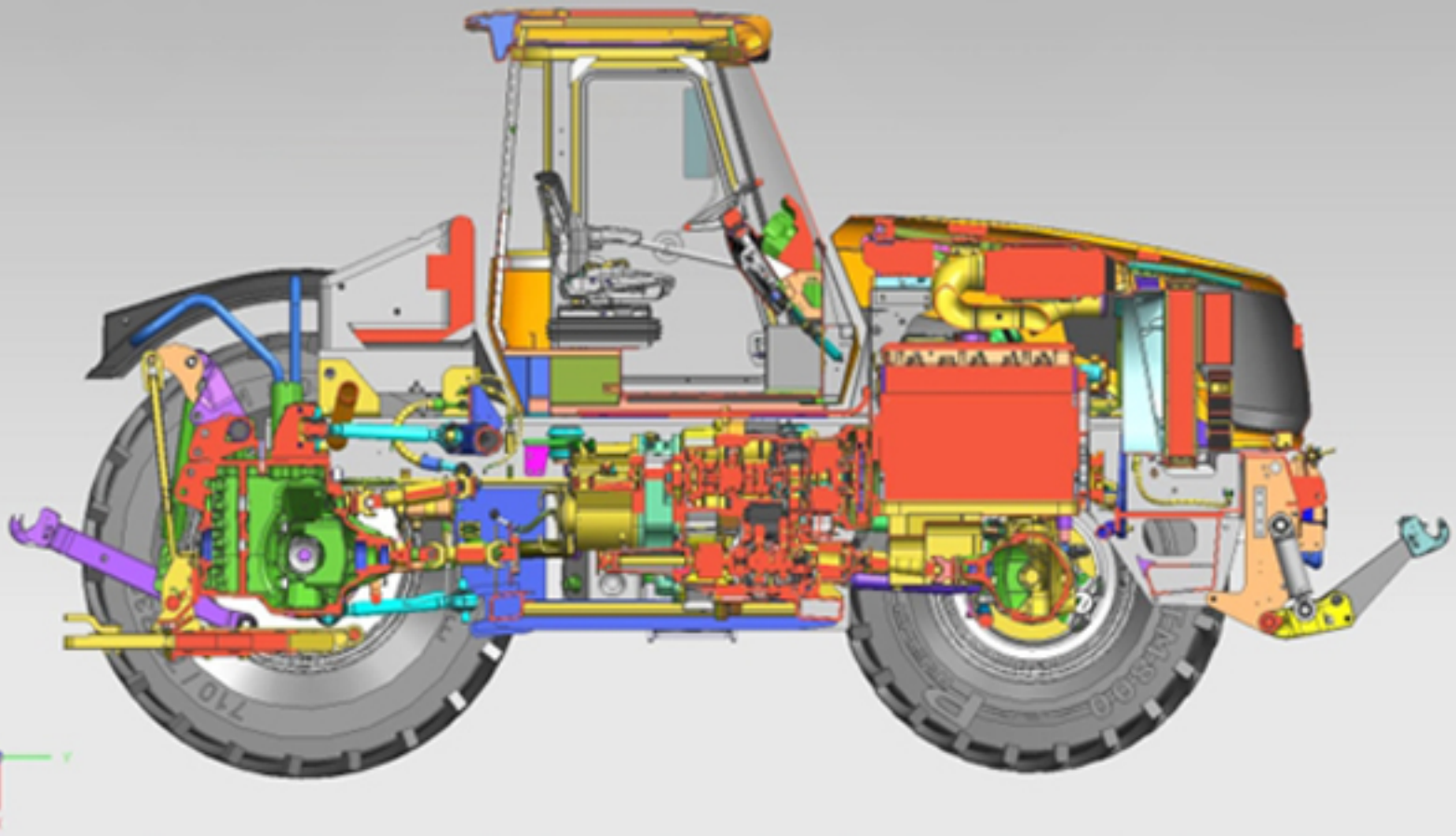
Handbook of
Knowledge Engineering



Kaylah Broadway

Marisa Shay

First Edition, 2012

ISBN 978-81-323-0916-1

Published by:
Academic Studio
4735/22 Prakashdeep Bldg,
Ansari Road, Darya Ganj,
Delhi - 110002
Email: info@wtbooks.com

Table of Contents

Chapter-1

Knowledge Engineering

Knowledge engineering (KE) was defined in 1983 by Edward Feigenbaum, and Pamela McCorduck as follows:

KE is an engineering discipline that involves integrating knowledge into computer systems in order to solve complex problems normally requiring a high level of human expertise.

At present, it refers to the building, maintaining and development of knowledge-based systems. It has a great deal in common with software engineering, and is used in many computer science domains such as artificial intelligence, including databases, data mining, expert systems, decision support systems and geographic information systems. Knowledge engineering is also related to mathematical logic, as well as strongly involved in cognitive science and socio-cognitive engineering where the knowledge is produced by socio-cognitive aggregates (mainly humans) and is structured according to our understanding of how human reasoning and logic works.

Various activities of KE specific for the development of a knowledge-based system:

- Assessment of the problem
- Development of a knowledge-based system shell/structure
- Acquisition and structuring of the related *information*, *knowledge* and specific *preferences* (IPK model)
- Implementation of the structured knowledge into knowledge bases
- Testing and validation of the inserted knowledge
- Integration and maintenance of the system
- Revision and evaluation of the system.

Being still more art than engineering, KE is not as neat as the above list in practice. The phases overlap, the process might be iterative, and many challenges could appear. Recently, emerges meta-knowledge engineering as a new formal systemic approach to the development of a unified knowledge and intelligence theory.

Knowledge engineering principles

Since the mid-1980s, knowledge engineers have developed a number of principles, methods and tools that have considerably improved the process of knowledge acquisition and ordering. Some of the key principles are summarized as follows:

- Knowledge engineers acknowledge that there are different types of knowledge, and that the right approach and technique should be used for the knowledge required.
- Knowledge engineers acknowledge that there are different types of experts and expertise, such that methods should be chosen appropriately.
- Knowledge engineers recognize that there are different ways of representing knowledge, which can aid the acquisition, validation and re-use of knowledge.
- Knowledge engineers recognize that there are different ways of using knowledge, so that the acquisition process can be guided by the project aims (goal-oriented).
- Knowledge engineers use structured methods to increase the efficiency of the acquisition process.
- Knowledge Engineering is the process of eliciting Knowledge for any purpose be it Expert system or AI development

Views of knowledge engineering

There are two main views to knowledge engineering:

- Transfer View – This is the traditional view. In this view, the assumption is to apply conventional knowledge engineering techniques to transfer human knowledge into artificial intelligence systems.
- Modeling View – This is the alternative view. In this view, the knowledge engineer attempts to model the knowledge and problem solving techniques of the domain expert into the artificial intelligence system.

A major concern in knowledge engineering is the construction of ontologies. One philosophical question in this area is the debate between foundationalism and coherentism - are fundamental axioms of belief required, or merely consistency of beliefs which may have no lower-level beliefs to justify them?

Overview of Trends in Knowledge Engineering

Some of the trends in Knowledge Engineering in the last few years are discussed here.The text below is a brief overview of paper "Knowledge Engineering: Principles and methods" authored by Rudi Studer,V.Richard Benjamins and Dieter Fensel.

The paradigm Shift from a transfer view to a modeling view

According to the transfer view the human knowledge required to solve a problem is transferred and implemented into the knowledge base. However this assumes that

concrete knowledge is already present in humans to solve a problem. The transfer view disregards the tacit knowledge an individual acquires in order to solve a problem. This is one of the reasons for a paradigm shift towards modeling view. This shift is compared to a shift from first generation expert systems to second generation expert systems.

The modeling view is a closer approximate of reality and perceives solving problems as a dynamic, cyclic, incessant process dependent on the knowledge acquired and the interpretations made by the system. This is similar to how an expert solves problems in real life.

The evolving of Role Limiting methods and Generic Tasks

Role limiting methods are based on reusable problem solving methods. Different knowledge roles are decided and the knowledge expected from each of these roles is clarified. However the disadvantage of role limiting methods is that there is no logical means of deciding whether a specific problem can be solved by a specific role-limiting method.

This disadvantage gave rise to Configurable role limiting methods. Configurable role limiting methods are based on the idea that a problem solving method can further be broken up into several smaller sub tasks each task solved by its own problem solving method.

Generic Tasks include a rigid knowledge structure, a standard strategy to solve problems, a specific input and a specific output.

The GT approach is based on the strong interaction problem hypothesis which states that the structure and representation of domain knowledge is completely determined by its use

The usage of Modeling Frameworks

The development of Specification languages and problem solving methods of knowledge based systems.Over the past few years the modeling frameworks that became prominent within Knowledge engineering are Common KADS, MIKE (Model-based and Incremental knowledge engineering) and PROTÉGÉ-II.PROTÉGÉ-II is a modeling framework influenced by the concept of ‘Ontology’.

The influence of Ontology

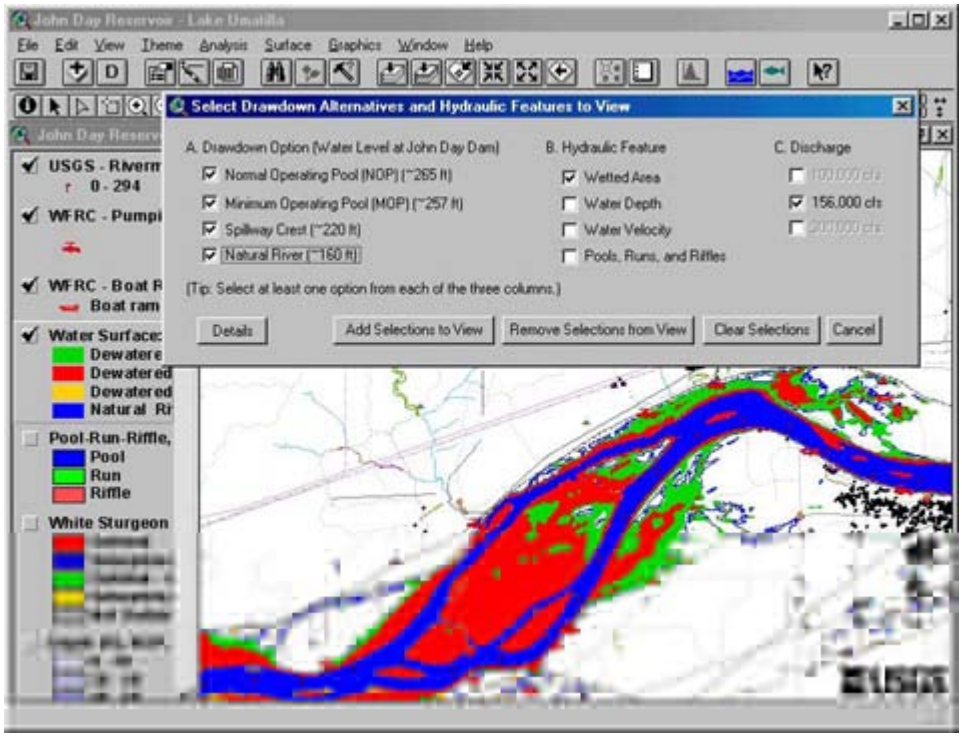
Ontologies help building model of a domain and define the terms inside the domain and the relationships between them. There are different types of Ontologies including Domain ontologies, Generic ontologies, application ontologies and representational ontologies.

While categorizing knowledge, storing, retrieving and managing information is not only useful for solving problems without direct need of human expertise but also leads to

‘Knowledge Management’ efforts that enable an organization to function efficiently in the long run.

Chapter-2

Decision Support System



Example of a Decision Support System for John Day Reservoir.

A **decision support system** (**DSS**) is a computer-based information system that supports business or organizational decision-making activities. DSSs serve the management, operations, and planning levels of an organization and help to make decisions, which may be rapidly changing and not easily specified in advance.

DSSs include knowledge-based systems. A properly designed DSS is an interactive software-based system intended to help decision makers compile useful information from a combination of raw data, documents, personal knowledge, or business models to identify and solve problems and make decisions.

Typical information that a decision support application might gather and present are:

- inventories of information assets (including legacy and relational data sources, cubes, data warehouses, and data marts),
- comparative sales figures between one period and the next,
- projected revenue figures based on product sales assumptions.

History

According to Keen (1978), the concept of decision support has evolved from two main areas of research: The theoretical studies of organizational decision making done at the Carnegie Institute of Technology during the late 1950s and early 1960s, and the technical work on interactive computer systems, mainly carried out at the Massachusetts Institute of Technology in the 1960s. It is considered that the concept of DSS became an area of research of its own in the middle of the 1970s, before gaining in intensity during the 1980s. In the middle and late 1980s, executive information systems (EIS), group decision support systems (GDSS), and organizational decision support systems (ODSS) evolved from the single user and model-oriented DSS.

According to Sol (1987) the definition and scope of DSS has been migrating over the years. In the 1970s DSS was described as "a computer based system to aid decision making". Late 1970s the DSS movement started focusing on "interactive computer-based systems which help decision-makers utilize data bases and models to solve ill-structured problems". In the 1980s DSS should provide systems "using suitable and available technology to improve effectiveness of managerial and professional activities", and end 1980s DSS faced a new challenge towards the design of intelligent workstations.

In 1987 Texas Instruments completed development of the Gate Assignment Display System (GADS) for United Airlines. This decision support system is credited with significantly reducing travel delays by aiding the management of ground operations at various airports, beginning with O'Hare International Airport in Chicago and Stapleton Airport in Denver Colorado.

Beginning in about 1990, data warehousing and on-line analytical processing (OLAP) began broadening the realm of DSS. As the turn of the millennium approached, new Web-based analytical applications were introduced.

The advent of better and better reporting technologies has seen DSS start to emerge as a critical component of management design. Examples of this can be seen in the intense amount of discussion of DSS in the education environment.

DSS also have a weak connection to the user interface paradigm of hypertext. Both the University of Vermont PROMIS system (for medical decision making) and the Carnegie Mellon ZOG/KMS system (for military and business decision making) were decision support systems which also were major breakthroughs in user interface research. Furthermore, although hypertext researchers have generally been concerned with

information overload, certain researchers, notably Douglas Engelbart, have been focused on decision makers in particular.

Taxonomies

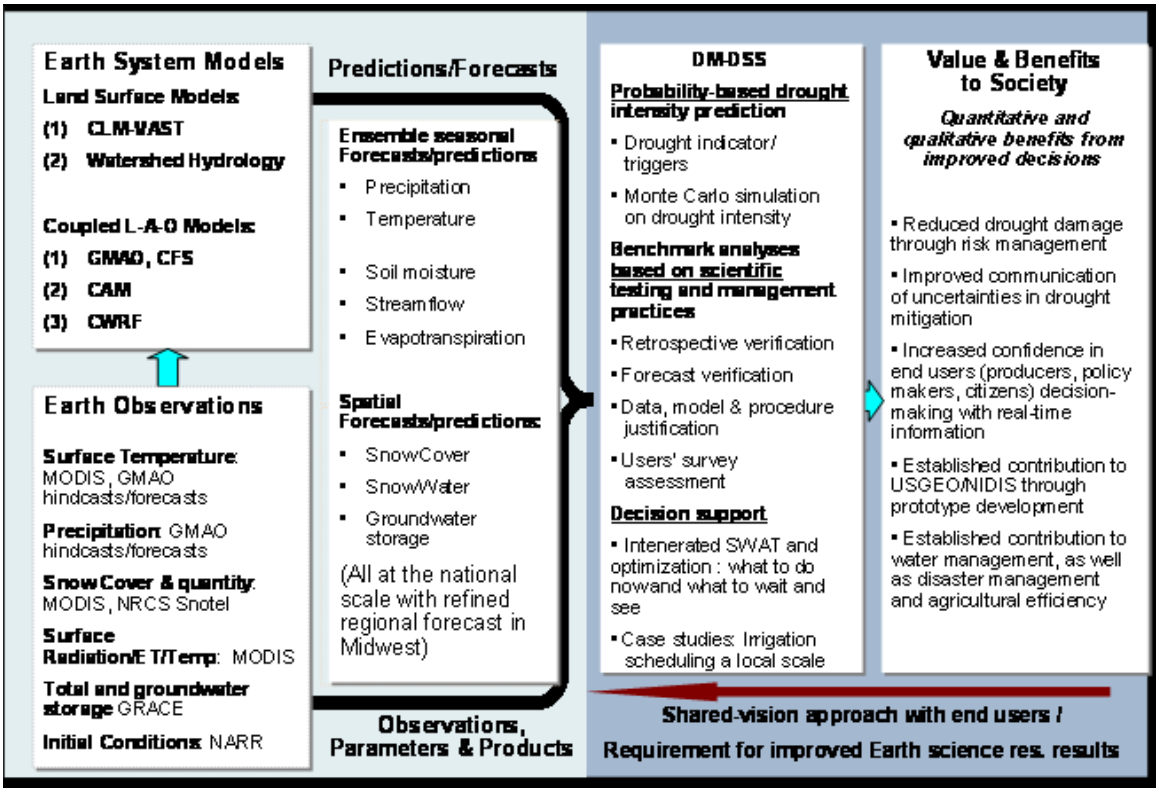
As with the definition, there is no universally-accepted taxonomy of DSS either. Different authors propose different classifications. Using the relationship with the user as the criterion, Haettenschwiler differentiates *passive*, *active*, and *cooperative DSS*. A *passive DSS* is a system that aids the process of decision making, but that cannot bring out explicit decision suggestions or solutions. An *active DSS* can bring out such decision suggestions or solutions. A *cooperative DSS* allows the decision maker (or its advisor) to modify, complete, or refine the decision suggestions provided by the system, before sending them back to the system for validation. The system again improves, completes, and refines the suggestions of the decision maker and sends them back to him for validation. The whole process then starts again, until a consolidated solution is generated.

Another taxonomy for DSS has been created by Daniel Power. Using the mode of assistance as the criterion, Power differentiates *communication-driven DSS*, *data-driven DSS*, *document-driven DSS*, *knowledge-driven DSS*, and *model-driven DSS*.

- A **communication-driven DSS** supports more than one person working on a shared task; examples include integrated tools like Microsoft's NetMeeting or Groove
- A **data-driven DSS** or data-oriented DSS emphasizes access to and manipulation of a time series of internal company data and, sometimes, external data.
- A **document-driven DSS** manages, retrieves, and manipulates unstructured information in a variety of electronic formats.
- A **knowledge-driven DSS** provides specialized problem-solving expertise stored as facts, rules, procedures, or in similar structures.
- A **model-driven DSS** emphasizes access to and manipulation of a statistical, financial, optimization, or simulation model. Model-driven DSS use data and parameters provided by users to assist decision makers in analyzing a situation; they are not necessarily data-intensive. Dicodess is an example of an open source model-driven DSS generator.

Using scope as the criterion, Power differentiates *enterprise-wide DSS* and *desktop DSS*. An *enterprise-wide DSS* is linked to large data warehouses and serves many managers in the company. A *desktop, single-user DSS* is a small system that runs on an individual manager's PC.

Components



Design of a Drought Mitigation Decision Support System.

Three fundamental components of a DSS architecture are:

1. the database (or knowledge base),
2. the model (i.e., the decision context and user criteria), and
3. the user interface.

The users themselves are also important components of the architecture.

Development Frameworks

DSS systems are not entirely different from other systems and require a structured approach. Such a framework includes people, technology, and the development approach.

DSS technology levels (of hardware and software) may include:

1. The actual application that will be used by the user. This is the part of the application that allows the decision maker to make decisions in a particular problem area. The user can act upon that particular problem.
2. Generator contains Hardware/software environment that allows people to easily develop specific DSS applications. This level makes use of case tools or systems such as Crystal, AIMMS, and iThink.

3. Tools include lower level hardware/software. DSS generators including special languages, function libraries and linking modules

An iterative developmental approach allows for the DSS to be changed and redesigned at various intervals. Once the system is designed, it will need to be tested and revised for the desired outcome.

Classification

There are several ways to classify DSS applications. Not every DSS fits neatly into one category, but may be a mix of two or more architectures.

Holsapple and Whinston classify DSS into the following six frameworks: Text-oriented DSS, Database-oriented DSS, Spreadsheet-oriented DSS, Solver-oriented DSS, Rule-oriented DSS, and Compound DSS.

A compound DSS is the most popular classification for a DSS. It is a hybrid system that includes two or more of the five basic structures described by Holsapple and Whinston.

The support given by DSS can be separated into three distinct, interrelated categories: Personal Support, Group Support, and Organizational Support.

DSS components may be classified as:

1. **Inputs:** Factors, numbers, and characteristics to analyze
2. **User Knowledge and Expertise:** Inputs requiring manual analysis by the user
3. **Outputs:** Transformed data from which DSS "decisions" are generated
4. **Decisions:** Results generated by the DSS based on user criteria

DSSs which perform selected cognitive decision-making functions and are based on artificial intelligence or intelligent agents technologies are called Intelligent Decision Support Systems (IDSS).

The nascent field of Decision engineering treats the decision itself as an engineered object, and applies engineering principles such as Design and Quality assurance to an explicit representation of the elements that make up a decision.

Applications

As mentioned above, there are theoretical possibilities of building such systems in any knowledge domain.

One example is the clinical decision support system for medical diagnosis. Other examples include a bank loan officer verifying the credit of a loan applicant or an engineering firm that has bids on several projects and wants to know if they can be competitive with their costs.

DSS is extensively used in business and management. Executive dashboard and other business performance software allow faster decision making, identification of negative trends, and better allocation of business resources.

A growing area of DSS application, concepts, principles, and techniques is in agricultural production, marketing for sustainable development. For example, the DSSAT4 package, developed through financial support of USAID during the 80's and 90's, has allowed rapid assessment of several agricultural production systems around the world to facilitate decision-making at the farm and policy levels. There are, however, many constraints to the successful adoption on DSS in agriculture.

DSS are also prevalent in forest management where the long planning time frame demands specific requirements. All aspects of Forest management, from log transportation, harvest scheduling to sustainability and ecosystem protection have been addressed by modern DSSs. A comprehensive list and discussion of all available systems in forest management is being compiled under the COST action Forsys

A specific example concerns the Canadian National Railway system, which tests its equipment on a regular basis using a decision support system. A problem faced by any railroad is worn-out or defective rails, which can result in hundreds of derailments per year. Under a DSS, CN managed to decrease the incidence of derailments at the same time other companies were experiencing an increase.

