G. O'Keefe. Ph.D. Thesis. The Australian National University. 2010.

[Paternò, 2000] Model-Based Design and Evaluation of Interactive Applications. By F. Paternò. Springer-Verlag. 2000.

[Petre, 1995] Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming. By M. Petre. Communications of the ACM. Volume 38. Issue 6. 1995. Pages 33-44.

[Reed, 2010] Thinking Visually. By S. K. Reed. Psychology Press. 2010.

[Reinhartz-Berger, Sturm, Clark, Cohen, Bettin, 2013] Domain Engineering: Product Lines, Languages, and Conceptual Models. By I. Reinhartz-Berger, A. Sturm, T. Clark, S. Cohen, J. Bettin (Editors). Springer-Verlag. 2013.

[Riel, 1996] Object-Oriented Design Heuristics. By A. J. Riel. Addison-Wesley. 1996.

[Rosen, 2012] Discrete Mathematics and Its Applications. By K. H. Rosen. Seventh Edition. McGraw-Hill. 2012.

[Rosenberg, Stephens, Collins-Cope, 2005] Agile Development with ICONIX Process: People, Process, and Pragmatism. By D. Rosenberg, M. Stephens, M. Collins-Cope. Apress. 2005.

[Rumbaugh, Jacobson, Booch, 2005] The Unified Modeling Language Reference Manual. By J. Rumbaugh, I. Jacobson, G. Booch. Second Edition. Addison-Wesley. 2005.

[Sangwan, 2015] Software and Systems Architecture in Action. By R. S. Sangwan. CRC Press. 2015.

[Schalles, 2013] Usability Evaluation of Modeling Languages: An Empirical Research Study. By C. Schalles. Springer. 2013.

[Seidl, Scholz, Huemer, Kappel, 2015] UML @ Classroom: An Introduction to Object-Oriented Modeling. By M. Seidl, M. Scholz, C. Huemer, G. Kappel. Springer International Publishing. 2015.

[Seidewitz, 2003] What Models Mean. By E. Seidewitz. IEEE Software. Volume 20. Issue 5. 2003. Pages 26-32.

[Selic, 2003] The Pragmatics of Model-Driven Development. By B. Selic. IEEE Software. Volume 20. Issue 5. 2003. Pages 19-25.

[Snoeck, 2014] Enterprise Information Systems Engineering: The MERODE Approach. By M. Snoeck. Springer. 2014.

[Soetaert, Herman, 2009] A Practical Guide to Ecological Modelling: Using R as a Simulation Platform. By K. Soetaert, P. M. J. Herman. Springer Science+Business Media. 2009.

[Spinellis, 2010] UML Everywhere. By D. Spinellis. IEEE Software. Volume 27. Issue 5. 2010. Pages 90-91.

[Stachowiak, 1973] Allgemeine Modelltheorie. By H. Stachowiak. Springer. 1973.

[Stone, 2013] Bayes' Rule: A Tutorial Introduction to Bayesian Analysis. By J. V. Stone. Sebtel Press. 2013.

[Tapp, 2012] Symmetry: A Mathematical Exploration. By K. Tapp. Springer Science+Business Media. 2012.

[Tufte, 1990] Envisioning Information. By E. R. Tufte. Graphics Press. 1990.

[Ware, 2013] Information Visualization: Perception for Design. By C. Ware. Third Edition. Morgan Kaufmann. 2013.

[Warmer, Kleppe, 2003] The Object Constraint Language: Getting Your Models Ready for MDA. By J. Warmer, A. Kleppe. Second Edition. Addison-Wesley. 2003.

[Wazlawick, 2014] Object-Oriented Analysis and Design for Information Systems: Modeling with UML, OCL, and IFML. By R. S. Wazlawick. Morgan Kaufmann. 2014.

[Weinert, 2016] The Demons of Science: What They Can and Cannot Tell Us About Our World. By F. Weinert. Springer International Publishing. 2016.

[Wong, 2006] Different Views of Software Quality. By B. Wong. In: Measuring Information Systems Delivery Quality. E. Duggan, J. Reichgelt (Editors). Idea Group. 2006. Pages 55-88.

APPENDIX A. A COMPARISON BETWEEN A MODEL AND AN ARTIFACT

There are **similarities** and **differences** between an artifact and a model.

The term ‘artifact’ is more general than the term ‘model’. In other words, **all models are artifacts**; however, the **converse** is not necessarily the case.

For example, for a given software project, a data model, design model, or a use case model, are models and therefore are also artifacts. However, the software project management plan, test plan, and the user manual, are artifacts but **not** models.

A glossary in the ‘classical’ sense is at the ‘boundary’ of an artifact and a model.