Benefits

1. Improves personal efficiency
2. Speed up the process of decision making
3. Increases organizational control
4. Encourages exploration and discovery on the part of the decision maker
5. Speeds up problem solving in an organization
6. Facilitates interpersonal communication
7. Promotes learning or training
8. Generates new evidence in support of a decision
9. Creates a competitive advantage over competition
10. Reveals new approaches to thinking about the problem space
11. Helps automate managerial processes

Chapter-3

NetWeaver Developer

NetWeaver Developer is a knowledgebase development system. This -

1. gives a brief history of the system,
2. summarizes key features of the software,
3. is a bit of a primer, describing basic attributes of a NetWeaver knowledgebase, and
4. provides secondary references that independently document some of the NetWeaver applications developed since the late 1980s.

First, though, a word about knowledgebases. While there are various ways of describing a knowledgebase, perhaps one of the more central concepts is that a knowledgebase provides a formal specification for interpreting information. Formal in this context means that the specification is ontologically committed to the semantics and syntax prescribed by a knowledgebase processor (aka, an engine).

A brief history

NetWeaver was created in late 1991 as a response to ease knowledge engineering tasks by giving a graphical user interface to the ICKEE (IConic Knowledge Engineering Environment) inference engine developed at Penn State University by Bruce J. Miller and Michael C. Saunders. The first iterations were simply a visual representation of dependency networks stored in a LISP-like syntax. NetWeaver quickly evolved into an interactive interface where the visual environment was also capable of editing the dependency networks and saving them in the ICKEE file format. Eventually NetWeaver became "live" in the sense that it could evaluate the dependency networks in real time.

NetWeaver basics

A NetWeaver knowledgebase graphically represents a problem to be evaluated as networks of topics, each of which evaluates a proposition. The formal specification of each topic is graphically constructed, and composed of other topics (e.g., premises) related by logic operators such as and, or, not, etc. NetWeaver topics and operators return a continuous-valued ‘‘truth value’’, that expresses the strength of evidence that the operator and its arguments provide to a topic or to another logic operator. The specification of an individual NetWeaver topic supports potentially complex reasoning

because both topics and logic operators may be specified as arguments to an operator. Considered in its entirety, the complete knowledgebase specification for a problem can be thought of a mental map of the logical dependencies among propositions. In other words, the knowledgebase amounts to a formal logical argument in the classical sense.

When logic meets graphics

Cognitive theory suggests that human beings have two fundamental modes of reasoning: logical (albeit however informally some folks may do that when left to their own devices) and spatial. Interesting things happen when logic is implemented graphically.

First, the knowledge of individual subject-matter experts engaged in [[knowledge engineering]] often is not fully integrated when dealing with complex problems, at least initially. Rather, this knowledge may exist in a somewhat more loosely organized state, a sort of knowledge soup with chunks of knowledge floating about in it. A common observation of knowledge engineers experienced in graphically designing knowledgebases is that the process of constructing a graphic representation of problem-solving knowledge in a formal logical framework seems to be synergistic, with new insights into the expert's knowledge emerging as the process unfolds.

Second, synergies similar to those observed in organizing the reasoning of individual subject-matter experts also can occur in knowledge engineering projects that require the interaction of multiple disciplines. For example, many different kinds of specialists may be involved in evaluating the overall health of a watershed. Use of a formal logic system, with well defined syntax and semantics, allows specialists' representation of their problem solving approach to be expressed in a common language, which in turn facilitates understanding of how all the various perspectives of the different specialists fit together.

About NetWeaver knowledgebases

A NetWeaver knowledgebase has been defined by the developers as a network of networks (Miller and Saunders 2002). Each network corresponds to a topic of interest in the problem being evaluated by the knowledgebase.

NetWeaver knowledgebases are object-based. There are two basic types of objects: networks, and data links, each of which is represented in the logic structure by a programming object which has both state and behavior.

The NetWeaver engine is a Windows dynamic link library (DLL) developed by Rules of Thumb, Inc. (North East, PA). NetWeaver Developer is an interface to the engine that is used for designing knowledgebases.

Logic networks

A knowledgebase represents knowledge about how to solve a problem in terms of the topics of interest in the problem domain, and relations among these topics. Each logic network in a NetWeaver knowledge base represents a proposition about the condition of some ecosystem state or process.

- State - The key state variable of a logic network is its truth value which expresses the degree to which evidence from antecedent networks and data links support or refute the proposition. Logically, network A is said to be antecedent to network B if B depends upon A because network A must be evaluated before network B can be evaluated.
- Behavior - The basic function of a network is to evaluate the truth of its proposition. NetWeaver networks have three basic behaviors related to this function:
    - They query their antecedants to determine the latters' state.
    - They evaluate their own state, given the state of their antecedants.
    - They inform higher level networks that depend on them about their state.

Data links

A data link is an elementary dependency network with slightly modified behavior.

- State - Like a network, a data link may evaluate to a truth value, given a data input. A data link may also hold a data value that is subsequently transformed by mathematical operations defined for a calculated data link.
- Behavior:
    - In NetWeaver Developer, data links prompt the user for data input.
    - On receipt of data, data links evaluate their state, given the data input (simple data links), or pass the data value to a special data link that performs some transformation of input data (calculated data link).
    - They inform higher level networks that depend on them about their state.

Truth values

The truth value is the basic state variable of networks and data links. It expresses an observation's degree of membership in a set. Evaluations of degree of set membership are quantified in the semantics of fuzzy logic. Equivalently, think of the truth value metric as expressing the degree to which evidence supports the proposition of the network or data link; in EMDS, the symbology for maps displaying network truth values is based on the concept of strength of evidence.

Data links are frequently used to read a datum and evaluate its degree of membership in a concept that is quantified in a fuzzy argument (an argument that quantifies fuzzy set membership). Thus, in a data link the argument is a mathematical statement of a proposition. Some simple examples include:

- If the datum fully satisfies the argument, then the truth value of the data link is 1 (full support).
- If the datum is fully contrary to the argument, then the truth value of the data link is -1 (no support).
- If the datum partially satisfies the argument, then the truth value of the data link is in the open interval (-1, 1). Note in particular that negative truth values greater than -1 do not connote negative truth. Rather, such values connote low membership, or low support.
- If the data is not known, then the truth value of the data link is 0 (undetermined).

Interpretation of truth values within networks must be treated more generally, because the truth value of a network may depend on several to many logic operators. Simple examples related to the two key logic operators, AND and OR, are:

- If ‘’’all’’’ logical antecedents to an AND operator fully support the AND relation, then the truth value of the operator is 1 (full support).
- If ‘’’any’’’ logical antecedent to an AND operator is fully contrary to the AND

relation, then the truth value of the operator is -1 (no support).

- If ‘’’any’’’ logical antecedent to an OR operator fully supports the OR relation, then the truth value is 1 (full support).
- If there is no evidence for or against an AND or OR relation, then the truth value of either operator is 0 (undetermined).

As with data links, networks may also evaluate to partially true. Two conditions give rise to this condition in NetWeaver:

- One or more data items are missing and cannot be supplied, and therefore contribute a value of 0 to an AND.
- One or more data items that influence the truth value of a dependency network have been evaluated against a fuzzy argument and found not to have full membership in the fuzzy set defined by the fuzzy argument (the data provides only partial support for the proposition).

Chapter-4

Ontology (Information Science)

In computer science and information science, an **ontology** is a formal representation of knowledge as a set of concepts within a domain, and the relationships between those concepts. It is used to reason about the entities within that domain, and may be used to describe the domain. Its meaning is vastly different from the word Ontology in philosophy.

In theory, an ontology is a "formal, explicit specification of a shared conceptualisation". An ontology provides a shared vocabulary, which can be used to model a domain — that is, the type of objects and/or concepts that exist, and their properties and relations.

Ontologies are the structural frameworks for organizing information and are used in artificial intelligence, the Semantic Web, systems engineering, software engineering, biomedical informatics, library science, enterprise bookmarking, and information architecture as a form of knowledge representation about the world or some part of it. The creation of domain ontologies is also fundamental to the definition and use of an enterprise architecture framework.

Overview

The term *ontology* has its origin in philosophy, and has been applied in many different ways. The word "ontology" comes from the Greek ὄν (on), which literally means 'existence'. The core meaning within computer science is a model for describing the world that consists of a set of types, properties, and relationship types. Exactly what is provided around these varies, but they are the essentials of an ontology. There is also generally an expectation that there be a close resemblance between the real world and the features of the model in an ontology.

What many ontologies have in common in both computer science and in philosophy is the representation of entities, ideas, and events, along with their properties and relations, according to a system of categories. In both fields, one finds considerable work on problems of ontological relativity (e.g., Quine and Kripke in philosophy, Sowa and Guarino in computer science), and debates concerning whether a normative ontology is viable (e.g., debates over foundationalism in philosophy, debates over the Cyc project in AI). Differences between the two are largely matters of focus. Philosophers are less concerned with establishing fixed, controlled vocabularies than are researchers in

computer science, while computer scientists are less involved in discussions of first principles (such as debating whether there are such things as fixed essences, or whether entities must be ontologically more primary than processes).

History

Historically, ontologies arise out of the branch of philosophy known as metaphysics, which deals with the nature of reality – of what exists. This fundamental branch is concerned with analyzing various types or modes of existence, often with special attention to the relations between particulars and universals, between intrinsic and extrinsic properties, and between essence and existence. The traditional goal of ontological inquiry in particular is to divide the world "at its joints", to discover those fundamental categories, or kinds, into which the world's objects naturally fall.

During the second half of the 20th century, philosophers extensively debated the possible methods or approaches to building ontologies, without actually *building* any very elaborate ontologies themselves. By contrast, computer scientists were building some large and robust ontologies (such as WordNet and Cyc) with comparatively little debate over *how* they were built.

Since the mid-1970s, researchers in the field of artificial intelligence (AI) have recognized that capturing knowledge is the key to building large and powerful AI systems. AI researchers argued that they could create new ontologies as computational models that enable certain kinds of automated reasoning. In the 1980s, the AI community began to use the term *ontology* to refer to both a theory of a modeled world and a component of knowledge systems. Some researchers, drawing inspiration from philosophical ontologies, viewed computational ontology as a kind of applied philosophy.

In the early 1990s, the widely cited Web page and paper "Toward Principles for the Design of Ontologies Used for Knowledge Sharing" by Tom Gruber is credited with a deliberate definition of *ontology* as a technical term in computer science. Gruber "introduced the term to mean a specification of a conceptualization. That is "an ontology is a description (like a formal specification of a program) of the concepts and relationships that can formally exist for an agent or a community of agents. This definition is consistent with the usage of ontology as set of concept definitions, but more general. And it is a different sense of the word than its use in philosophy".

According to Gruber (1993) "ontologies are often equated with taxonomic hierarchies of classes, class definitions, and the subsumption relation, but ontologies need not be limited to these forms. Ontologies are also not limited to conservative definitions – that is, definitions in the traditional logic sense that only introduce terminology and do not add any knowledge about the world. To specify a conceptualization, one needs to state axioms that do constrain the possible interpretations for the defined terms.

In the early years of the 21st century, the interdisciplinary project of cognitive science has been bringing the two circles of scholars closer together. For example, there is talk of

a "computational turn in philosophy" that includes philosophers analyzing the formal ontologies of computer science (sometimes even working directly with the software), while researchers in computer science have been making more references to those philosophers who work on ontology (sometimes with direct consequences for their methods). Still, many scholars in both fields are uninvolved in this trend of cognitive science, and continue to work independently of one another, pursuing separately their different concerns.

Ontology components

Contemporary ontologies share many structural similarities, regardless of the language in which they are expressed. As mentioned above, most ontologies describe individuals (instances), classes (concepts), attributes, and relations. Here each of these components is discussed in turn.

Common components of ontologies include:

- Individuals: instances or objects (the basic or "ground level" objects)
- Classes: sets, collections, concepts, classes in programming, types of objects, or kinds of things
- Attributes: aspects, properties, features, characteristics, or parameters that objects (and classes) can have
- Relations: ways in which classes and individuals can be related to one another
- Function terms: complex structures formed from certain relations that can be used in place of an individual term in a statement
- Restrictions: formally stated descriptions of what must be true in order for some assertion to be accepted as input
- Rules: statements in the form of an if-then (antecedent-consequent) sentence that describe the logical inferences that can be drawn from an assertion in a particular form
- Axioms: assertions (including rules) in a logical form that together comprise the overall theory that the ontology describes in its domain of application. This definition differs from that of "axioms" in generative grammar and formal logic. In those disciplines, axioms include only statements asserted as *a priori* knowledge. As used here, "axioms" also include the theory derived from axiomatic statements
- Events: the changing of attributes or relations

Ontologies are commonly encoded using ontology languages.

Domain ontologies and upper ontologies

A domain ontology (or domain-specific ontology) models a specific domain, or part of the world. It represents the particular meanings of terms as they apply to that domain. For example the word *card* has many different meanings. An ontology about the domain of poker would model the "playing card" meaning of the word, while an ontology about the

domain of computer hardware would model the "punched card" and "video card" meanings.

An upper ontology (or foundation ontology) is a model of the common objects that are generally applicable across a wide range of domain ontologies. It employs a core glossary that contains, the terms, and associated object descriptions, as they are used in various, relevant domain sets. There are several standardized upper ontologies available for use, including Dublin Core, GFO, OpenCyc/ResearchCyc, SUMO, and DOLCE. WordNet, while considered an upper ontology by some, is not strictly an ontology. However, it has been employed as a linguistic tool for learning domain ontologies.

The Gellish ontology is an example of a combination of an upper and a domain ontology.

Since domain ontologies represent concepts in very specific and often eclectic ways, they are often incompatible. As systems that rely on domain ontologies expand, they often need to merge domain ontologies into a more general representation. This presents a challenge to the ontology designer. Different ontologies in the same domain can also arise due to different perceptions of the domain based on cultural background, education, ideology, or because a different representation language was chosen.

At present, merging ontologies that are not developed from a common foundation ontology is a largely manual process and therefore time-consuming and expensive. Domain ontologies that use the same foundation ontology to provide a set of basic elements with which to specify the meanings of the domain ontology elements can be merged automatically. There are studies on generalized techniques for merging ontologies, but this area of research is still largely theoretical.

Ontology engineering

Ontology engineering (or ontology building) is a subfield of knowledge engineering that studies the methods and methodologies for building ontologies. It studies the ontology development process, the ontology life cycle, the methods and methodologies for building ontologies, and the tool suites and languages that support them.

Ontology engineering aims to make explicit the knowledge contained within software applications, and within enterprises and business procedures for a particular domain. Ontology engineering offers a direction towards solving the interoperability problems brought about by semantic obstacles, such as the obstacles related to the definitions of business terms and software classes. Ontology engineering is a set of tasks related to the development of ontologies for a particular domain.

Ontology languages

An ontology language is a formal language used to encode the ontology. There are a number of such languages for ontologies, both proprietary and standards-based:

- Common Algebraic Specification Language is a general logic-based specification language developed within the IFIP working group 1.3 "Foundations of System Specifications" and functions as a de facto standard in the area of software specifications. It is now being applied to ontology specifications in order to provide modularity and structuring mechanisms.
- Common logic is ISO standard 24707, a specification for a family of ontology languages that can be accurately translated into each other.
- The Cyc project has its own ontology language called CycL, based on first-order predicate calculus with some higher-order extensions.
- DOGMA (Developing Ontology-Grounded Methods and Applications) adopts the fact-oriented modeling approach to provide a higher level of semantic stability.
- The Gellish language includes rules for its own extension and thus integrates an ontology with an ontology language.
- IDEF5 is a software engineering method to develop and maintain usable, accurate, domain ontologies.
- KIF is a syntax for first-order logic that is based on S-expressions.
- Rule Interchange Format (RIF) and F-Logic combine ontologies and rules.
- OWL is a language for making ontological statements, developed as a follow-on from RDF and RDFS, as well as earlier ontology language projects including OIL, DAML and DAML+OIL. OWL is intended to be used over the World Wide Web, and all its elements (classes, properties and individuals) are defined as RDF resources, and identified by URIs.
- Semantic Application Design Language (SADL) captures a subset of the expressiveness of OWL, using an English-like language entered via an Eclipse Plug-in.
- OBO, a language used for biological and biomedical ontologies.
- (E)MOF and UML are standards of the OMG

Examples of published ontologies

- Basic Formal Ontology, a formal upper ontology designed to support scientific research
- BioPAX, an ontology for the exchange and interoperability of biological pathway (cellular processes) data
- BMO, an e-Business Model Ontology based on a review of enterprise ontologies and business model literature
- CCO (Cell Cycle Ontology), an application ontology that represents the cell cycle
- CContology (Customer Complaint Ontology), an e-business ontology to support online customer complaint management
- CIDOC Conceptual Reference Model, an ontology for cultural heritage
- COSMO, a Foundation Ontology (current version in OWL) that is designed to contain representations of all of the primitive concepts needed to logically specify the meanings of any domain entity. It is intended to serve as a basic ontology that can be used to translate among the representations in other ontologies or databases. It started as a merger of the basic elements of the OpenCyc and SUMO ontologies, and has been supplemented with other ontology elements (types,

relations) so as to include representations of all of the words in the Longman dictionary defining vocabulary.
- Cyc, a large Foundation Ontology for formal representation of the universe of discourse.
- Disease Ontology, designed to facilitate the mapping of diseases and associated conditions to particular medical codes
- DOLCE, a Descriptive Ontology for Linguistic and Cognitive Engineering
- Dublin Core, a simple ontology for documents and publishing
- Foundational, Core and Linguistic Ontologies
- Foundational Model of Anatomy, an ontology for human anatomy
- Friend of a Friend, an ontology for describing persons, their activities and their relations to other people and objects
- Gene Ontology for genomics
- Gellish English dictionary, an ontology that includes a dictionary and taxonomy that includes an upper ontology and a lower ontology that focusses on industrial and business applications in engineering, technology and procurement.
- Geopolitical ontology, an ontology describing geopolitical information created by Food and Agriculture Organization(FAO). The geopolitical ontology includes names in multiple languages (English, French, Spanish, Arabic, Chinese, Russian and Italian); maps standard coding systems (UN, ISO, FAOSTAT, AGROVOC, etc); provides relations among territories (land borders, group membership, etc); and tracks historical changes. In addition, FAO provides web services of geopolitical ontology and a module maker to download modules of the geopolitical ontology into different formats (RDF, XML, and EXCEL).
- GOLD, General Ontology for Linguistic Description
- GUM (Generalized Upper Model), a linguistically-motivated ontology for mediating between clients systems and natural language technology
- IDEAS Group, a formal ontology for enterprise architecture being developed by the Australian, Canadian, UK and U.S. Defence Depts.
- Linkbase, a formal representation of the biomedical domain, founded upon Basic Formal Ontology.
- LPL, Lawson Pattern Language
- NIFSTD Ontologies from the Neuroscience Information Framework: a modular set of ontologies for the neuroscience domain**.**
- OBO Foundry, a suite of interoperable reference ontologies in biomedicine
- Ontology for Biomedical Investigations, an open access, integrated ontology for the description of biological and clinical investigations
- OMNIBUS Ontology, an ontology of learning, instruction, and instructional design
- Plant Ontology for plant structures and growth/development stages, etc.
- POPE, Purdue Ontology for Pharmaceutical Engineering
- PRO, the Protein Ontology of the Protein Information Resource, Georgetown University.
- Program abstraction taxonomy program abstraction taxonomy
- Protein Ontology for proteomics
- Suggested Upper Merged Ontology, a formal upper ontology

- Systems Biology Ontology (SBO), for computational models in biology
- SWEET, Semantic Web for Earth and Environmental Terminology
- ThoughtTreasure ontology
- TIME-ITEM, Topics for Indexing Medical Education
- UMBEL, a lightweight reference structure of 20,000 subject concept classes and their relationships derived from OpenCyc
- WordNet, a lexical reference system
- YAMATO, Yet Another More Advanced Top-level Ontology

The W3C Linking Open Data Community Project coordinates attempts to converge different ontologies into worldwide Data Web.

Ontology libraries

The development of ontologies for the Web has led to the emergence of services providing lists or directories of ontologies with search facility. Such directories have been called ontology libraries.

The following are static libraries of human-selected ontologies.

- DAML Ontology Library maintains a legacy of ontologies in DAML.
- Protege Ontology Library contains a set of OWL, Frame-based and other format ontologies.
- SchemaWeb is a directory of RDF schemata expressed in RDFS, OWL and DAML+OIL.

The following are both directories and search engines. They include crawlers searching the Web for well-formed ontologies.

- OBO Foundry / Bioportal is a suite of interoperable reference ontologies in biology and biomedicine.
- OntoSelect Ontology Library offers similar services for RDF/S, DAML and OWL ontologies.
- Ontaria is a "searchable and browsable directory of semantic web data", with a focus on RDF vocabularies with OWL ontologies. (NB Project "on hold" since 2004).
- Swoogle is a directory and search engine for all RDF resources available on the Web, including ontologies.

Chapter-5

Knowledge Modeling and Ontology Engineering

Knowledge modeling

Knowledge modeling is a process of creating a computer interpretable model of knowledge or standard specifications about a kind of process and/or about a kind of facility or product. The resulting knowledge model can only be computer interpretable when it is expressed in some knowledge representation language or data structure that enables the knowledge to be interpreted by software and to be stored in a database or data exchange file.
Knowledge-based engineering or knowledge-aided design is a process of computer-aided usage of such knowledge models for the design of products, facilities or processes. The design of products or facilities then uses the knowledge model to guide the creation of the facility or product that need to be designed. In other words it used knowledge about a kind of object to create a product model of an (imaginary) individual object. Similarly, the design of a particular process implies the creation of a process model, which design activity can be guided by the knowledge that is contained in a knowledge model about such a kind of process. The resulting process model, product model or facility model is typically also stored in a database.

Usually the knowledge representation language only allows to represent knowledge (about kinds of things), whereas another language or data structure is required to represent and store the information models about individual things. If the knowledge representation language enables to express both, then the knowledge model and the information model can be expressed in the same language (or data structure). An example of a language that enables the expression of knowledge as well as information about individual things is Gellish English.

The basis of a knowledge model of an assembly physical object is a decomposition structure that specifies the components of the assembly and possible the sub-components of the components. For example, knowledge about a compressor system includes that a compressor system consists of a compressor, a lubrication system, etc, whereas a lubrication system consists of a pump system, etc. Assume that this knowledge is expressed in a knowledge representation language that expresses knowledge as a collection of relations between two kinds of things, whereas in that language a relation

type is defined that is called <shall have as part a>. Then a part of a knowledge model about a compressor system will consist of the following expressions of knowledge facts:

- compressor system shall have as part a compressor
- compressor system shall have as part a lubrication system
- lubrication system shall have as part a pump system
- pump system shall have as part a pump

Such a knowledge model will be further extended with knowledge and specifications about the properties of the components, their fabrications and possibly testing and maintenance requirements.

Similarly, a knowledge model of a process is basically a specification of the sequence of process stages. This sequence is determined by the fact that a kind of stream is output of a kind of process stage, whereas that same type of stream in input in the next process stage. So the defined streams have roles as inputs to process stages, whereas the same streams are outputs of other process stages. For example:

- water shall be input in a boiler
- steam shall be output of a boiler
- steam shall be input in a heater
- condensate shall be output of a heater
- etc.

Explicitation of document content

Knowledge modeling includes the explicitation of knowledge and requirements that is available in documents, such as design manuals, (international) standard specifications and standard data sheets. In order to make such knowledge computer interpretable it need to be expressed in a formal knowledge representation language and thus transformed into a computer interpretable form. For example in the form of an expressions Gellish English. This enables that the knowledge and requirements are related to the objects in the knowledge model, whereas the whole model is again stored in a Database.
The knowledge that is contained in documents can be modeled at various levels of explicitation. A low level of explicitation keeps large parts of the specifications in the form of natural language text. This means that the text is only human interpretable, but is nevertheless related to the objects in the knowledge model. Thus software can still present the information to users when knowledge about that object is requested. The other extreme is that the content of each sentence in a documents is converted in the formal knowledge representation language and thus the objects that are mentioned in those sentences become an integral part of the computer interpretable knowledge model. For example, the knowledge that the API 617 standard contains a standard specification for compressors can be linked to the concept compressor in the knowledge model of a compressor system. This can be expressed in a knowledge representation language (using the relation type <is specified in> as follows:

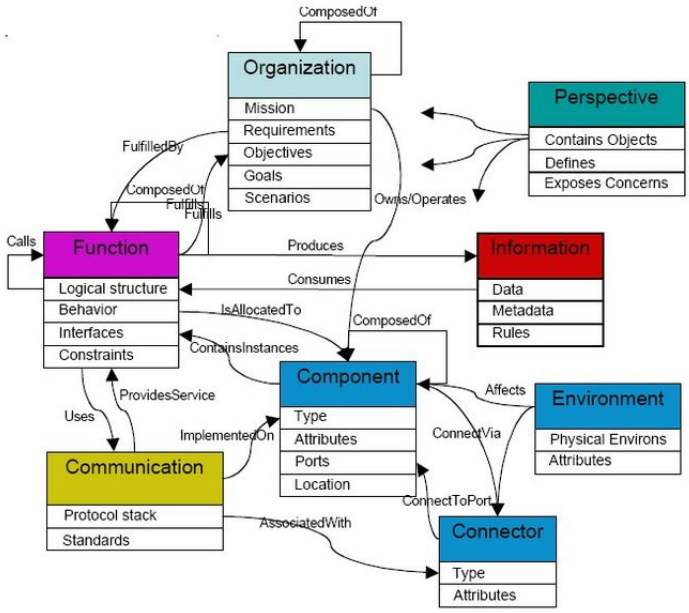
- compressor <is specified in> API 617

A higher level of explicitation means that paragraphs or sentences in natural language are related to components in the knowledge model. A full explicit model means that the natural language sentences are completely transformed into data in a database structure. For example, a specification of a minimum shaft diameter might be included in the knowledge model as follows:

- shaft diameter <shall have on scale a value greater than> 20 mm

The above described explicitation process results in Knowledge Models and Standard Specifications Models that enable their use for computer supported knowledge-aided design as well as for automated verification of designs.

Ontology engineering



Example of a constructed MBED Top Level Ontology based on the Nominal set of views.

Ontology engineering in computer science and information science is a new field, which studies the methods and methodologies for building ontologies: formal representations of a set of concepts within a domain and the relationships between those concepts. A large scale representation of abstract concepts such as actions, time, physical objects and beliefs would be an example of ontological engineering.

Overview

[Ontology engineering] aims at making explicit the knowledge contained within software applications, and within enterprises and business procedures for a particular domain. Ontology engineering offers a direction towards solving the inter-operability problems brought about by semantic obstacles, i.e. the obstacles related to the definitions of business terms and software classes. Ontology engineering is a set of tasks related to the development of ontologies for a particular domain.

Ontologies provide a common vocabulary of an area and define, with different levels of formality, the meaning of the terms and the relationships between them. During the last decade, increasing attention has been focused on ontologies. Ontologies are now widely used in knowledge engineering, artificial intelligence and computer science; in applications related to areas such as knowledge management, natural language processing, e-commerce, intelligent information integration, bio-informatics, education; and in new emerging fields like the semantic web. Ontological engineering is a new field of study concerning the ontology development process, the ontology life cycle, the methods and methodologies for building ontologies, and the tool suites and languages that support them.

Ontology languages

An ontology language is a formal language used to encode the ontology. There are a number of such languages for ontologies, both proprietary and standards-based:

- Common logic is ISO standard 24707, a specification for a family of ontology languages that can be accurately translated into each other.
- The Cyc project has its own ontology language called CycL, based on first-order predicate calculus with some higher-order extensions.
- The Gellish language includes rules for its own extension and thus integrates an ontology with an ontology language.
- IDEF5 is a software engineering method to develop and maintain usable, accurate, domain ontologies.
- KIF is a syntax for first-order logic that is based on S-expressions.
- Rule Interchange Format (RIF) and F-Logic combine ontologies and rules.
- OWL is a language for making ontological statements, developed as a follow-on from RDF and RDFS, as well as earlier ontology language projects including OIL, DAML and DAML+OIL. OWL is intended to be used over the World Wide Web, and all its elements (classes, properties and individuals) are defined as RDF resources, and identified by URIs.
- XBRL (Extensible Business Reporting Language) is a syntax for expressing business semantics.

Ontology Engineering In Life Sciences

Life sciences is flourishing with ontologies that biologists use to make sense of their experiments. For inferring correct conclusions from experiments, ontologies have to be structured optimally against the knowledge base they represent. The structure of an ontology needs to be changed continuously so that it is an accurate representation of the underlying domain.

Recently, an automated method was introduced for engineering ontologies in life sciences such as Gene Ontology (GO), one of the most successful and widely used biomedical ontology. Based on information theory, it restructures ontologies so that the levels represent the desired specificity of the concepts. Similar information theoretic approaches have also been used for optimal partition of Gene Ontology. Given the mathematical nature of such engineering algorithms, these optimizations can be automated to produce a principled and scalable architecture to restructure ontologies such as GO.

Tools for ontology engineering

- DOGMA
- DogmaModeler
- KAON
- OntoClean
- OnToContent
- HOZO
- Protégé (software)

Chapter-6

Knowledge Representation and Reasoning

Knowledge representation (KR) and reasoning' is an area of artificial intelligence whose fundamental goal is to represent knowledge in a manner that facilitates inferencing (i.e. drawing conclusions) from knowledge. It analyzes how to formally think - how to use a symbol system to represent a domain of discourse (that which can be talked about), along with functions that allow inference (formalized reasoning) about the objects. Generally speaking, some kind of logic is used both to supply formal semantics of how reasoning functions apply to symbols in the domain of discourse, as well as to how to supply operators such as quantifiers, modal operators, etc. that, along with an interpretation theory, give meaning to the sentences in the logic.

When we design a knowledge representation (and a knowledge representation system to interpret sentences in the logic in order to derive inferences from them) we have to make choices across a number of design spaces. The single most important decision to be made, is the *expressivity* of the KR. The more expressive, the easier and more compact it is to "say something". However, more expressive languages are harder to automatically derive inferences from. An example of a less expressive KR would be propositional logic. An example of a more expressive KR would be autoepistemic temporal modal logic. Less expressive KRs may be both complete and consistent (formally less expressive than set theory). More expressive KRs may be neither complete nor consistent.

The key problem is to find a KR and a supporting reasoning system that can make the inferences your application needs within the resource constraints appropriate to the problem at hand. Recent developments in KR have been driven by the Semantic Web, and have included development of XML-based knowledge representation languages and standards, including Resource Description Framework (RDF), RDF Schema, Topic Maps, DARPA Agent Markup Language (DAML), Ontology Inference Layer (OIL), and Web Ontology Language (OWL).

Overview

In field there are a number of representation techniques such as frames, rules, tagging, and semantic networks which have originated from theories of human information processing. Since knowledge is used to achieve intelligent behavior, the fundamental goal of knowledge representation is to represent knowledge in a manner which will facilitate

reasoning (aka inferencing or drawing conclusions); knowledge representation and reasoning being seen as two sides of a coin. A good knowledge representation must be both declarative and procedural knowledge. What is knowledge representation can best be understood in terms of five distinct roles it plays, each crucial to the task at hand :

- A knowledge representation (KR) is most fundamentally a surrogate, a substitute for the thing itself, used to enable an entity to determine consequences by thinking rather than acting, i.e., by reasoning about the world rather than taking action in it.
- It is a set of ontological commitments, i.e., an answer to the question: In what terms should I think about the world?
- It is a fragmentary theory of intelligent reasoning, expressed in terms of three components: (i) the representation's fundamental conception of intelligent reasoning; (ii) the set of inferences the representation sanctions; and (iii) the set of inferences it recommends.
- It is a medium for pragmatically efficient computation, i.e., the computational environment in which thinking is accomplished. One contribution to this pragmatic efficiency is supplied by the guidance a representation provides for organizing information so as to facilitate making the recommended inferences.
- It is a medium of human expression, i.e., a language in which we say things about the world."

Some issues that arise in knowledge representation from an AI perspective are:

- How do people represent knowledge?
- What is the nature of knowledge?
- Should a representation scheme deal with a particular domain or should it be general purpose?
- How expressive is a representation scheme or formal language?
- Should the scheme be declarative or procedural?

There has been very little top-down discussion of the knowledge representation (KR) issues and research in this area is a well aged quillwork. There are well known problems such as "spreading activation" (this is a problem in navigating a network of nodes), "subsumption" (this is concerned with selective inheritance; e.g. an ATV can be thought of as a specialization of a car but it inherits only particular characteristics) and "classification." For example a tomato could be classified both as a fruit and a vegetable.

In the field of artificial intelligence, problem solving can be simplified by an appropriate choice of *knowledge representation*. Representing knowledge in some ways makes certain problems easier to solve. For example, it is easier to divide numbers represented in Hindu-Arabic numerals than numbers represented as Roman numerals.

Characteristics

A good knowledge representation covers six basic characteristics:

- Coverage, which means the KR covers a breath and depth of information. Without a wide coverage, the KR cannot determine anything or resolve ambiguities.
- Understandable by humans. KR is viewed as a natural language, so the logic should flow freely. It should support modularity and hierarchies of classes (Polar bears are bears, which are animals). It should also have simple primitives that combine in complex forms.
- Consistency. If John closed the door, it can also be interpreted as the door was closed by John. By being consistent, the KR can eliminate redundant or conflicting knowledge.
- Efficient
- Easy to modify and update.
- Supports the intelligent activity which uses the knowledge base

To gain a better understanding of why these characteristics represent a good knowledge representation, think about how an encyclopedia is structured. There are millions of articles (coverage), and they are sorted into categories, content types, and similar topics (understandable). It redirects different titles but same content to the same article (consistency). It is efficient, easy to add new pages or update existing ones, and allows users on their mobile phones and desktops to view its knowledge base.

History of knowledge representation and reasoning

In computer science, particularly artificial intelligence, a number of representations have been devised to structure information.

KR is most commonly used to refer to representations intended for processing by modern computers, and in particular, for representations consisting of explicit objects (the class of all elephants, or Clyde a certain individual), and of assertions or claims about them ('Clyde is an elephant', or 'all elephants are grey'). Representing knowledge in such explicit form enables computers to draw conclusions from knowledge already stored ('Clyde is grey').

Many KR methods were tried in the 1970s and early 1980s, such as heuristic question-answering, neural networks, theorem proving, and expert systems, with varying success. Medical diagnosis (e.g., Mycin) was a major application area, as were games such as chess.

In the 1980s formal computer knowledge representation languages and systems arose. Major projects attempted to encode wide bodies of general knowledge; for example the "Cyc" project (still ongoing) went through a large encyclopedia, encoding not the information itself, but the information a reader would need in order to understand the

encyclopedia: naive physics; notions of time, causality, motivation; commonplace objects and classes of objects.

Through such work, the difficulty of KR came to be better appreciated. In computational linguistics, meanwhile, much larger databases of language information were being built, and these, along with great increases in computer speed and capacity, made deeper KR more feasible.

Several programming languages have been developed that are oriented to KR. Prolog developed in 1972, but popularized much later, represents propositions and basic logic, and can derive conclusions from known premises. KL-ONE (1980s) is more specifically aimed at knowledge representation itself. In 1995, the Dublin Core standard of metadata was conceived.

In the electronic document world, languages were being developed to represent the structure of documents, such as SGML (from which HTML descended) and later XML. These facilitated information retrieval and data mining efforts, which have in recent years begun to relate to knowledge representation.

Development of the Semantic Web, has included development of XML-based knowledge representation languages and standards, including RDF, RDF Schema, Topic Maps, DARPA Agent Markup Language (DAML), Ontology Inference Layer (OIL), and Web Ontology Language (OWL).

Topics in Knowledge representation and reasoning

Language and notation

Some think it is best to represent knowledge in the same way that it is represented in the human mind, or to represent knowledge in the form of human language.

Psycholinguistics investigates how the human mind stores and manipulates language. Other branches of cognitive science examine how human memory stores sounds, sights, smells, emotions, procedures, and abstract ideas. Science has not yet completely described the internal mechanisms of the brain to the point where they can simply be replicated by computer programmers.

Various artificial languages and notations have been proposed for representing knowledge. They are typically based on logic and mathematics, and have easily parsed grammars to ease machine processing. They usually fall into the broad domain of ontologies.

Ontology Engineering

After CycL, a number of ontology languages have been developed. Most are declarative languages, and are either frame languages, or are based on first-order logic. Most of these

languages only define an upper ontology with generic concepts, whereas the domain concepts are not part of the language definition. These languages all use special-purpose knowledge engineering because as stated by Tom Gruber, "Every ontology is a treaty- a social agreement among people with common motive in sharing." There are always many competing and differing views that make any general purpose ontology impossible. A general purpose ontology would have to be applicable in any domain and different areas of knowledge need to be unified. Gellish English is an example of an ontological language that includes a full engineering English Dictionary.

There is a long history of work attempting to build good ontologies for a variety of task domains, including early work on an ontology for liquids , the lumped element model widely used in representing electronic circuits (e.g.,), as well as ontologies for time, belief, and even programming itself. Each of these offers a way to see some part of the world. The lumped element model, for instance, suggests that we think of circuits in terms of components with connections between them, with signals flowing instantaneously along the connections. This is a useful view, but not the only possible one. A different ontology arises if we need to attend to the electrodynamics in the device: Here signals propagate at finite speed and an object (like a resistor) that was previously viewed as a single component with an I/O behavior may now have to be thought of as an extended medium through which an electromagnetic wave flows.

Ontologies can of course be written down in a wide variety of languages and notations (e.g., logic, LISP, etc.); the essential information is not the form of that language but the content, i.e., the set of concepts offered as a way of thinking about the world. Simply put, the important part is notions like connections and components, not whether we choose to write them as predicates or LISP constructs.

The commitment we make by selecting one or another ontology can produce a sharply different view of the task at hand. Consider the difference that arises in selecting the lumped element view of a circuit rather than the electrodynamic view of the same device. As a second example, medical diagnosis viewed in terms of rules (e.g., MYCIN) looks substantially different from the same task viewed in terms of frames (e.g., INTERNIST). Where MYCIN sees the medical world as made up of empirical associations connecting symptom to disease, INTERNIST sees a set of prototypes, in particular prototypical diseases, to be matched against the case at hand.

Commitment begins with the earliest choices

The INTERNIST example also demonstrates that there is significant and unavoidable ontological commitment even at the level of the familiar representation technologies. Logic, rules, frames, etc., each embody a viewpoint on the kinds of things that are important in the world. Logic, for instance, involves a (fairly minimal) commitment to viewing the world in terms of individual entities and relations between them. Rule-based systems view the world in terms of attribute-object-value triples and the rules of plausible inference that connect them, while frames have us thinking in terms of prototypical objects. Each of these thus supplies its own view of what is important to attend to, and each suggests, conversely, that anything not easily seen in those terms may be ignored. This is of course not guaranteed to be correct, since anything ignored may later prove to be relevant. But the task is hopeless in principle--every representation ignores something

about the world--hence the best we can do is start with a good guess. The existing representation technologies supply one set of guesses about what to attend to and what to ignore. Selecting any of them thus involves a degree of ontological commitment: the selection will have a significant impact on our perception of and approach to the task, and on our perception of the world being modeled.

The commitments accumulate in layers

The ontologic commitment of a representation thus begins at the level of the representation technologies and accumulates from there. Additional layers of commitment are made as we put the technology to work. The use of frame-like structures in INTERNIST offers an illustrative example. At the most fundamental level, the decision to view diagnosis in terms of frames suggests thinking in terms of prototypes, defaults, and a taxonomic hierarchy. But prototypes of what, and how shall the taxonomy be organized? An early description of the system shows how these questions were answered in the task at hand, supplying the second layer of commitment:

The knowledge base underlying the INTERNIST system is composed of two basic types of elements: disease entities and manifestations.... [It] also contains a...hierarchy of disease categories, organized primarily around the concept of organ systems, having at the top level such categories as "liver disease," "kidney disease," etc.

The prototypes are thus intended to capture prototypical diseases (e.g., a "classic case" of a disease), and they will be organized in a taxonomy indexed around organ systems. This is a sensible and intuitive set of choices but clearly not the only way to apply frames to the task; hence it is another layer of ontological commitment.
At the third (and in this case final) layer, this set of choices is instantiated: which diseases will be included and in which branches of the hierarchy will they appear? Ontologic questions that arise even at this level can be quite fundamental. Consider for example determining which of the following are to be considered diseases (i.e., abnormal states requiring cure): alcoholism, homosexuality, and chronic fatigue syndrome. The ontologic commitment here is sufficiently obvious and sufficiently important that it is often a subject of debate in the field itself, quite independent of building automated reasoners.
Similar sorts of decisions have to be made with all the representation technologies, because each of them supplies only a first order guess about how to see the world: they offer a way of seeing but don't indicate how to instantiate that view. As frames suggest prototypes and taxonomies but do not tell us which things to select as prototypes, rules suggest thinking in terms of plausible inferences, but don't tell us which plausible inferences to attend to. Similarly logic tells us to view the world in terms of individuals and relations, but does not specify which individuals and relations to use.
Commitment to a particular view of the world thus starts with the choice of a representation technology, and accumulates as subsequent choices are made about how to see the world in those terms.

Reminder: A KR is not a data structure
Note that at each layer, even the first (e.g., selecting rules or frames), the choices being

made are about representation, not data structures. Part of what makes a language representational is that it carries meaning , i.e., there is a correspondence between its constructs and things in the external world. That correspondence in turn carries with it constraint. A semantic net, for example, is a representation, while a graph is a data structure. They are different kinds of entities, even though one is invariably used to implement the other, precisely because the net has (should have) a semantics. That semantics will be manifest in part because it constrains the network topology: a network purporting to describe family memberships as we know them cannot have a cycle in its parent links, while graphs (i.e., data structures) are of course under no such constraint and may have arbitrary cycles.
While every representation must be implemented in the machine by some data structure, the representational property is in the correspondence to something in the world and in the constraint that

Links and structures

While hyperlinks have come into widespread use, the closely related semantic link is not yet widely used. The mathematical table has been used since Babylonian times. More recently, these tables have been used to represent the outcomes of logic operations, such as truth tables, which were used to study and model Boolean logic, for example. Spreadsheets are yet another tabular representation of knowledge. Other knowledge representations are trees, graphs and hypergraphs, by means of which the connections among fundamental concepts and derivative concepts can be shown.

Visual representations are relatively new in the field of knowledge management but give the user a way to visualise how one thought or idea is connected to other ideas enabling the possibility of moving from one thought to another in order to locate required information.

Notation

The recent fashion in knowledge representation languages is to use XML as the low-level syntax. This tends to make the output of these KR languages easy for machines to parse, at the expense of human readability and often space-efficiency.

First-order predicate calculus is commonly used as a mathematical basis for these systems, to avoid excessive complexity. However, even simple systems based on this simple logic can be used to represent data that is well beyond the processing capability of current computer systems.

Examples of notations:

- DATR is an example for representing lexical knowledge
- RDF is a simple notation for representing relationships between and among objects

Storage and manipulation

One problem in knowledge representation is how to store and manipulate knowledge in an information system in a formal way so that it may be used by mechanisms to accomplish a given task. Examples of applications are expert systems, machine translation systems, computer-aided maintenance systems and information retrieval systems (including database front-ends).

Semantic networks may be used to represent knowledge. Each node represents a concept and arcs are used to define relations between the concepts.The Conceptual graph model is probably the oldest model still alive. One of the most expressive and comprehensively described knowledge representation paradigms along the lines of semantic networks is MultiNet (an acronym for Multilayered Extended Semantic Networks).

From the 1960s, the knowledge frame or just *frame* has been used. Each frame has its own name and a set of **attributes**, or **slots** which contain values; for instance, the frame for *house* might contain a *color* slot, *number of floors* slot, etc.

Using frames for expert systems is an application of object-oriented programming, with inheritance of features described by the "is-a" link. However, there has been no small amount of inconsistency in the usage of the "is-a" link: Ronald J. Brachman wrote a paper titled "What IS-A is and isn't", wherein 29 different semantics were found in projects whose knowledge representation schemes involved an "is-a" link. Other links include the "part-of" link.

Frame structures are well-suited for the representation of schematic knowledge and stereotypical cognitive patterns. The elements of such schematic patterns are weighted unequally, attributing higher weights to the more typical elements of a schema. A pattern is activated by certain expectations: If a person sees a big bird, he or she will classify it rather as a sea eagle than a golden eagle, assuming that his or her "sea-scheme" is currently activated and his "land-scheme" is not.

Frame representations are object-centered in the same sense as semantic networks are: All the facts and properties connected with a concept are located in one place - there is no need for costly search processes in the database.

A behavioral script is a type of frame that describes what happens temporally; the usual example given is that of describing going to a restaurant. The steps include waiting to be seated, receiving a menu, ordering, etc. The different solutions can be arranged in a so-called semantic spectrum with respect to their semantic expressivity.

Chapter-7

Universal Decimal Classification

The **Universal Decimal Classification** is a system of library classification developed by the Belgian bibliographers Paul Otlet and Henri La Fontaine at the end of the 19th century. It is based on the Dewey Decimal Classification, but uses auxiliary signs to indicate various special aspects of a subject and relationships between subjects. It thus contains a significant faceted or analytico-synthetic element, and is used especially in specialist libraries. UDC has been modified and extended through the years to cope with the increasing output in all disciplines of human knowledge, and is still under continuous review to take account of new developments.

The documents classified by UDC may be in any form. They will often be literature, i.e. written documents, but may also be in other media such as films, video and sound recordings, illustrations, maps, and realia such as museum pieces.

UDC classifications use Arabic numerals and are based on the decimal system. Every number is thought of as a decimal fraction with the initial decimal point omitted, which determines filing order. For ease of reading, a UDC identifier is usually punctuated after every third digit. Thus, after 61 "Medical sciences" come the subdivisions 611 to 619; under 611 "Anatomy" come its subdivisions 611.1 to 611.9; under 611.1 come all of its subdivisions before 611.2 occurs, and so on; after 619 comes 620. An advantage of this system is that it is infinitely extensible, and when new subdivisions are introduced, they need not disturb the existing allocation of numbers.

The main categories

- 0 generalities
- 1 philosophy, psychology
- 2 religion, theology
- 3 social sciences
- 4
- 5 natural sciences
- 6 technology
- 7 the arts
- 8 language, linguistics, literature
- 9 geography, biography, history

A document may be classified under a combination of different categories through the use of additional symbols. For example:

Symbol	Symbol name	Meaning	Example
+	plus	addition	e.g. 59+636 zoology and animal breeding
/	stroke	extension	e.g. 592/599 Systematic zoology (everything from 592 to 599 inclusive)
:	colon	relation	e.g. 17:7 Relation of ethics to art
[]	square brackets	algebraic subgrouping	e.g. 311:[622+669](485) statistics of mining and metallurgy in Sweden (the auxiliary qualifies 622+669 considered as a unit)
=	equals	language	e.g. =111 in English; 59=111 Zoology, in English

The design of UDC lends itself to machine readability, and the system has been used both with early automatic mechanical sorting devices, and modern library OPACs. A core version of UDC, with 65,000 subdivisions, is now available in database format, and is called the Master Reference File (MRF). The current full version of the UDC has 220,000 subdivisions.

UDC Classes Table

Main Table

0 Generalities

```
 000 Computer science, knowledge & systems
   001          Science and knowledge in general. Organization of
intellectual work
   002  Documentation. Books. Writings. Authorship
   003  Writing systems and scripts. Including: signs and symbols
   004  Computer science and technology. Computing
     004.2      Computer architecture
     004.3      Computer hardware
     004.4      Software
     004.5      Human-computer interaction
     004.6      Data
     004.7      Computer communication
     004.8      Artificial intelligence
     004.9   Application-oriented computer-based techniques
   005  Management (Revision from 2001)
     005.1      Management Theory
     005.2      Management agents. Mechanisms. Measures
     005.3      Management activities
     005.32     Organizational behaviour. Management psychology
     005.5      Management operations. Direction
     005.6      Quality management. Total quality management (TQM)
     005.7      Organizational management (OM)
     005.9      Fields of management
```

005.92 Records management
005.93 Plant management. Physical resources management
005.94 Knowledge management
005.95/.96 Personnel management. Human Resources management
006 Standardization of products, operations, weights, measures and time
007 Activity and organizing. Information. Communication and control theory generally (cybernetics)
008 Civilization. Culture. Progress
009 Humanities. Arts subjects in general
010 Bibliographies
020 Library and information sciences
030 Encyclopedias & books of facts
040 [Unassigned]
050 Magazines, journals, periodicals & serials
060 Associations and organizations & museums
070 News media, journalism, Mass media & publishing
080 Quotations
090 Manuscripts & rare books

1 Philosophy. Psychology

100 Philosophy
110 Metaphysics
120 Epistemology
130 Parapsychology & ism
140 Philosophical schools of thought
159.9 Psychology
159.91 Psychophysiology (physiological psychology). Mental physiology
159.92 Mental development and capacity. Comparative psychology
159.93 Sensation. Sensory perception
159.94 Executive functions
159.95 Higher mental processes
159.96 Special mental states and processes
159.97 Abnormal psychology. Insanity. Mental deficiency
159.98 Applied psychology
160 Logic
170 Ethics
180 Ancient, medieval & eastern philosophy
190 Modern western philosophy

2 Religion. Theology

200 Religion
21 Prehistoric and primitive religions
22 Religions of the Far East
221 Religions of China
221.3 Taoism
223 Religions of Korea
225 Religions of Japan
23 Religions of the Indian subcontinent
233 Hinduism narrowly
234 Jainism
235 Sikhism
24 Buddhism

```
     241        Hinayana (Theravada) Buddhism
     242        Mahayana Buddhism
     243        Lamaism
     244        Japanese Buddhism
   25  Religions of antiquity. Minor cults and religions
     252        Religions of Mesopotamia
     254        Religions of Iran
     257        Religions of Europe
   26   Judaism
     262        Ashkenazi Judaism
     264        Sephardi Judaism
     265        Orthodox Judaism
     266        Progressive Judaism
     267        Modern movements arising from Judaism
   27   Christianity
     271        Eastern church
     272/279 Western church
       272      Roman Catholic church
       273      Non-Roman Catholic episcopal churches
       274      Protestantism generally. Protestants. Dissenters.
Puritans
       275      Reformed churches
       276      Anabaptists
       277      Free churches. Non-conformists
       278      Other protestant churches
       279      Other Christian movements and churches
   28   Islam
     282        Sunni. Sunnite Islam
     284        Shi'a. Shi'ite Islam
     285        Babi-Baha'i
     286        Baha'i
   29   Modern spiritual movements
 210 Philosophy & theory of religion
 220 Religions of the Far East
 230 Religions of the Indian subcontinent
 240 Buddhism
 250 Religions of antiquity. Minor cults and religions
 260 Judaism
 270 Christianity
 280 Islam
 290 Modern spiritual movements
```

3 Social Sciences

```
 300 Social sciences, sociology & anthropology
 310 Statistics
 320 Political science
 330 Economics
 340 Law
 350 Public administration & military science
 360 Social problems & social services
 370 Education
 380 Commerce, communications & transportation
 390 Customs, etiquette & folklore
```

5 Mathematics and natural sciences

500 Science
510 Mathematics
  510 Fundamental and general consideration of mathematics
  511 Number theory
  512 Algebra
  514 Geometry
  515.1 Topology
  517 Analysis
  519.1 Combinatorial analysis. Graph theory
520 Astronomy
530 Physics
  531 General mechanics. Mechanics of solid and rigid bodies
  532 Fluid mechanics in general. Mechanics of liquids
(hydromechanics)
  533 Mechanics of gases. Aeromechanics. Plasma physics
  534 Vibrations. Acoustics
  535 Optics
  536 Heat. Thermodynamics
  537 Electricity. Magnetism. Electromagnetism
  539 Physical nature of matter
540 Chemistry
  542 Practical laboratory chemistry
  543 Analytical chemistry
  544 Physical chemistry
  546 Inorganic chemistry
  547 Organic chemistry
  548 Crystallography
550 Earth sciences & geology
560 Fossils & prehistoric life
570 Life sciences; biology
580 Plants (Botany)
590 Animals (Zoology)

6 Applied sciences. Medicine. Technology

600 Technology
610 Medicine & health
  611 Medicine
  612 Pharmacy
  613 Biomedical Sciences
  614 Public Health
  615 Kinesitherapy - Physical Training
620 Engineering. Technology in general
  620 Materials testing. Commercial materials. Power stations.
Economics of energy
  621 Mechanical engineering in general. Nuclear technology.
Electrical engineering. Machinery
  621.3 Electrical engineering
  622 Mining
  623 Military engineering
  624 Civil and structural engineering in general
  625 Civil engineering and land transport. Railway engineering.
Highway engineering
  626 Hydraulic engineering in general
  627 Natural waterway, port, harbour and shore engineering.
Navigational, dredging, salvage and
      rescue facilities. Dams and hydraulic power plant

628 Public health engineering. Water. Sanitation. Illuminating engineering
629 Transport vehicle engineering
630 Agriculture
640 Home & family management
650 Management & public relations
660 Chemical engineering
670 Manufacturing
680 Manufacture for specific uses
690 Building & construction

7 The arts. Recreation. Entertainment. Sport

700 Arts
710 Landscaping & area planning
720 Architecture
730 Sculpture, ceramics & metalwork
740 Drawing & decorative arts
750 Painting
760 Graphic arts
770 Photography & computer art
780 Music
790 Sports, games & entertainment

8 Language. Linguistics. Literature

800 General questions. Including: Philology. Rhetoric
810 Linguistics and languages
820 Literature

9 Geography. Biography. History

900 History
910 Geography & travel
920 Biography & genealogy
930 History of ancient world (to ca. 499)
940 History of Europe
950 History of Asia
960 History of Africa
970 History of North America
980 History of South America
990 History of other areas

Table 2 (Geographic Areas, Historical Periods, Persons)

-1 Place in general
-2 Physiographic designation
-3 The ancient world
-4 Europe
-41 British Isles (geographical whole)
-410 United Kingdom of Gt Britain and N Ireland
-410.1 England
-410.3 Wales
-410.5 Scotland
-410.7 Northern Ireland
-415 Ireland (geographical whole)
-417 Republic of Ireland
-430 Germany

-436 Austria
-437.1 Czech Republic
-437.6 Slovak Republic
-438 Poland
-439 Hungary
-44 France
-450 Italy
-4549 San Marino
-45634 Vatican City
-4585 Malta
-460 Spain
-469 Portugal
-47 Former European USSR
-470 Russia
-48 Scandinavia
-480 Finland
-481 Norway
-485- Sweden
-489 Denmark
-492 Netherlands
-493 Belgium
-494 Switzerland
-495 Greece
-497 Balkan States
-5 Asia
-61 Tunisia, Libya
-7 North and Central America
-71 Canada
-72 Mexico
-728 Central America
-729 West Indies
-73 USA
-74 N E States
-75 S E States
-76 S Central States
-77 N Central States
-78 W States
-79 Pacific States
-8 South America
-9 South Pacific and Australia. Arctic. Antarctic

Table 5 (Ethnic and National Groups)

-5 Italians, Romanians, related groups
-591 Romanians
-5994 Ladins
-5998 Sardinians and Corsicans
-59982 Sardinians
-59984 Corsicans

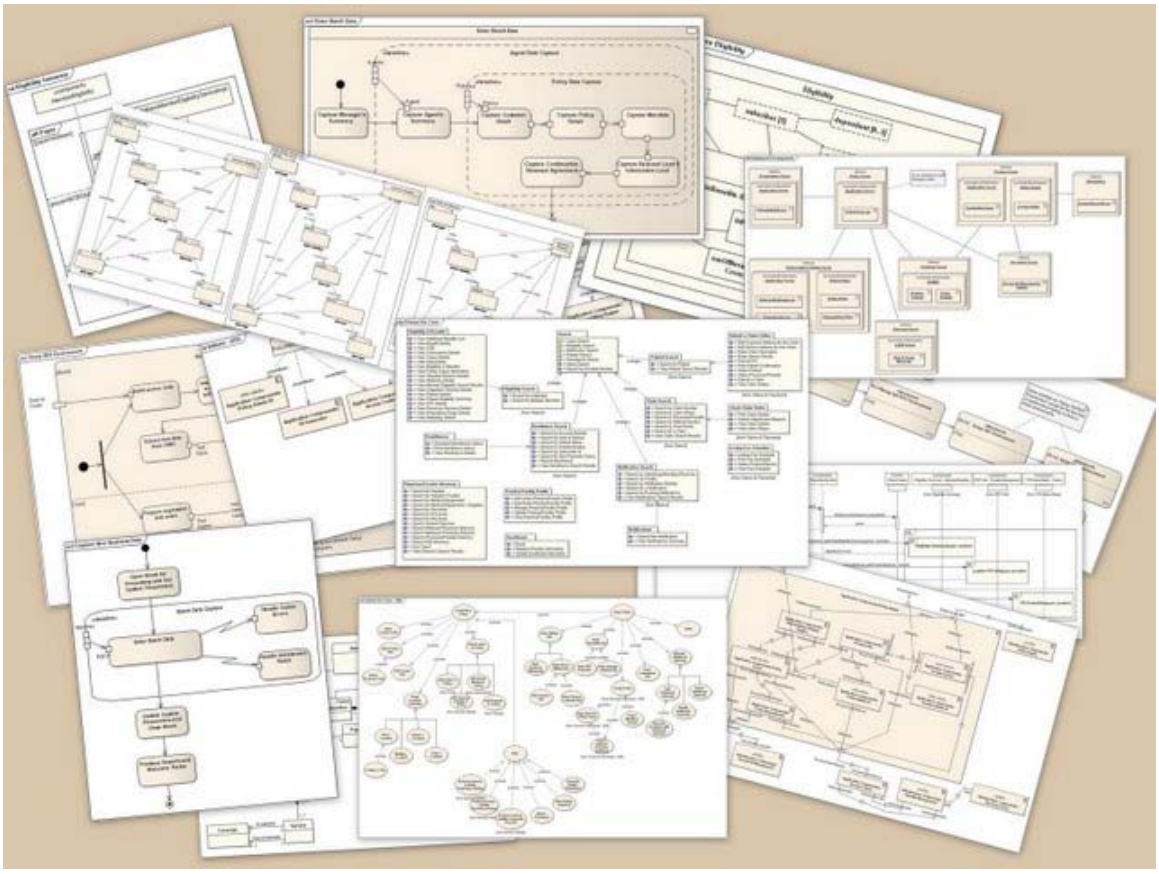
Table 6 (Languages)

-21 English language
-59 Romanian, Rhaetian, Sardinian, Corsican
-5992 Friulian language
-5994 Ladin language
-5996 Romansch language

-59982 Sardinian
-59984 Corsican
-67 Judeo-Spanish (Ladino)
-9455 Sami

Chapter-8

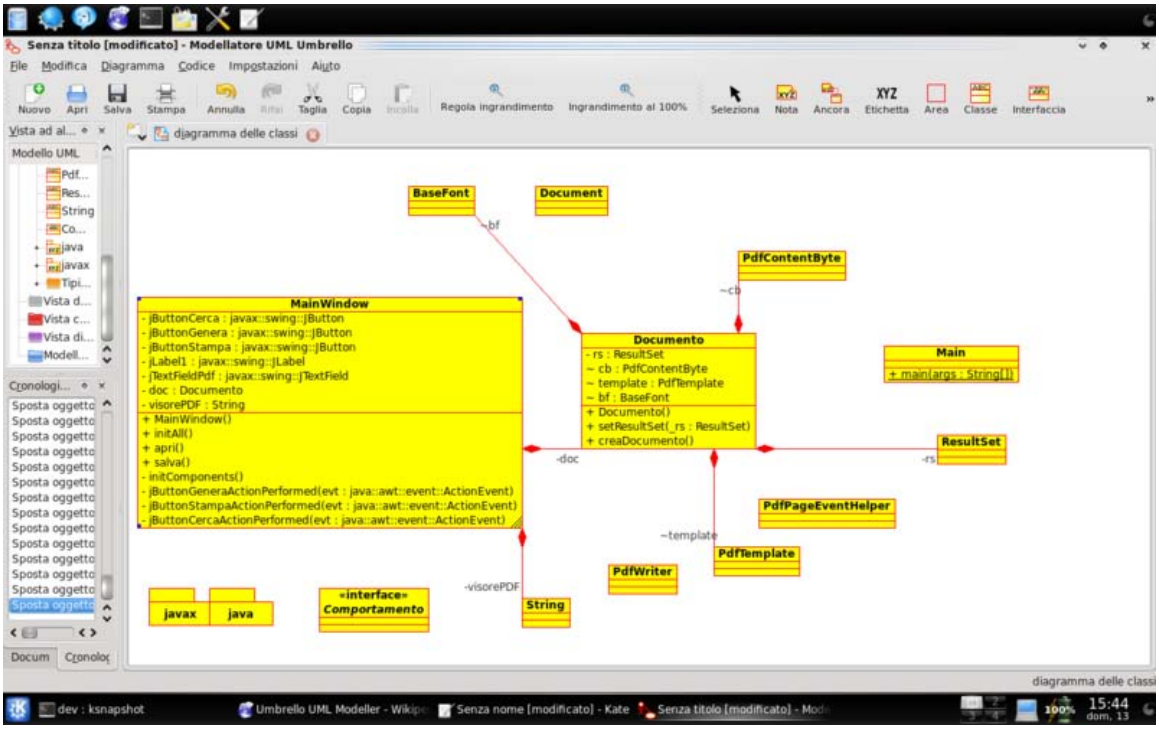
Unified Modeling Language



A collage of UML diagrams.

Unified Modeling Language (**UML**) is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

UML includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems.



Screenshot of Umbrello UML Modeller.

Overview

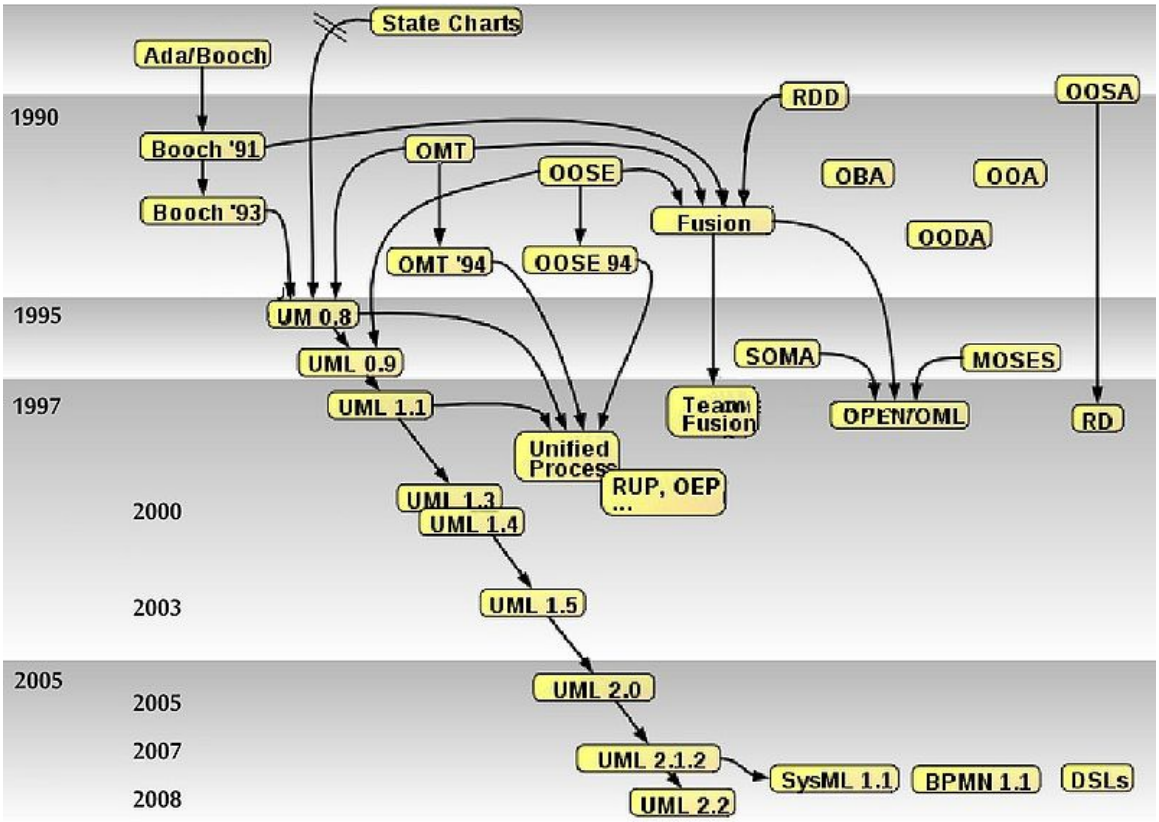
The Unified Modeling Language (UML) is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software-intensive system under development. UML offers a standard way to visualize a system's architectural blueprints, including elements such as:

- activities
- actors
- business processes
- database schemas
- (logical) components
- programming language statements
- reusable software components.

UML combines techniques from data modeling (entity relationship diagrams), business modeling (work flows), object modeling, and component modeling. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies. UML has synthesized the notations of the Booch method, the Object-modeling technique (OMT) and Object-oriented software engineering (OOSE) by fusing them into a single, common and widely usable modeling language. UML aims to be a standard modeling language which can model concurrent and distributed systems. UML is a de facto industry standard, and is evolving under the auspices of the Object Management Group (OMG).

UML models may be automatically transformed to other representations (e.g. Java) by means of QVT-like transformation languages. UML is extensible, with two mechanisms for customization: profiles and stereotypes.

History



History of object-oriented methods and notation.

Before UML 1.x

After Rational Software Corporation hired James Rumbaugh from General Electric in 1994, the company became the source for the two most popular object-oriented modeling approaches of the day: Rumbaugh's Object-modeling technique (OMT), which was better for object-oriented analysis (OOA), and Grady Booch's Booch method, which was better for object-oriented design (OOD). They were soon assisted in their efforts by Ivar Jacobson, the creator of the object-oriented software engineering (OOSE) method. Jacobson joined Rational in 1995, after his company, Objectory AB, was acquired by Rational. The three methodologists were collectively referred to as the *Three Amigos*.

In 1996 Rational concluded that the abundance of modeling languages was slowing the adoption of object technology, so repositioning the work on an unified method, they tasked the Three Amigos with the development of a non-proprietary Unified Modeling Language. Representatives of competing object technology companies were consulted

during OOPSLA '96; they chose *boxes* for representing classes rather than the *cloud* symbols that were used in Booch's notation.

Under the technical leadership of the Three Amigos, an international consortium called the UML Partners was organized in 1996 to complete the *Unified Modeling Language (UML)* specification, and propose it as a response to the OMG RFP. The UML Partners' UML 1.0 specification draft was proposed to the OMG in January 1997. During the same month the UML Partners formed a Semantics Task Force, chaired by Cris Kobryn and administered by Ed Eykholt, to finalize the semantics of the specification and integrate it with other standardization efforts. The result of this work, UML 1.1, was submitted to the OMG in August 1997 and adopted by the OMG in November 1997.

UML 1.x

As a modeling notation, the influence of the OMT notation dominates (e. g., using rectangles for classes and objects). Though the Booch "cloud" notation was dropped, the Booch capability to specify lower-level design detail was embraced. The use case notation from Objectory and the component notation from Booch were integrated with the rest of the notation, but the semantic integration was relatively weak in UML 1.1, and was not really fixed until the UML 2.0 major revision.

Concepts from many other OO methods were also loosely integrated with UML with the intent that UML would support all OO methods. Many others also contributed, with their approaches flavouring the many models of the day, including: Tony Wasserman and Peter Pircher with the "Object-Oriented Structured Design (OOSD)" notation (not a method), Ray Buhr's "Systems Design with Ada", Archie Bowen's use case and timing analysis, Paul Ward's data analysis and David Harel's "Statecharts"; as the group tried to ensure broad coverage in the real-time systems domain. As a result, UML is useful in a variety of engineering problems, from single process, single user applications to concurrent, distributed systems, making UML rich but also large.

The Unified Modeling Language is an international standard:

> ISO/IEC 19501:2005 Information technology – Open Distributed Processing – Unified Modeling Language (UML) Version 1.4.2

UML 2.x

UML has matured significantly since UML 1.1. Several minor revisions (UML 1.3, 1.4, and 1.5) fixed shortcomings and bugs with the first version of UML, followed by the UML 2.0 major revision that was adopted by the OMG in 2005.

Although UML 2.1 was never released as a formal specification, versions 2.1.1 and 2.1.2 appeared in 2007, followed by UML 2.2 in February 2009. UML 2.3 was formally released in May 2010.

There are four parts to the UML 2.x specification:

1. The Superstructure that defines the notation and semantics for diagrams and their model elements
2. The Infrastructure that defines the core metamodel on which the Superstructure is based
3. The Object Constraint Language (OCL) for defining rules for model elements
4. The UML Diagram Interchange that defines how UML 2 diagram layouts are exchanged

The current versions of these standards follow: UML Superstructure version 2.3, UML Infrastructure version 2.3, OCL version 2.2, and UML Diagram Interchange version 1.0.

Although many UML tools support some of the new features of UML 2.x, the OMG provides no test suite to objectively test compliance with its specifications.

Topics

Software development methods

UML is not a development method by itself; however, it was designed to be compatible with the leading object-oriented software development methods of its time (for example OMT, Booch method, Objectory). Since UML has evolved, some of these methods have been recast to take advantage of the new notations (for example OMT), and new methods have been created based on UML, such as IBM Rational Unified Process (RUP). Others include Abstraction Method and Dynamic Systems Development Method.

Modeling

It is important to distinguish between the UML model and the set of diagrams of a system. A diagram is a partial graphic representation of a system's model. The model also contains documentation that drive the model elements and diagrams (such as written use cases).

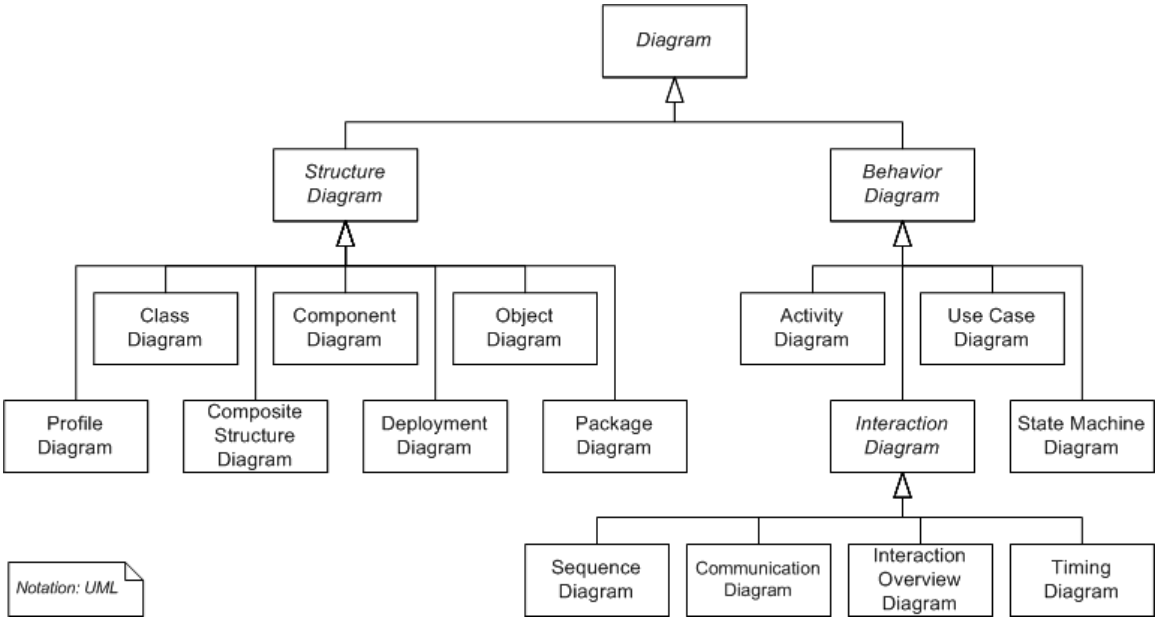
UML diagrams represent two different views of a system model:

- Static (or *structural*) view: emphasizes the static structure of the system using objects, attributes, operations and relationships. The structural view includes class diagrams and composite structure diagrams.
- Dynamic (or *behavioral*) view: emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. This view includes sequence diagrams, activity diagrams and state machine diagrams.

UML models can be exchanged among UML tools by using the XMI interchange format.

Diagrams overview

UML 2.2 has 14 types of diagrams divided into two categories. Seven diagram types represent *structural* information, and the other seven represent general types of *behavior*, including four that represent different aspects of *interactions*. These diagrams can be categorized hierarchically as shown in the following class diagram:



UML does not restrict UML element types to a certain diagram type. In general, every UML element may appear on almost all types of diagrams; this flexibility has been partially restricted in UML 2.0. UML profiles may define additional diagram types or extend existing diagrams with additional notations.

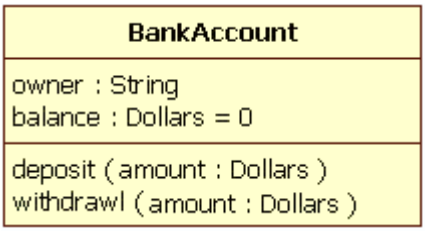
In keeping with the tradition of engineering drawings, a comment or note explaining usage, constraint, or intent is allowed in a UML diagram.

Structure diagrams

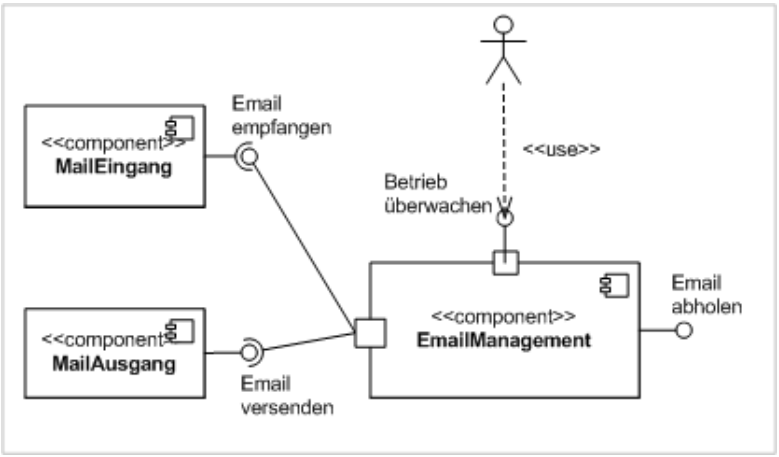
Structure diagrams emphasize the things that must be present in the system being modeled. Since structure diagrams represent the structure, they are used extensively in documenting the software architecture of software systems.

- Class diagram: describes the structure of a system by showing the system's classes, their attributes, and the relationships among the classes.
- Component diagram: describes how a software system is split up into components and shows the dependencies among these components.
- Composite structure diagram: describes the internal structure of a class and the collaborations that this structure makes possible.

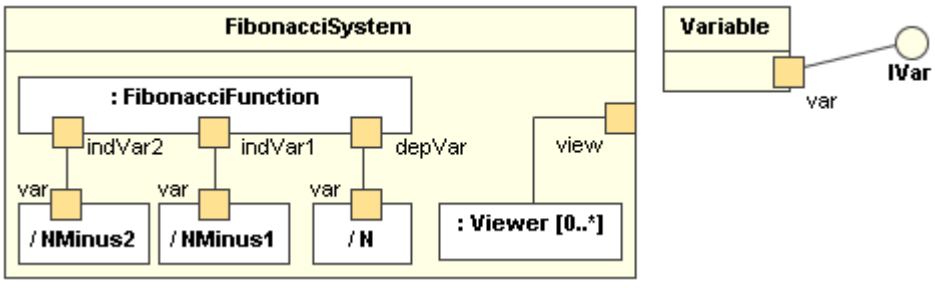
- Deployment diagram: describes the hardware used in system implementations and the execution environments and artifacts deployed on the hardware.
- Object diagram: shows a complete or partial view of the structure of a modeled system at a specific time.
- Package diagram: describes how a system is split up into logical groupings by showing the dependencies among these groupings.
- Profile diagram: operates at the metamodel level to show stereotypes as classes with the <<stereotype>> stereotype, and profiles as packages with the <<profile>> stereotype. The extension relation (solid line with closed, filled arrowhead) indicates what metamodel element a given stereotype is extending.



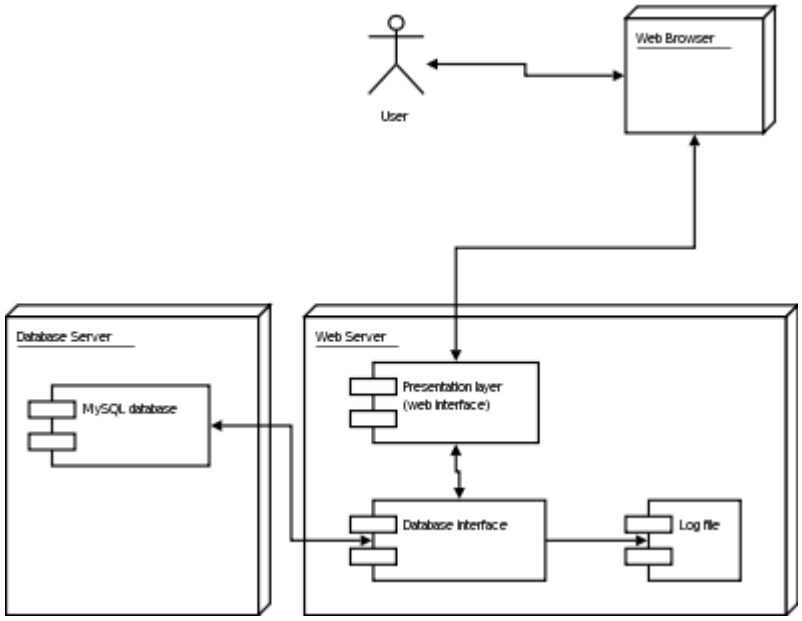
Class diagram



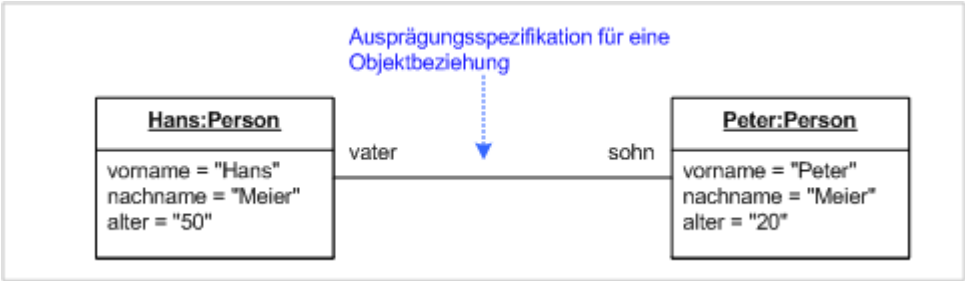
Component diagram



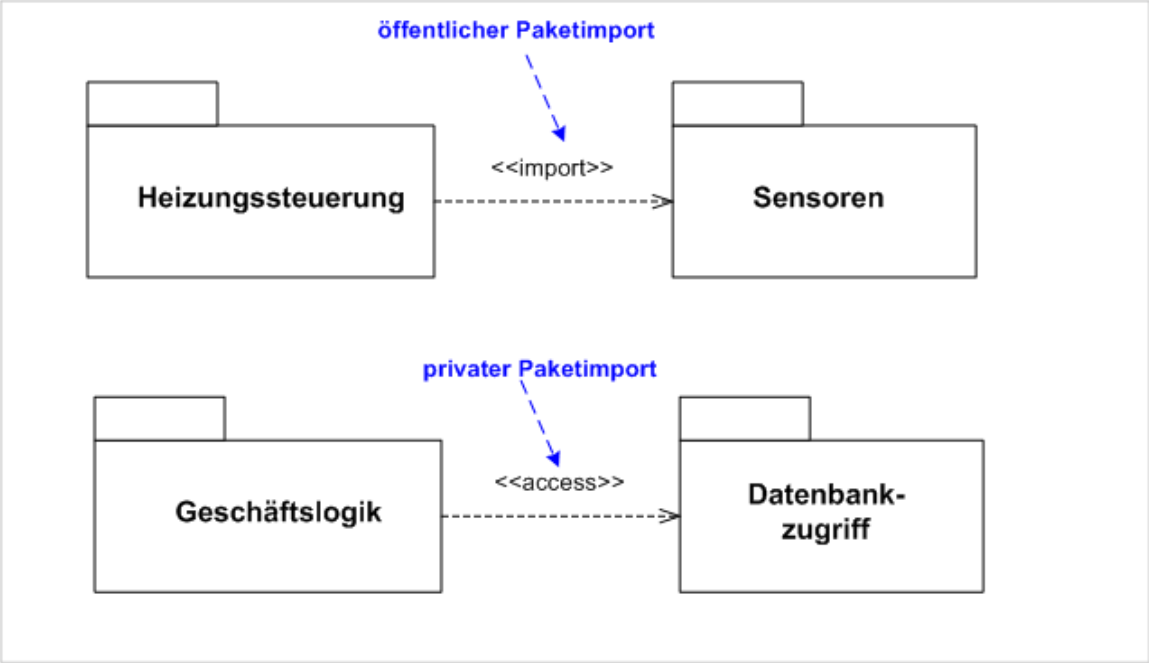
Composite structure diagrams



Deployment diagram



Object diagram

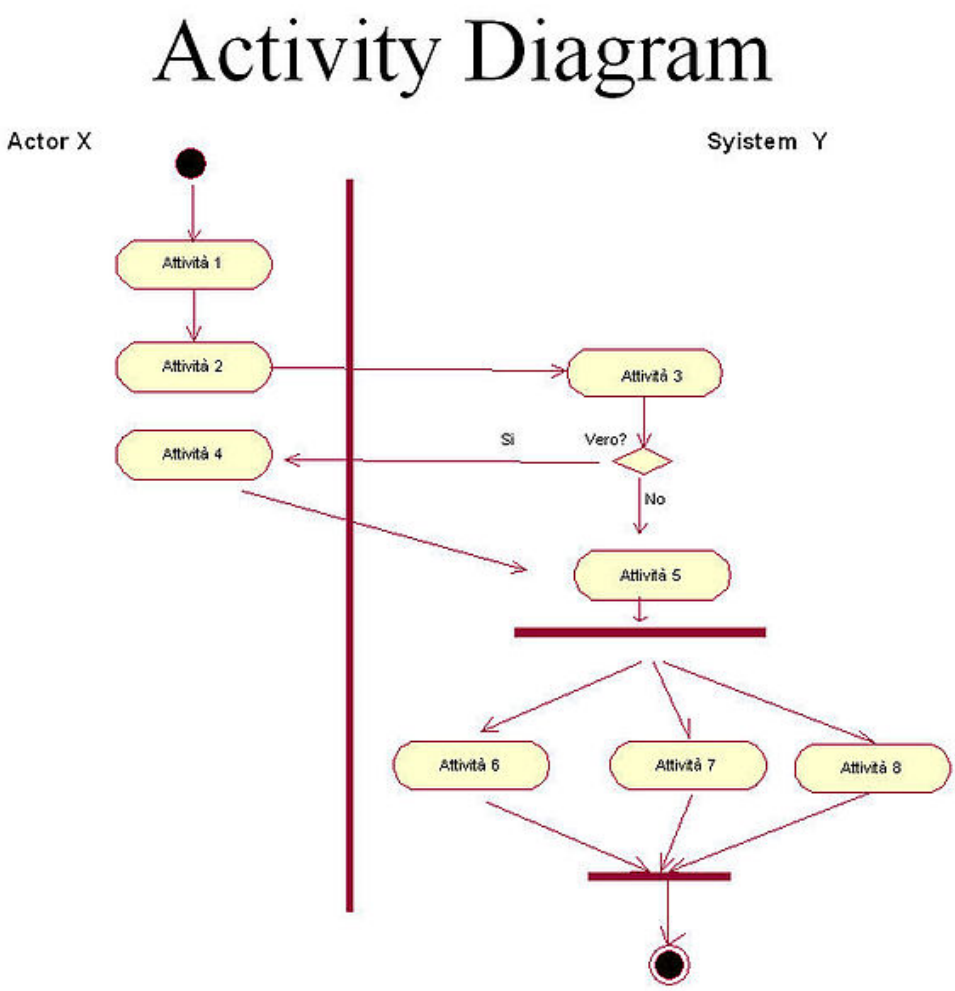


Package diagram

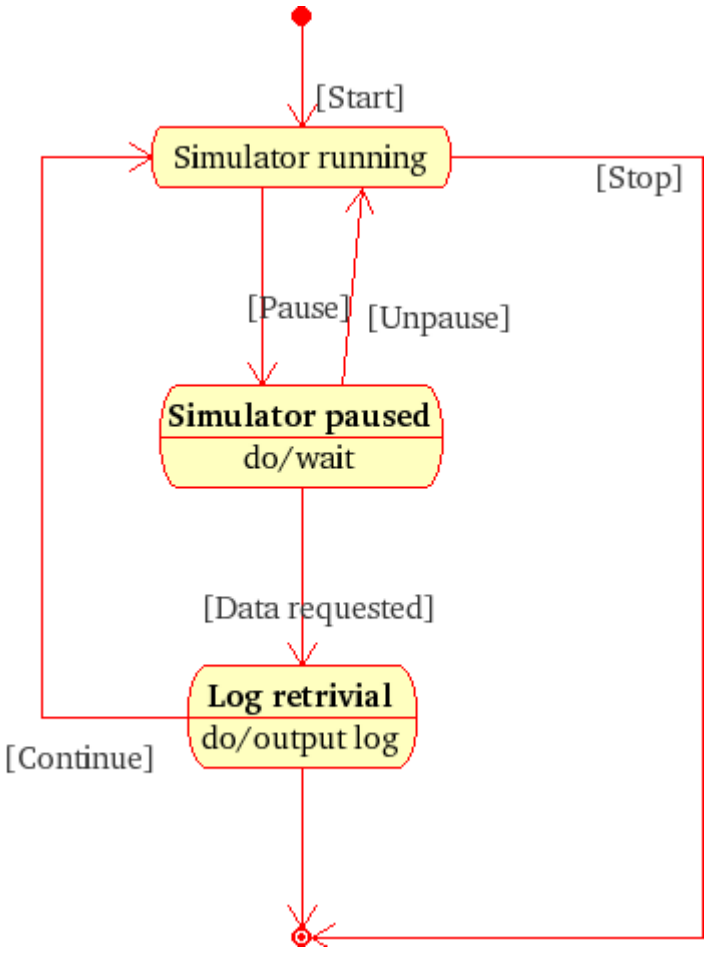
Behaviour diagrams

Behavior diagrams emphasize what must happen in the system being modeled. Since behavior diagrams illustrate the behavior of a system, they are used extensively to describe the functionality of software systems.

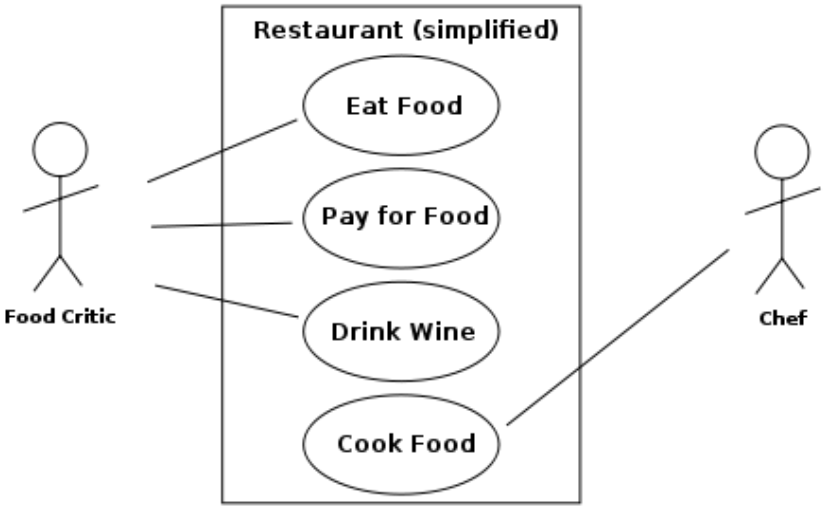
- Activity diagram: describes the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.
- UML state machine diagram: describes the states and state transitions of the system.
- Use case diagram: describes the functionality provided by a system in terms of actors, their goals represented as use cases, and any dependencies among those use cases.



UML Activity Diagram



State Machine diagram

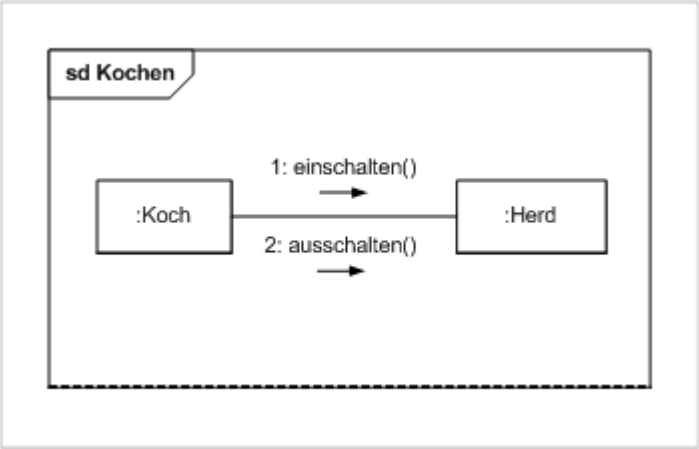


Use case diagram

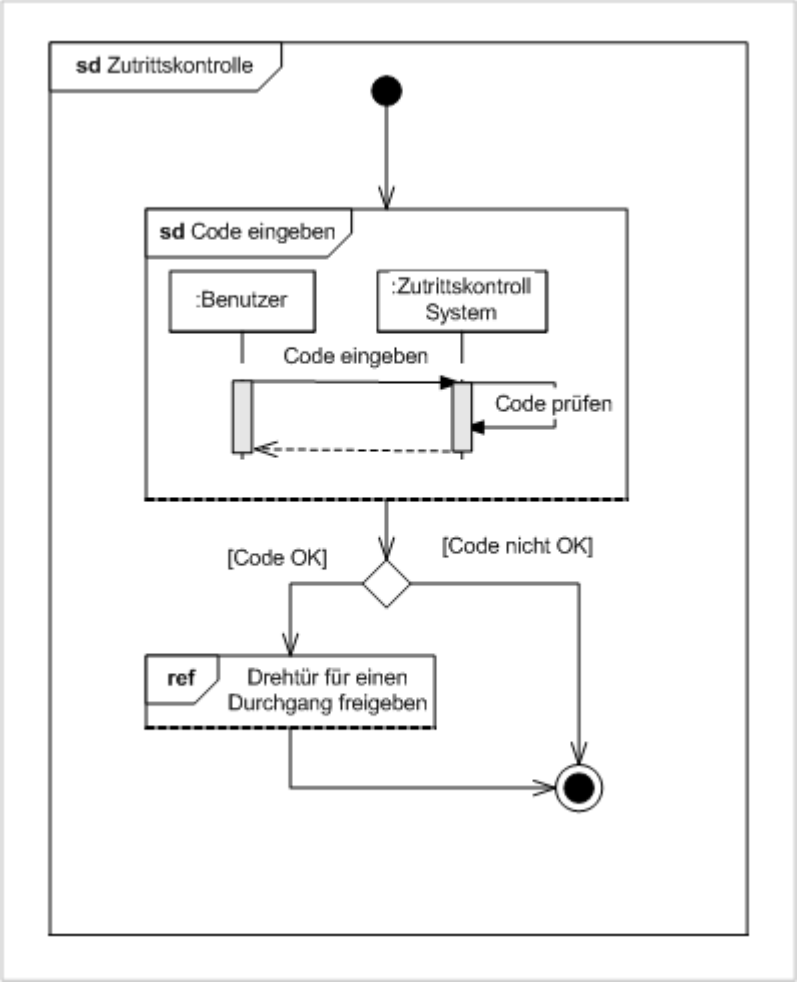
Interaction diagrams

Interaction diagrams, a subset of behaviour diagrams, emphasize the flow of control and data among the things in the system being modeled:

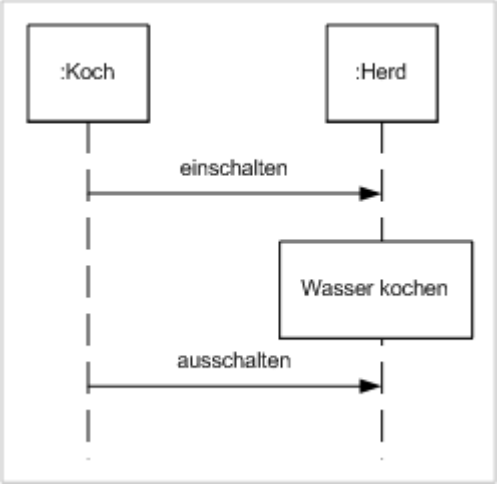
- Communication diagram: shows the interactions between objects or parts in terms of sequenced messages. They represent a combination of information taken from Class, Sequence, and Use Case Diagrams describing both the static structure and dynamic behavior of a system.
- Interaction overview diagram: provides an overview in which the nodes represent communication diagrams.
- Sequence diagram: shows how objects communicate with each other in terms of a sequence of messages. Also indicates the lifespans of objects relative to those messages.
- Timing diagrams: a specific type of interaction diagram where the focus is on timing constraints.



Communication diagram



Interaction overview diagram



Sequence diagram

The Protocol State Machine is a sub-variant of the State Machine. It may be used to model network communication protocols.

Meta modeling

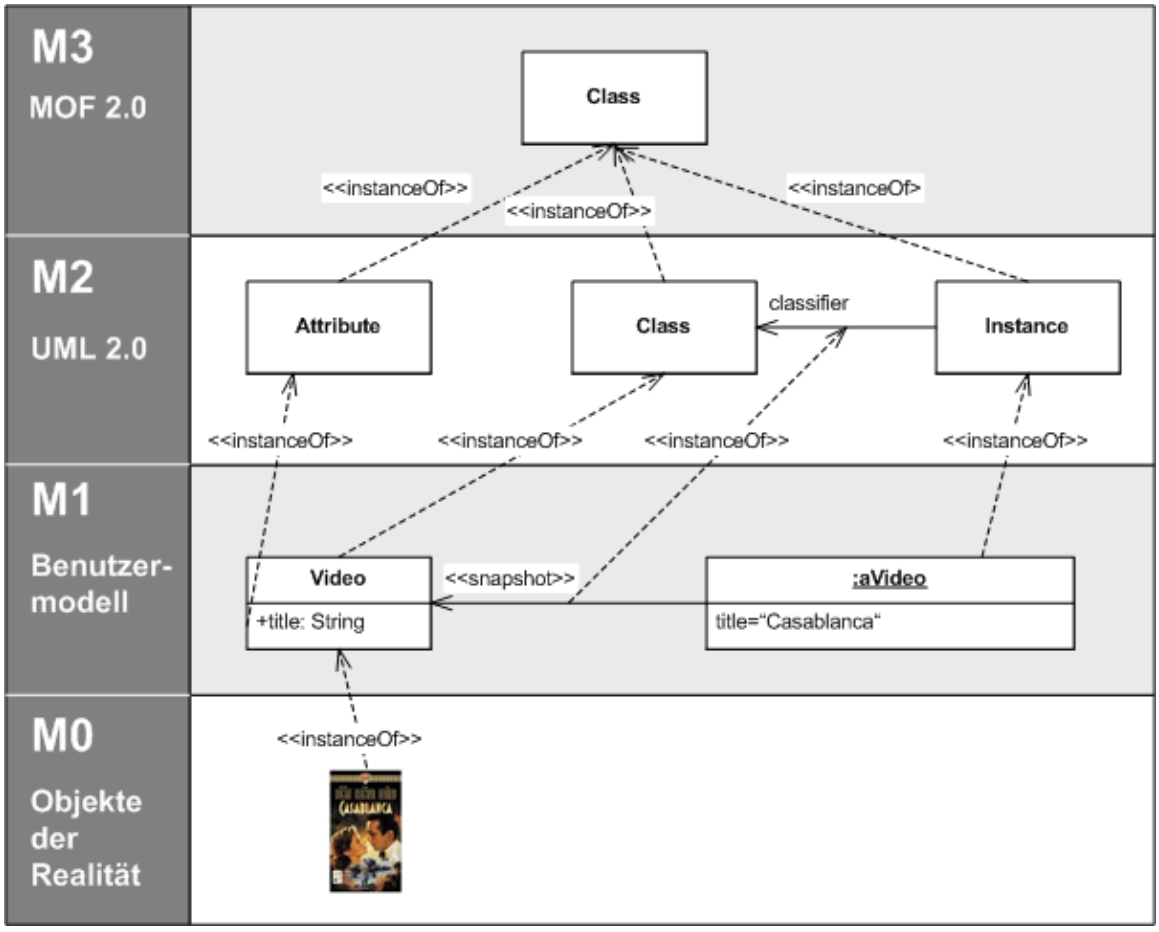


Illustration of the Meta-Object Facility.

The Object Management Group (OMG) has developed a metamodeling architecture to define the Unified Modeling Language (UML), called the Meta-Object Facility (MOF). The Meta-Object Facility is a standard for model-driven engineering, designed as a four-layered architecture, as shown in the image at right. It provides a meta-meta model at the top layer, called the M0 layer. This M0-model is the language used by Meta-Object Facility to build metamodels, called M1-models. The most prominent example of a Layer 1 Meta-Object Facility model is the UML metamodel, the model that describes the UML itself. These M1-models describe elements of the M2-layer, and thus M2-models. These would be, for example, models written in UML. The last layer is the M3-layer or data layer. It is used to describe runtime instance of the system.

Beyond the M0-model, the Meta-Object Facility describes the means to create and manipulate models and metamodels by defining CORBA interfaces that describe those operations. Because of the similarities between the Meta-Object Facility M0-model and

UML structure models, Meta-Object Facility metamodels are usually modeled as UML class diagrams. A supporting standard of the Meta-Object Facility is XMI, which defines an XML-based exchange format for models on the M0-, M1-, or M2-Layer.

Criticisms

Although UML is a widely recognized and used modeling standard, it is frequently criticized for the following:

Standards bloat
: Bertrand Meyer, in a satirical essay framed as a student's request for a grade change, apparently criticized UML as of 1997 for being unrelated to object-oriented software development; a disclaimer was added later pointing out that his company nevertheless supports UML. Ivar Jacobson, a co-architect of UML, said that objections to UML 2.0's size were valid enough to consider the application of intelligent agents to the problem. It contains many diagrams and constructs that are redundant or infrequently used.

Problems in learning and adopting
: The problems cited here make learning and adopting UML problematic, especially when required of engineers lacking the prerequisite skills. In practice, people often draw diagrams with the symbols provided by their CASE tool, but without the meanings those symbols are intended to provide.

Linguistic incoherence
: The extremely poor writing of the UML standards themselves—assumed to be the consequence of having been written by a non-native English speaker—seriously reduces their normative value. In this respect the standards have been widely cited, and indeed pilloried, as prime examples of unintelligible geekspeak.

Capabilities of UML and implementation language mismatch
: As with any notational system, UML is able to represent some systems more concisely or efficiently than others. Thus a developer gravitates toward solutions that reside at the intersection of the capabilities of UML and the implementation language. This problem is particularly pronounced if the implementation language does not adhere to orthodox object-oriented doctrine, as the intersection set between UML and implementation language may be that much smaller.

Dysfunctional interchange format
: While the XMI (XML Metadata Interchange) standard is designed to facilitate the interchange of UML models, it has been largely ineffective in the practical interchange of UML 2.x models. This interoperability ineffectiveness is attributable to two reasons. Firstly, XMI 2.x is large and complex in its own right, since it purports to address a technical problem more ambitious than exchanging UML 2.x models. In particular, it attempts to provide a mechanism for facilitating the exchange of any arbitrary modeling language defined by the OMG's Meta-Object Facility (MOF). Secondly, the UML 2.x Diagram Interchange specification lacks sufficient detail to facilitate reliable interchange of UML 2.x notations between modeling tools. Since UML is a visual modeling language, this shortcoming is substantial for modelers who don't want to redraw their diagrams.

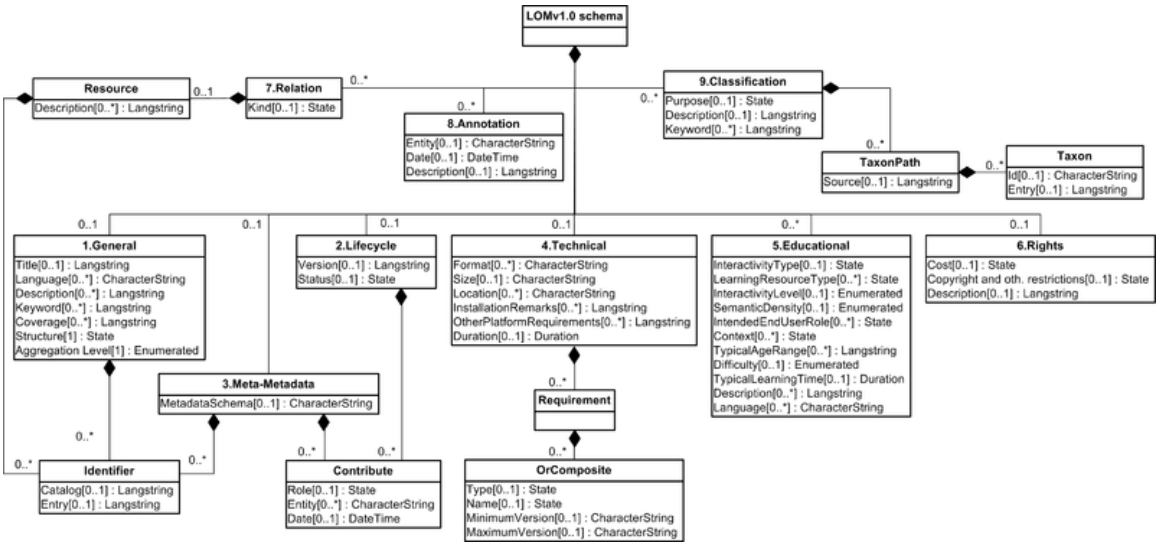
Modeling experts have written sharp criticisms of UML, including Bertrand Meyer's "UML: The Positive Spin", and Brian Henderson-Sellers and Cesar Gonzalez-Perez in "Uses and Abuses of the Stereotype Mechanism in UML 1.x and 2.0".

UML modelling tools

The most well-known UML modelling tool is IBM Rational Rose. Other tools include Rational Rhapsody, MagicDraw UML, StarUML, ArgoUML, Umbrello, BOUML, PowerDesigner, and Dia. Some of popular development environments also offer UML modelling tools, e.g.: Eclipse, NetBeans, and Visual Studio.

Chapter-9

Learning Object Metadata



A schematic representation of the hierarchy of elements in the LOM data model

Learning Object Metadata is a data model, usually encoded in XML, used to describe a learning object and similar digital resources used to support learning. The purpose of learning object metadata is to support the reusability of learning objects, to aid discoverability, and to facilitate their interoperability, usually in the context of online learning management systems (LMS).

The IEEE 1484.12.1 – 2002 Standard for Learning Object Metadata is an internationally-recognised open standard (published by the Institute of Electrical and Electronics Engineers Standards Association, New York) for the description of “learning objects”. Relevant attributes of learning objects to be described include: type of object; author; owner; terms of distribution; format; and pedagogical attributes, such as teaching or interaction style.

IEEE 1484.12.1 – 2002 Standard for Learning Object Metadata

In brief

The IEEE working group that developed the standard defined learning objects, *for the purposes of the standard,* as being "any entity, digital or non-digital, that may be used for learning, education or training." This definition has struck many commentators as being rather broad in its scope, but the definition was intended to provide a broad class of objects to which LOM metadata might usefully be associated rather than to give an instructional or pedagogic definition of a learning object. *IEEE 1484.12.1* is the first part of a multipart standard, and describes the LOM data model. The LOM data model specifies which aspects of a learning object should be described and what vocabularies may be used for these descriptions; it also defines how this data model can be amended by additions or constraints. Other parts of the standard are being drafted to define bindings of the LOM data model, i.e. define how LOM records should be represented in XML and RDF (*IEEE 1484.12.3* and *IEEE 1484.12.4* respectively). Here we, focuses on the LOM data model rather than issues relating to XML or other bindings.

IMS Global Learning Consortium is an international consortium that contributed to the drafting of the IEEE Learning Object Metadata (together with the ARIADNE Foundation) and endorsed early drafts of the data model as part of the IMS Learning Resource Meta-data specification (IMS LRM, versions 1.0 – 1.2.2). Feedback and suggestions from the implementers of IMS LRM fed into the further development of the LOM, resulting in some drift between version 1.2 of the IMS LRM specification and what was finally published at the LOM standard. Version 1.3 of the IMS LRM specification realigns the IMS LRM data model with the IEEE LOM data model and specifies that the IEEE XML binding should be used. Thus, we can now use the term 'LOM' in referring to both the IEEE standard and version 1.3 of the IMS specification. The IMS LRM specification also provides an extensive *Best Practice and Implementation Guide*, and an *XSL transform* that can be used to migrate metadata instances from the older versions of the IMS LRM XML binding to the IEEE LOM XML binding.

Technical details

How the data model works

The LOM comprises a **hierarchy of elements**. At the first level, there are nine categories, each of which contains sub-elements; these sub-elements may be simple elements that hold data, or may themselves be aggregate elements, which contain further sub-elements. The semantics of an element are determined by its context: they are affected by the parent or container element in the hierarchy and by other elements in the same container. For example, the various *Description* elements (1.4, 5.10, 6.3, 7.2.2, 8.3 and 9.3) each derive their context from their parent element. In addition, description element 9.3 also takes its context from the value of element 9.1 *Purpose* in the same instance of *Classification*.

The data model specifies that some elements may be repeated either individually or as a group; for example, although the elements 9.3 (*Description*) and 9.1 (*Purpose*) can only occur once within each instance of the *Classification* container element, the *Classification* element may be repeated - thus allowing many descriptions for different purposes.

The data model also specifies the **value space** and **datatype** for each of the simple data elements. The value space defines the restrictions, if any, on the data that can be entered for that element. For many elements, the value space allows any string of Unicode character to be entered, whereas other elements entries must be drawn from a declared list (i.e. a controlled vocabulary) or must be in a specified format (e.g. date and language codes). Some element datatypes simply allow a string of characters to be entered, and others comprise two parts, as described below:

- **LangString** items contain Language and String parts, allowing the same information to be recorded in multiple languages
- **Vocabulary** items are constrained in such a way that their entries have to be chosen from a controlled list of terms - composed of Source-Value pairs - with the Source containing the name of the list of terms being used and the Value containing the chosen term
- **DateTime** and **Duration** items contain one part that allows the date or duration to be given in a machine readable format, and a second that allows a description of the date or duration (for example “mid summer, 1968”).

When implementing the LOM as a data or service provider, it is not necessary to support all the elements in the data model, nor need the LOM data model limit the information which may be provided. The creation of an application profile allows a community of users to specify which elements and vocabularies they will use. Elements from the LOM may be dropped and elements from other metadata schemas may be brought in; likewise, the vocabularies in the LOM may be supplemented with values appropriate to that community.

Requirements

The key requirements for exploiting the LOM as a data or service provider are to:

- Understand user/community needs and to express these as an application profile
- Have a strategy for creating high quality metadata
- Store this metadata in a form which can be exported as LOM records
- Agree a binding for LOM instances when they are exchanged
- Be able to exchange records with other systems either as single instances or *en masse*.

Related specifications

There are many metadata specifications; of particular interest is the Dublin Core Metadata Element Set (commonly known as Simple Dublin Core, standardised as *ANSI/NISO Z39.85 – 2001*), which provides a simpler, more loosely-defined set of elements with some overlap with the LOM, and which is useful for sharing metadata across a wide range of disparate services. The Dublin Core Metadata Initiative is also working on a set of terms which allow the Dublin Core Element Set to be used with greater semantic precision (Qualified Dublin Core). The Dublin Education Working Group aims to provide refinements of Dublin Core for the specific needs of the education community. Details of Dublin Core can be found at the Dublin Core website .

Many other education-related specifications allow for LO metadata to be embedded within XML instances, such as: describing the resources in an IMS Content Package or Resource List; describing the vocabularies and terms in an IMS VDEX (Vocabulary Definition and Exchange) file; and describing the question items in an IMS QTI (Question and Test Interoperability) file. Details of these can be found at the IMS Global website .

The IMS Vocabulary Definition and Exchange (VDEX) specification has a double relation with the LOM, since not only can the LOM provide metadata on the vocabularies in a VDEX instance, but VDEX can be used to describe the controlled vocabularies which are the value space for many LOM elements.

LOM records can be transported between systems using a variety of protocols, perhaps the most widely used being OAI-PMH.

Application profiles

UK LOM Core

For UK Further and Higher Education, the most relevant family of application profiles are those based around the *UK LOM Core* . The UK LOM Core is currently a draft schema researched by a community of practitioners to identify common UK practice in learning object content, by comparing 12 metadata schemas.

CanCore

CanCore provides detailed guidance for the interpretation and implementation of each data element in the LOM standard. These guidelines constitute a 250-page document, and have been developed over three years under the leadership of Norm Friesen, and through consultation with experts across Canada and throughout the world. These guidelines are also available at no charge from the CanCore Website.

ANZ-LOM

ANZ-LOM is a metadata profile developed for the education sector in Australia and New Zealand. The profile provides interpretations of metadata structures and illustrates how to apply controlled vocabularies, especially using the "classification" element. It is supported by detailed examples of learning resource metadata, including regional vocabularies. The ANZ-LOM profile was first published by The Le@rning Federation (TLF) in January, 2008.

Vetadata

The Australian Vocational Training and Education (VET) sector uses an application profile of the IEEE LOM called Vetadata. The profile contains five mandatory elements, and makes use of a number of vocabularies specific to the Australian VET sector. This application profile was first published in 2005. The Vetadata and ANZ-LOM profiles are closely aligned.

NORLOM

NORLOM is the Norwegian LOM profile. The profile is managed by NSSL (The Norwegian Secretariat for Standardization of Learning Technologies)

ISRACore

ISRACORE is the Israeli LOM profile. The Israel Internet Association (ISOC-IL) and Inter University Computational Center (IUCC) have teamed up to manage and establish an e-learning objects database.

SWE-LOM

SWE-LOM is the Swedish LOM profile that is managed by IML at Umeå University as a part of the work with the national standardization group TK450 at Swedish Standards Institute.

Chapter-10

Knowledge-Based Engineering

Knowledge-based engineering (KBE) is a discipline with roots in computer-aided design (CAD) and knowledge-based systems but has several definitions and roles depending upon the context. An early role was support tool for a design engineer generally within the context of product design. Success of early KBE prototypes was remarkable; eventually this led to KBE being considered as the basis for generative design with many expectations for hands-off performance where there would be limited human involvement in the design process.

Overview

KBE can be defined as engineering on the basis of electronic knowledge models. Such knowledge models are the result of knowledge modeling that uses knowledge representation techniques to create the computer interpretable models. The knowledge models can be imported in and/or stored in specific engineering applications that enable engineers to specify requirements or create designs on the basis of the knowledge in such models. There are various methods available for the development of knowledge models, most of them are system dependent. An example of a system-independent language for the development machine-readable ontology databases, including support for basic engineering knowledge, is called Gellish English. An example of a CAD-specific system that can store knowledge and use it for design is the CATIA program through its KnowledgeWare module. An example of a CAD-independent, language-based KBE system with full compiler and support for runtime application deployment is General-purpose Declarative Language (GDL) from Genworks.

KBE can have a wide scope that covers the full range of activities related to Product Lifecycle Management and Multidisciplinary design optimization. KBE's scope would include design, analysis (computer-aided engineering – CAE), manufacturing, and support. In this inclusive role, KBE has to cover a large multi-disciplinary role related to many computer aided technologies (CAx).

KBE also has more general overtones. One of its roles is to bridge knowledge management and design automation. Knowledge processing is a recent advance in computing. It has played a successful role in engineering and is now undergoing modifications (to be explained). An example of KBE’s role is generative mechanical design. There are others. KBE can be thought of as an advanced form of computer

applications (in some forms with an extreme end-user computing flavor) that support PLM and CAx.

There are similar techniques, such as electronic design automation. AAAI provides a long list of engineering applications. some of which are within the KBE umbrella. At some point, the concept of KBE might split into several sub-categories as MCAD and ECAD are just two of many possible types of design automation.

History

KBE essentially was a complementary development to CAx and can be dated from the 1980s. CAx has been developing along with the computer after making large strides in the 1970s.

As with any bit of progress, KBE flashed on the horizon, lit the sky for a while, and then experienced a downslide. KBE had sufficient success stories that sustained it long enough into the 1990s to get attention. Some prime contributors to the hiatus of KBE were unmanageable expectations, increasing tedium associated with forming completion of results, and some notion that the architecture for KBE was not sufficiently based upon the newer technology.

KBE continued to exist in pockets. With the prevalence of object-oriented methods, systems advanced enough to allow re-implementation. This reconstruction has been on-going for several years and has been frustratingly slow. Now, with the basis for this discipline becoming more robust, it starts to get interesting again.

KBE, as implemented with ICAD can be thought of as an advanced form of computer applications (in some forms with an extreme end-user computing flavor) that support PLM and CAx.

KBE and product lifecycle management

The scope of PLM involves all the steps that exist within any industry that produces goods. KBE at this level will deal with product issues of a more generic nature than it will with CAx. Some might call this level 'assembly' in orientation. However, it's much more than that as PLM covers both the technical and the business side of a product.

KBE then needs to support the decision processes involved with configuration, trades, control, management, and a number of other areas, such as optimization.

Recently the Object Management Group released a RFP document and requested feedback.

KBE and CAX

CAx crosses many disciplinary bounds and provides a sound basis for PLM. In a sense, CAx is a form of applied science that uses most of the disciplines of engineering and their associated fields. Materials science comes to mind.

KBE's support of CAx may have some similarities with its support of PLM but, in a sense, the differences are going to be larger.

The KBE flavor at the CAx level may assume a strong behavioral flavor. Given the underlying object oriented focus, there is a natural use of entities possessing complicated attributes and fulfilling non-trivial roles. One vendor's approach provides a means via workbenches to embed attributes and methods within sub-parts (object) or within a joining of sub-parts into a part.

As an aggregate, the individual actions, that are event driven, can be fairly involved. This fact identifies one major problem, namely control of what is essentially a non-deterministic mixture. This characteristic of the decision problem will get more attention as the KBE systems subsume more levels and encompasses a broader scope of PLM.

KBE and knowledge management

KBE is related to knowledge management which has many levels itself. Some approaches to knowledge are reductionistic, as well they ought to be given the pragmatic focus of knowledge modeling. However, due to KBE dealing with aggregates that can be quite complicated both in structure and in behavior, some holistic notions (note link to complexity theory) might be apropos.

Also, given all the layers of KBE and given the fact that one part of an associated space is heavily mathematical (namely, manifold in nature), KBE is extremely interesting from the knowledge viewpoint (or one would hope).

All one has to do is note that the KBE process's goal is to produce results in the 'real world' via artifacts and to do so using techniques that are highly computational. That, in essence, is the epitome of applied science/engineering, and it could never be non-interesting.

KBE methodology

The development of KBE applications concerns the requirements to identify, capture, structure, formalize and finally implement knowledge. Many different so-called KBE platforms support only the implementation step which is not always the main bottleneck in the KBE development process. In order to limit the risk associated with the development and maintenance of KBE application there is a need to rely on an appropriate methodology for managing the knowledge and maintaining it up to date. As example of such KBE methodology the EU project MOKA "Methodology and tools

Oriented to Knowledge based Applications" propose solutions which focus on the structuration and formalization steps as well as links to the implementation.

An alternative to MOKA is to use a general methodology for developing knowledge bases for expert systems and for intranet pages. Such a methodology is described in "Knowledge Acquisition in Practice: A Step-by-step Guide" by Nick Milton.

Languages for KBE

Some questions can be asked in regard to KBE implementation: can we represent knowledge in a vendor-neutral format? can the knowledge in our designs be retained for decades, long after a vendor system (such as CATIA) has disappeared?

These questions are addressed in a 2005 Aerospace COE presentation A Proposal for CATIA V6 by Walter Wilson of Lockheed Martin.

Mr. Wilson advocates using a type of programming language to define design data—operations, parameters, formulas, etc. -- instead of a proprietary file format (such as Dassault's CATIA). One's data would no longer be tied to a specific CAD system. Unlike STEP, which inevitably lags commercial CAD systems in the features it supports, programmability would allow the definition of new design features.

A logic programming language is proposed as the basis for the engineering design language because of its simplicity and extensibility. The geometric engine for the language features would be open source to give engineers control over approximation algorithms and to better guarantee long-term accessibility of the data.

Meanwhile, the commercially available General-purpose Declarative Language (GDL) from Genworks International addresses the issue of application longevity by providing a high-level declarative language kernel which is a superset of a standard dialect of the Lisp programming language (currently ANSI Common Lisp, or CL).

The GDL kernel follows a concise, pragmatic language specification representing something akin to a *de-facto* neutral format for representing KBE-style knowledge. It consists of the same Smalltalk-inspired declarative object-oriented (and object-centric) message-passing format which been a common thread among classical KBE systems for more than two decades. While CL is a multi-paradigm language (supporting procedural, object-oriented, and functional programming), the core features of KBE tend to share several aspects with Functional programming languages "under the hood" (e.g. lazy evaluation, immutable data). However there is a consensus that pure Functional programming is too esoteric for typical engineers to practice, so one of the purposes of a declarative KBE language such as GDL is to provide an "engineer-friendly" object-centric front-end on what is essentially a Functional language programming environment.

Because GDL applications are written as a strict superset of a standard Lisp dialect, only the high-level declarative surface syntax is GDL-specific. The bulk of application code is

purely compliant with the underlying language standard. And because of Lisp's inherent (and unique) support for code transformation macros, even this surface syntax is subject to straightforward automated conversion among other variations of the de-facto standard. It is reasonable to expect that implementations following this approach will eventually converge on a true vendor-neutral Standard KBE language specification.

The Pacelab Suite adresses the problem that functional programming languages are still not widely accepted by potential KBE users. Engineers are usually trained in procedural and object-oriented programming languages; in quite a few software environments (Excel, MATLAB and FORTRAN) used by engineers the procedural programming style clearly dominates. Therefore the Pacelab Suite rebases the inherent technological advantages offered by a development and runtime environment like Common Lisp, on a new technological paradigm (.NET Framework) equally supporting procedural, object-oriented and functional languages.

KBE in Academia

- Design of Aircraft and Rotorcraft, Faculty of Aerospace Engineering, Delft University of Technology, The Netherlands, has adopted GDL from Genworks International as a basis for KBE and their Multi-model Generator (MMG).
- Knowledge-based engineering at the Norwegian University of Science and Technology (NTNU)
- Knowledge Based Engineering department at the Faculty of Aerospace Engineering of the Delft University of Technology

Implementations

The following KBE development packages are commercially available:

For CAD

- Adaptive Modeling Language from TechnoSoft Inc.
- DriveWorks A SolidWorks Certified Gold Partner
- GDL from Genworks International
- Kadviser from NIMTOTH previously edited by Kade-Tech
- KBEWorks by VisionKBE
- Knowledge Fusion from Siemens PLM Software
- Knowledgeware from Dassault Systemes
- Magix by Navitech
- Pro/ENGINEER Expert Framework from Parametric Technology Corporation
- SmartAssembly for Pro/ENGINEER from Sigmaxim Inc
- TactonWorks Interactive design automation inside SolidWorks
- YVE - Your Variant Engineer from tecneos software-engineering
- ICAD from Dassault Systemes (no longer available)
- KBMax Configurator by Citius Corporation

For General-purpose development of Web-deployed applications

- GDL from Genworks International

For analysis, design and engineering processes

- GDL from Genworks International
- Pacelab Suite by PACE Aerospace Engineering and Information Technology GmbH
- PCPACK by Tacit Connexions
- Quaestor by Qnowledge Modeling Technologies

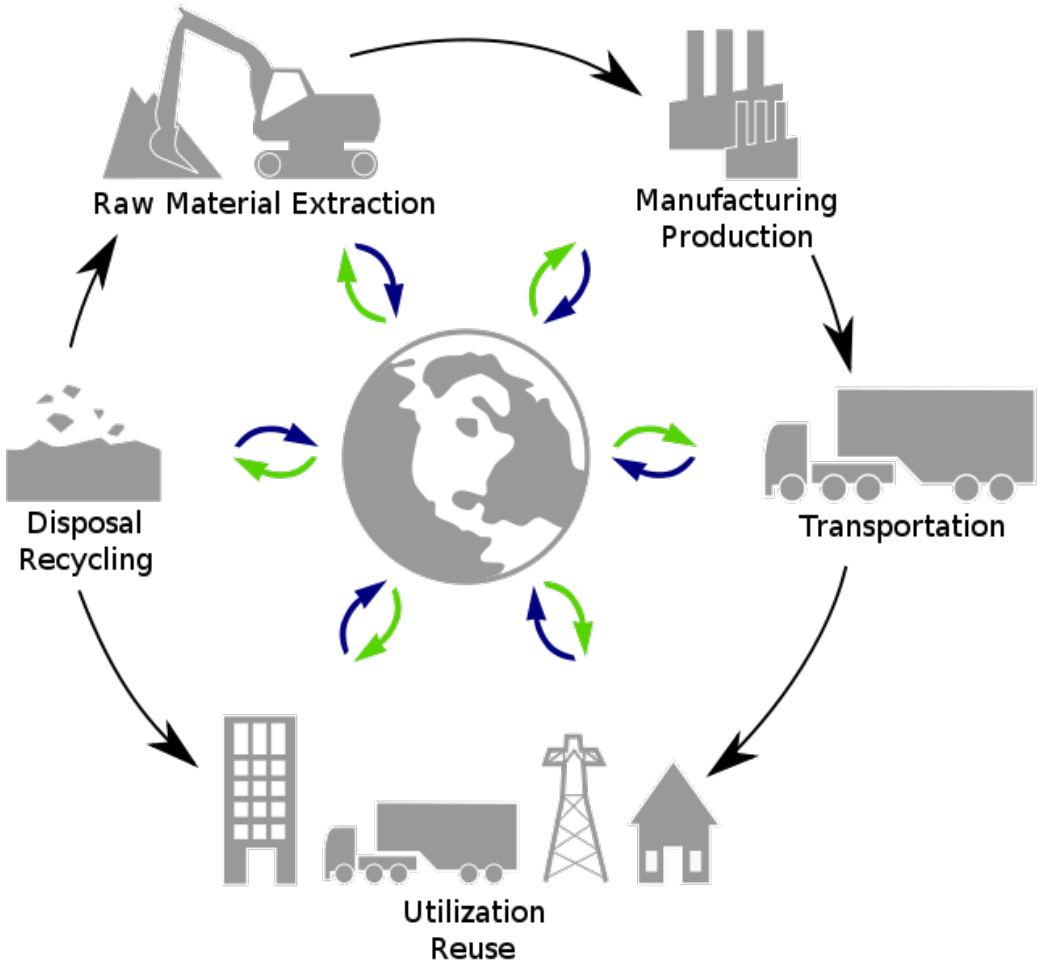
KBE futures, KBE theory

KBE, as a particular example of KS, is a multi-disciplinary framework that has more than practical considerations. Not only will KBE require successful handling of issues of the computational (Ontology, Artificial Intelligence, Entscheidungsproblem, Interactive computation, Category Theory, ...) and logic (non-monotonic issues related to the qualification, frame, and ramification problems)), it will touch upon all sciences that deal with matter, its manipulations, and the related decisions. In a sense, PLM allows us to have the world as a large laboratory for experimental co-evolution of our knowledge and our artificial co-horts. As noted in the ACM Communications, "Computers will grow to become scientists in their own right, with intuitions and computational variants of fascination and curiosity." What better framework is there to explore the "increasingly complicated mappings between the human world and the computational"?

In terms of methodology and their associated means, KBE offers support via several paradigms. These range from the home-grown all the way to strategically defined and integrated tools that cover both breadth and depth. A continuing theme will be resolving the contextual definitions for KBE into a coherent discipline (or at least attempting this) and keeping a handle on managing the necessary quantitative comparisons. One issue of importance considers what limits there may be to the computational; this study requires a multi-disciplinary focus and an understanding of the quasi-empirical. Given the knowledge focus of KBE, another issue involves what limits there might be to a computational basis for knowledge and whether these are overcome with the more advanced types of human-machine interface.

Chapter-11

Product Lifecycle Management



A generic lifecycle of products

In industry, **product lifecycle management** (**PLM**) is the process of managing the entire lifecycle of a product from its conception, through design and manufacture, to service and disposal. PLM integrates people, data, processes and business systems and provides a product information backbone for companies and their extended enterprise.

'Product lifecycle management' (PLM) should be distinguished from 'Product life cycle management (marketing)' (PLCM). PLM describes the engineering aspect of a product, from managing descriptions and properties of a product through its development and useful life; whereas, PLCM refers to the commercial management of life of a product in the business market with respect to costs and sales measures.

Product lifecycle management is one of the four cornerstones of a corporation's information technology structure. All companies need to manage communications and information with their customers (CRM-Customer Relationship Management), their suppliers (SCM-Supply Chain Management), their resources within the enterprise (ERP-Enterprise Resource Planning) and their planning (SDLC-Systems Development Life Cycle). In addition, manufacturing engineering companies must also develop, describe, manage and communicate information about their products.

A form of PLM called people-centric PLM. While traditional PLM tools have been deployed only on release or during the release phase, people-centric PLM targets the design phase.

Recent (as of 2009) ICT development (EU funded PROMISE project 2004-2008) has allowed PLM to extend beyond traditional PLM and integrate sensor data and real time 'lifecycle event data' into PLM, as well as allowing this information to be made available to different players in the total lifecycle of an individual product (closing the information loop). This has resulted in the extension of PLM into Closed Loop Lifecycle Management (CL_2M).

Benefits

Documented benefits of product lifecycle management include:

- Reduced time to market
- Improved product quality
- Reduced prototyping costs
- More accurate and timely Request For Quote generation
- Ability to quickly identify potential sales opportunities and revenue contributions
- Savings through the re-use of original data
- A framework for product optimization
- Reduced waste
- Savings through the complete integration of engineering workflows
- Documentation that can assist in proving Compliance for RoHS or Title 21 CFR Part 11
- Ability to provide Contract Manufacturers with access to a centralized product record

Areas of PLM

Within PLM there are five primary areas;

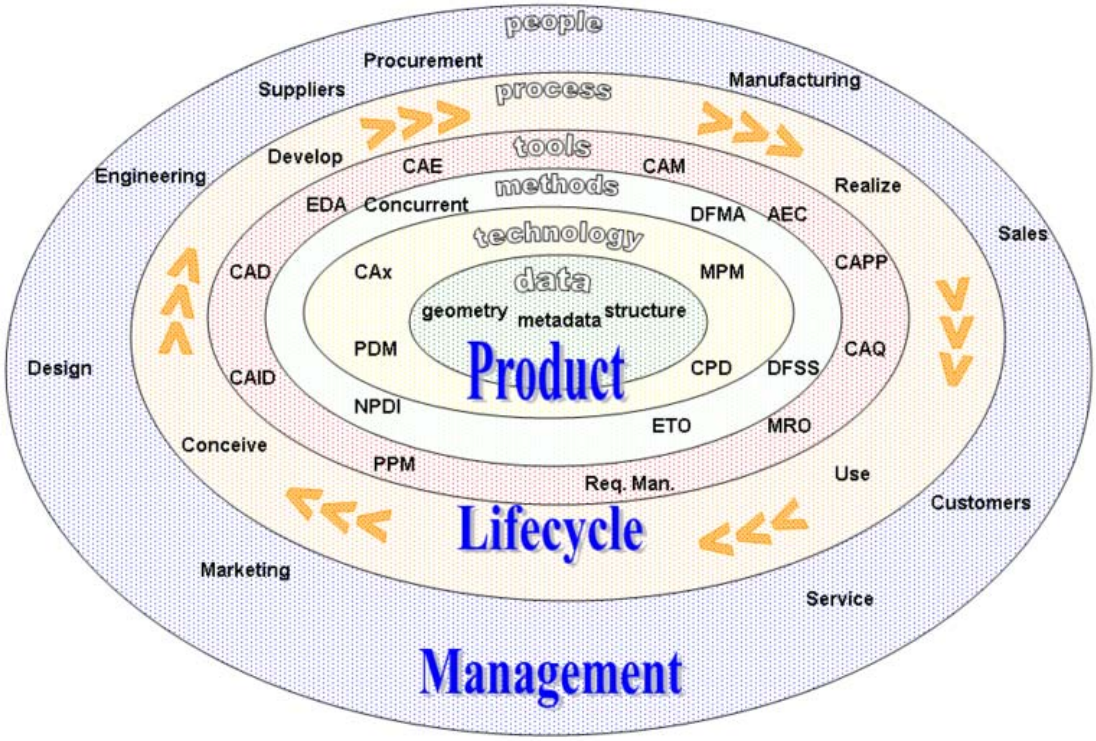
1. Systems Engineering (SE)
2. Product and Portfolio Management (PPM)
3. Product Design (CAx)
4. Manufacturing Process Management (MPM)
5. Product Data Management (PDM)

Note: While application software is not required for PLM processes, the business complexity and rate of change requires organizations execute as rapidly as possible.

Systems Engineering is focused on meeting all requirements, primary meeting customer needs, and coordinating the Systems Design process by involving all relevant disciplines. Product and Portfolio Management is focused on managing resource allocation, tracking progress vs. plan for projects in the new product development projects that are in process (or in a holding status). Portfolio management is a tool that assists management in tracking progress on new products and making trade-off decisions when allocating scarce resources. Product Data Management is focused on capturing and maintaining information on products and/or services through their development and useful life.

Introduction to development process

The core of PLM (product lifecycle management) is in the creations and central management of all product data and the technology used to access this information and knowledge. PLM as a discipline emerged from tools such as CAD, CAM and PDM, but can be viewed as the integration of these tools with methods, people and the processes through all stages of a product's life. It is not just about software technology but is also a business strategy.



For simplicity the stages described are shown in a traditional sequential engineering workflow. The exact order of event and tasks will vary according to the product and industry in question but the main processes are:

- Conceive
  - Specification
  - Concept design
- Design
  - Detailed design
  - Validation and analysis (simulation)
  - Tool design
- Realize
  - Plan manufacturing
  - Manufacture
  - Build/Assemble
  - Test (quality check)
- Service
  - Sell and Deliver
  - Use
  - Maintain and Support
  - Dispose

The major key point events are:

- Order
- Idea
- Kick-off
- Design freeze
- Launch

The reality is however more complex, people and departments cannot perform their tasks in isolation and one activity cannot simply finish and the next activity start. Design is an iterative process, often designs need to be modified due to manufacturing constraints or conflicting requirements. Where exactly a customer order fits into the time line depends on the industry type, whether the products are for example Build to Order, Engineer to Order, or Assemble to Order.

History

Inspiration for the burgeoning business process now known as PLM came when American Motors Corporation (AMC) was looking for a way to speed up its product development process to compete better against its larger competitors in 1985, according to François Castaing, Vice President for Product Engineering and Development. After introducing its compact Jeep Cherokee (XJ), the vehicle that launched the modern sport utility vehicle (SUV) market, AMC began development of a new model, that later came out as the Jeep Grand Cherokee. The first part in its quest for faster product development was computer-aided design (CAD) software system that make engineers more productive. The second part in this effort was the new communication system that allowed conflicts to be resolved faster, as well as reducing costly engineering changes because all drawings and documents were in a central database. The product data management was so effective, that after AMC was purchased by Chrysler, the system was expanded throughout the enterprise connecting everyone involved in designing and building products. While an early adopter of PLM technology, Chrysler was able to become the auto industry's lowest-cost producer, recording development costs that were half of the industry average by the mid-1990s.

Phases of product lifecycle and corresponding technologies

Many software solutions have developed to organize and integrate the different phases of a product’s lifecycle. PLM should not be seen as a single software product but a collection of software tools and working methods integrated together to address either single stages of the lifecycle or connect different tasks or manage the whole process. Some software providers cover the whole PLM range while others a single niche application. Some applications can span many fields of PLM with different modules within the same data model. An overview of the fields within PLM is covered here. It should be noted however that the simple classifications do not always fit exactly, many areas overlap and many software products cover more than one area or do not fit easily into one category. It should also not be forgotten that one of the main goals of PLM is to

collect knowledge that can be reused for other projects and to coordinate simultaneous concurrent development of many products. It is about business processes, people and methods as much as software application solutions. Although PLM is mainly associated with engineering tasks it also involves marketing activities such as Product Portfolio Management (PPM), particularly with regards to New product introduction (NPI).

Phase 1: Conceive

Imagine, specify, plan, innovate

The first stage in idea is the definition of its requirements based on customer, company, market and regulatory bodies' viewpoints. From this specification of the products major technical parameters can be defined. Parallel to the requirements specification the initial concept design work is carried out defining the visual aesthetics of the product together with its main functional aspects. For the Industrial Design, Styling, work many different media are used from pencil and paper, clay models to 3D CAID Computer-aided industrial design software.

Phase 2: Design

Describe, define, develop, test, analyze and validate

This is where the detailed design and development of the product's form starts, progressing to prototype testing, through pilot release to full product launch. It can also involve redesign and ramp for improvement to existing products as well as planned obsolescence. The main tool used for design and development is CAD Computer-aided design. This can be simple 2D Drawing / Drafting or 3D Parametric Feature Based Solid/Surface Modeling. Such software includes technology such as Hybrid Modeling, Reverse Engineering, KBE (Knowledge-Based Engineering), NDT (Nondestructive testing), Assembly construction.

This step covers many engineering disciplines including: Mechanical, Electrical, Electronic, Software (embedded), and domain-specific, such as Architectural, Aerospace, Automotive, ... Along with the actual creation of geometry there is the analysis of the components and product assemblies. Simulation, validation and optimization tasks are carried out using CAE (Computer-aided engineering) software either integrated in the CAD package or stand-alone. These are used to perform tasks such as:- Stress analysis, FEA (Finite Element Analysis); Kinematics; Computational fluid dynamics (CFD); and mechanical event simulation (MES). CAQ (Computer-aided quality) is used for tasks such as Dimensional Tolerance (engineering) Analysis. Another task performed at this stage is the sourcing of bought out components, possibly with the aid of Procurement systems.

Phase 3: Realize

Manufacture, make, build, procure, produce, sell and deliver

Once the design of the product's components is complete the method of manufacturing is defined. This includes CAD tasks such as tool design; creation of CNC Machining instructions for the product's parts as well as tools to manufacture those parts, using integrated or separate CAM Computer-aided manufacturing software. This will also involve analysis tools for process simulation for operations such as casting, molding, and die press forming. Once the manufacturing method has been identified CPM comes into play. This involves CAPE (Computer-aided Production Engineering) or CAP/CAPP – (Production Planning) tools for carrying out Factory, Plant and Facility Layout and Production Simulation. For example: Press-Line Simulation; and Industrial Ergonomics; as well as tool selection management. Once components are manufactured their geometrical form and size can be checked against the original CAD data with the use of Computer Aided Inspection equipment and software. Parallel to the engineering tasks, sales product configuration and marketing documentation work will be taking place. This could include transferring engineering data (geometry and part list data) to a web based sales configurator and other Desktop Publishing systems.

Phase 4: Service

Use, operate, maintain, support, sustain, phase-out, retire, recycle and disposal

The final phase of the lifecycle involves managing of in service information. Providing customers and service engineers with support information for repair and maintenance, as well as waste management/recycling information. This involves using such tools as Maintenance, Repair and Operations Management (MRO) software.

All phases: product lifecycle

Communicate, manage and collaborate

None of the above phases can be seen in isolation. In reality a project does not run sequentially or in isolation of other product development projects. Information is flowing between different people and systems. A major part of PLM is the co-ordination of and management of product definition data. This includes managing engineering changes and release status of components; configuration product variations; document management; planning project resources and timescale and risk assessment.

For these tasks graphical, text and metadata such as product bills of materials (BOMs) needs to be managed. At the engineering departments level this is the domain of PDM – (Product Data Management) software, at the corporate level EDM (Enterprise Data Management) software, these two definitions tend to blur however but it is typical to see two or more data management systems within an organization. These systems are also

linked to other corporate systems such as SCM, CRM, and ERP. Associated with these system are Project Management Systems for Project/Program Planning.

This central role is covered by numerous Collaborative Product Development tools which run throughout the whole lifecycle and across organizations. This requires many technology tools in the areas of Conferencing, Data Sharing and Data Translation. The field being Product visualization which includes technologies such as DMU (Digital Mock-Up), Immersive Virtual Digital Prototyping (virtual reality) and Photo realistic Imaging.

User skills

The broad array of solutions that make up the tools used within a PLM solution-set (e.g., CAD, CAM, CAx...) were initially used by dedicated practitioners who invested time and effort to gain the required skills. Designers and engineers worked wonders with CAD systems, manufacturing engineers became highly skilled CAM users while analysts, administrators and managers fully mastered their support technologies. However, achieving the full advantages of PLM requires the participation of many people of various skills from throughout an extended enterprise, each requiring the ability to access and operate on the inputs and output of other participants.

Despite the increased ease of use of PLM tools, cross-training all personnel on the entire PLM tool-set has not proven to be practical. Now, however, advances are being made to address ease of use for all participants within the PLM arena. One such advance is the availability of "role" specific user interfaces. Through Tailorable UIs, the commands that are presented to users are appropriate to their function and expertise.

Product development processes and methodologies

A number of established methodologies have been adopted by PLM and been further advanced. Together with PLM digital engineering techniques, they have been advanced to meet company goals such as reduced time to market and lower production costs. Reducing lead times is a major factor as getting a product to market quicker than the competition will help with higher revenue and profit margins and increase market share.

These techniques include:-

- Concurrent engineering workflow
- Industrial Design
- Bottom-up design
- Top-down design
- Front loading design workflow
- Design in context
- Modular design
- NPD New product development
- DFSS Design for Six Sigma

- DFMA Design for manufacture / assembly
- Digital simulation engineering
- Requirement driven design
- Specification managed validation
- Configuration Management

Concurrent engineering workflow

Concurrent engineering (British English: **simultaneous engineering**) is a workflow that, instead of working sequentially through stages, carries out a number of tasks in parallel. For example: starting tool design before the detailed designs of the product are finished, or starting on detail design solid models before the concept design surfaces models are complete. Although this does not necessarily reduce the amount of manpower required for a project, it does drastically reduce lead times and thus time to market. Feature-based CAD systems have for many years allowed the simultaneous work on 3D solid model and the 2D drawing by means of two separate files, with the drawing looking at the data in the model; when the model changes the drawing will associatively update. Some CAD packages also allow associative copying of geometry between files. This allows, for example, the copying of a part design into the files used by the tooling designer. The manufacturing engineer can then start work on tools before the final design freeze; when a design changes size or shape the tool geometry will then update. Concurrent engineering also has the added benefit of providing better and more immediate communication between departments, reducing the chance of costly, late design changes. It adopts a problem prevention method as compared to the problem solving and re-designing method of traditional sequential engineering.

Bottom-up design

Bottom-up design (CAD Centric) occurs where the definition of 3D models of a product starts with the construction of individual components. These are then virtually brought together in sub-assemblies of more than one level until the full product is digitally defined. This is sometimes known as the review structure showing what the product will look like. The BOM contains all of the physical (solid) components; it may (but not also) contain other items required for the final product BOM such as paint, glue, oil and other materials commonly described as 'bulk items'. Bulk items typically have mass and quantities but are not usually modelled with geometry.

Bottom-up design tends to focus on the capabilities of available real-world physical technology, implementing those solutions which this technology is most suited to. When these bottom-up solutions have real-world value, bottom-up design can be much more efficient than top-down design. The risk of bottom-up design is that it very efficiently provides solutions to low-value problems. The focus of Bottom-Up design is "what can we most efficiently do with this technology?" rather than the focus of Top-Down which is "What is the most valuable thing to do?"

Top-down design

Top-Down design is focused on high-level functional requirements, with relatively less focus on existing implementation technology. A top level spec is decomposed into lower and lower level structures and specifications, until the physical implementation layer is reached. The risk of a top-down design is that it will not take advantage of the most efficient applications of current physical technology, especially with respect to hardware implementation. Top-Down design sometimes results in excessive layers of lower-level abstraction and inefficient performance when the Top-Down model has followed an abstraction path which does not efficiently fit available physical-level technology. The positive value of Top-Down design is that it preserves a focus on the optimum solution requirements.

A Part-Centric Top-down design may eliminate some of the risks of Top-Down design. This starts with a layout model, often a simple 2D sketch defining basic sizes and some major defining parameters. Industrial Design, brings creative ideas to product development. Geometry from this is associatively copied down to the next level, which represents different sub-systems of the product. The geometry in the sub-systems is then used to define more detail in levels below. Depending on the complexity of the product, a number of levels of this assembly are created until the basic definition of components can be identified, such as position and principal dimensions. This information is then associatively copied to component files. In these files the components are detailed; this is where the classic bottom-up assembly starts.

The top down assembly is sometime known as a control structure. If a single file is used to define the layout and parameters for the review structure it is often known as a skeleton file.

Defense engineering traditionally develops the product structure from the top down. The system engineering process prescribes a functional decomposition of requirements and then physical allocation of product structure to the functions. This top down approach would normally have lower levels of the product structure developed from CAD data as a bottom up structure or design.

Both-Ends-Against-The-Middle design

Both-Ends-Against-The-Middle (BEATM) design is a design process that endeavors to combine the best features of Top-Down design, and Bottom-Up design into one process. A BEATM design process flow may begin with an emergent technology which suggests solutions which may have value, or it may begin with a top-down view of an important problem which needs a solution. In either case the key attribute of BEATM design methodology is to immediately focus at both ends of the design process flow: a top-down view of the solution requirements, and a bottom-up view of the available technology which may offer promise of an efficient solution. The BEATM design process proceeds

from both ends in search of an optimum merging somewhere between the top-down requirements, and bottom-up efficient implementation. In this fashion, BEATM has been shown to genuinely offer the best of both methodologies. Indeed some of the best success stories from either top-down or bottom-up have been successful because of an intuitive, yet unconscious use of the BEATM methodology. When employed consciously, BEATM offers even more powerful advantages.

Front loading design and workflow

Front loading is taking top-down design to the next stage. The complete control structure and review structure, as well as downstream data such as drawings, tooling development and CAM models, are constructed before the product has been defined or a project kick-off has been authorized. These assemblies of files constitute a template from which a family of products can be constructed. When the decision has been made to go with a new product, the parameters of the product are entered into the template model and all the associated data is updated. Obviously predefined associative models will not be able to predict all possibilities and will require additional work. The main principle is that a lot of the experimental/investigative work has already been completed. A lot of knowledge is built into these templates to be reused on new products. This does require additional resources "up front" but can drastically reduce the time between project kick-off and launch. Such methods do however require organizational changes, as considerable engineering efforts are moved into "offline" development departments. It can be seen as an analogy to creating a concept car to test new technology for future products, but in this case the work is directly used for the next product generation.

Design in context

Individual components cannot be constructed in isolation. CAD; CAiD models of components are designed within the context of part or all of the product being developed. This is achieved using assembly modelling techniques. Other components' geometry can be seen and referenced within the CAD tool being used. The other components within the sub-assembly, may or may not have been constructed in the same system, their geometry being translated from other CPD formats. Some assembly checking such as DMU is also carried out using Product visualization software.

Product and process lifecycle management (PPLM)

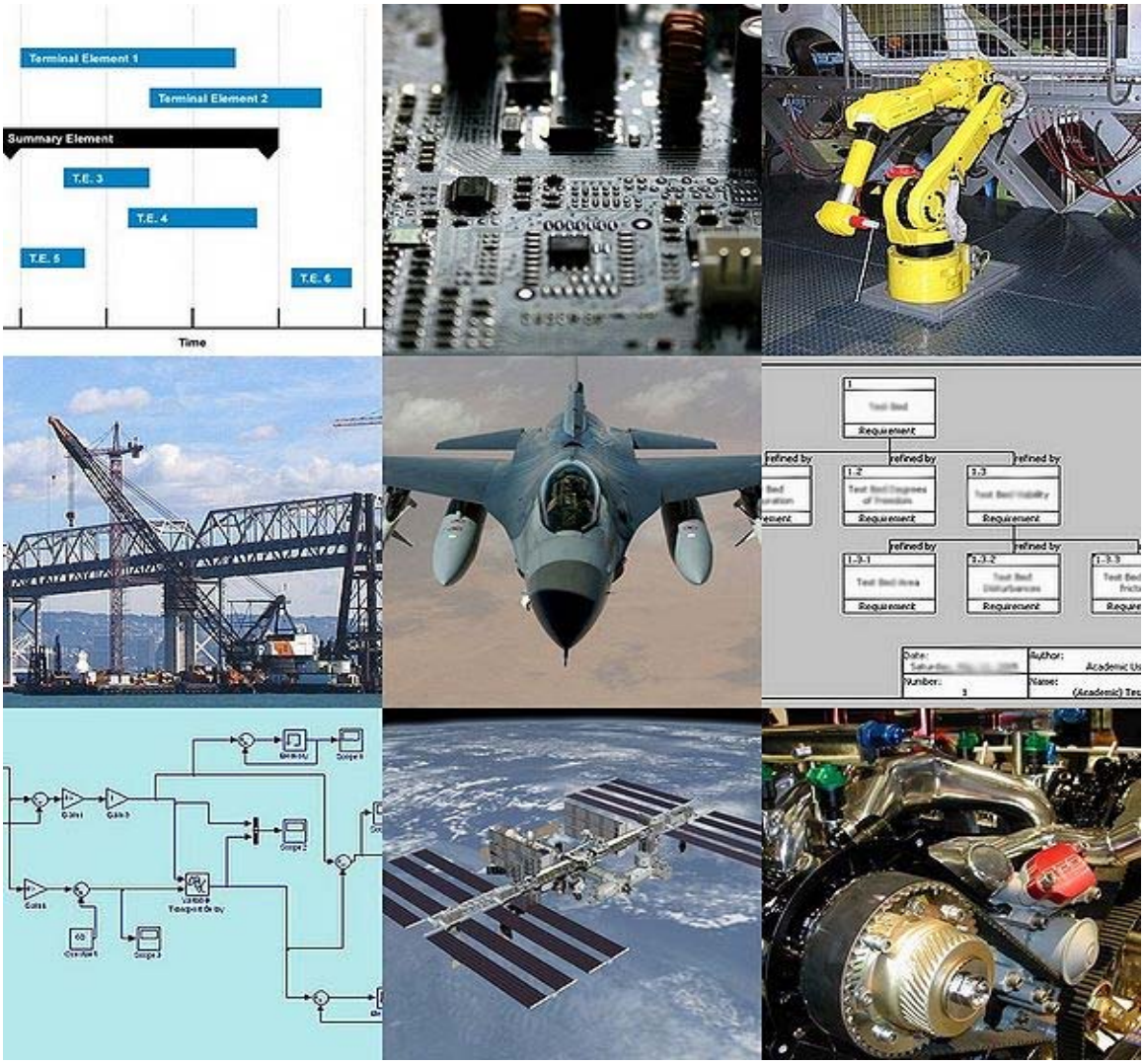
Product and process lifecycle management **(PPLM)** is an alternate genre of PLM in which the process by which the product is made is just as important as the product itself. Typically, this is the life sciences and advanced specialty chemicals markets. The process behind the manufacture of a given compound is a key element of the regulatory filing for a new drug application. As such, PPLM seeks to manage information around the development of the process in a similar fashion that baseline PLM talks about managing information around development of the product.

Market size

Total spending on PLM software and services was estimated in 2006 to be above $15 billion a year, but it is difficult to find any two market analysis reports that agree on figures. Market growth estimates are in the 10% area.

Chapter-12

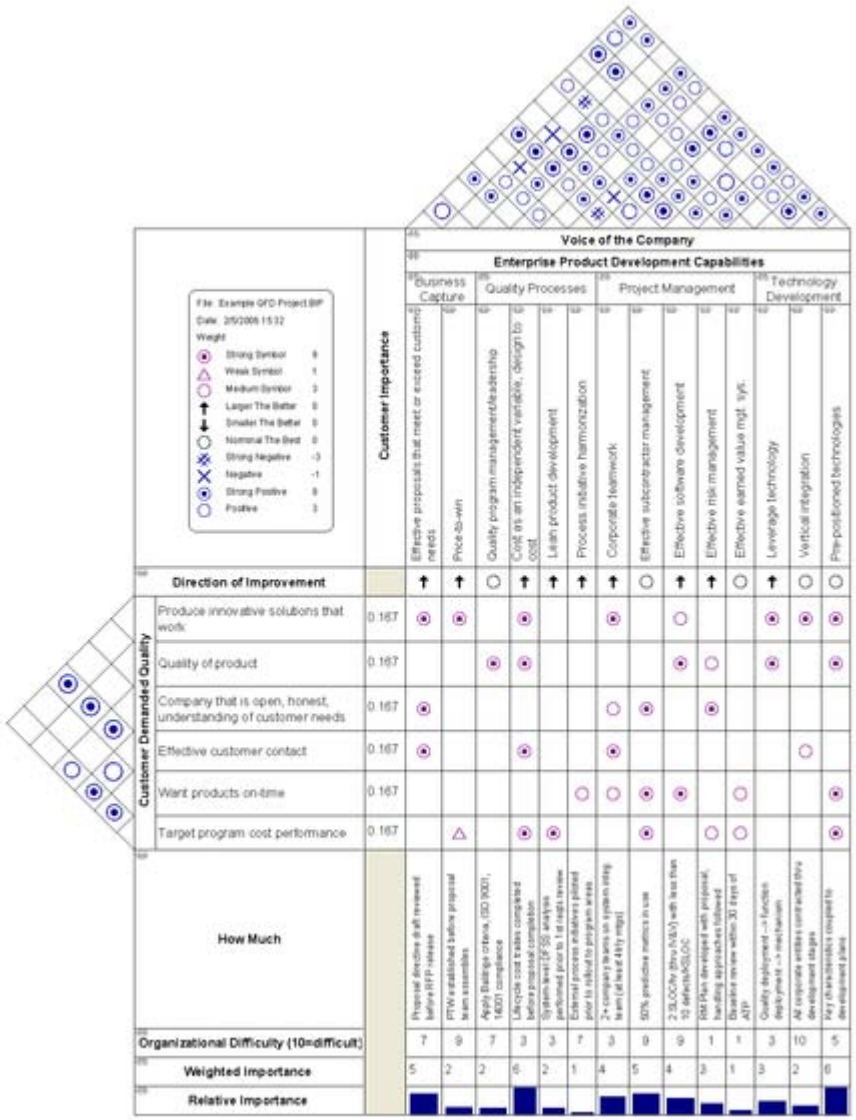
Systems Engineering



Systems engineering techniques are used in complex projects: spacecraft design, computer chip design, robotics, software integration, and bridge building. Systems engineering uses a host of tools that include modeling and simulation, requirements analysis and scheduling to manage complexity.

Systems engineering is an interdisciplinary field of engineering that focuses on how complex engineering projects should be designed and managed over the life cycle of the project.. Issues such as logistics, the coordination of different teams, and automatic control of machinery become more difficult when dealing with large, complex projects. Systems engineering deals with work-processes and tools to handle such projects, and it overlaps with both technical and human-centered disciplines such as control engineering, industrial engineering, organizational studies, and project management.

History



QFD House of Quality for Enterprise Product Development Processes

The term *systems engineering* can be traced back to Bell Telephone Laboratories in the 1940s. The need to identify and manipulate the properties of a system as a whole, which in complex engineering projects may greatly differ from the sum of the parts' properties, motivated the Department of Defense, NASA, and other industries to apply the discipline.

When it was no longer possible to rely on design evolution to improve upon a system and the existing tools were not sufficient to meet growing demands, new methods began to be developed that addressed the complexity directly. The evolution of systems engineering, which continues to this day, comprises the development and identification of new methods and modeling techniques. These methods aid in better comprehension of engineering systems as they grow more complex. Popular tools that are often used in the systems engineering context were developed during these times, including USL, UML, QFD, and IDEF0.

In 1990, a professional society for systems engineering, the *National Council on Systems Engineering* (NCOSE), was founded by representatives from a number of U.S. corporations and organizations. NCOSE was created to address the need for improvements in systems engineering practices and education. As a result of growing involvement from systems engineers outside of the U.S., the name of the organization was changed to the International Council on Systems Engineering (INCOSE) in 1995. Schools in several countries offer graduate programs in systems engineering, and continuing education options are also available for practicing engineers.

Concept

Systems engineering signifies both an approach and, more recently, as a discipline in engineering. The aim of education in systems engineering is to simply formalize the approach and in doing so, identify new methods and research opportunities similar to the way it occurs in other fields of engineering. As an approach, systems engineering is holistic and interdisciplinary in flavour.

Origins and traditional scope

The traditional scope of engineering embraces the design, development, production and operation of physical systems, and systems engineering, as originally conceived, falls within this scope. "Systems engineering", in this sense of the term, refers to the distinctive set of concepts, methodologies, organizational structures (and so on) that have been developed to meet the challenges of engineering functional physical systems of unprecedented complexity. The Apollo program is a leading example of a systems engineering project.

The use of the term " system engineer " has evolved over time to embrace a wider, more holistic concept of "systems" and of engineering processes. This evolution of the definition has been a subject of ongoing controversy , and the term continues to be applied to both the narrower and broader scope.

Holistic view

Systems engineering focuses on analyzing and eliciting customer needs and required functionality early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation while considering the complete problem, the system lifecycle. Oliver *et al.* claim that the systems engineering process can be decomposed into

- a *Systems Engineering Technical Process*, and
- a *Systems Engineering Management Process*.

Within Oliver's model, the goal of the Management Process is to organize the technical effort in the lifecycle, while the Technical Process includes *assessing available information*, *defining effectiveness measures*, to *create a behavior model*, *create a structure model*, *perform trade-off analysis*, and *create sequential build & test plan*.

Depending on their application, although there are several models that are used in the industry, all of them aim to identify the relation between the various stages mentioned above and incorporate feedback. Examples of such models include the Waterfall model and the VEE model.

Interdisciplinary field

System development often requires contribution from diverse technical disciplines. By providing a systems (holistic) view of the development effort, systems engineering helps mold all the technical contributors into a unified team effort, forming a structured development process that proceeds from concept to production to operation and, in some cases, to termination and disposal.

This perspective is often replicated in educational programs in that systems engineering courses are taught by faculty from other engineering departments which, in effect, helps create an interdisciplinary environment.

Managing complexity

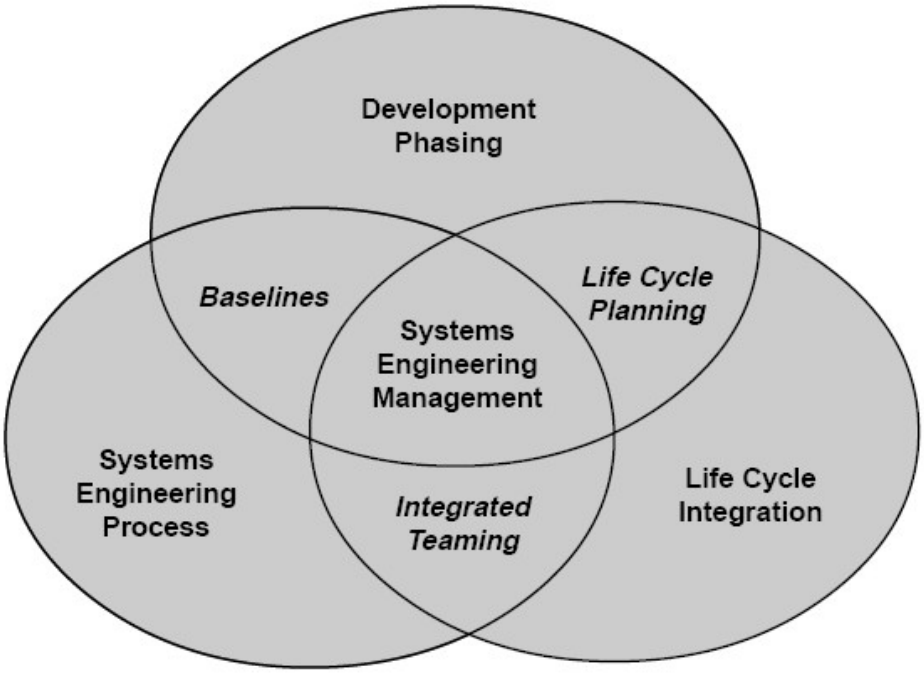
The need for systems engineering arose with the increase in complexity of systems and projects. When speaking in this context, complexity incorporates not only engineering systems, but also the logical human organization of data. At the same time, a system can become more complex due to an increase in size as well as with an increase in the amount of data, variables, or the number of fields that are involved in the design. The International Space Station is an example of such a system.

The development of smarter control algorithms, microprocessor design, and analysis of environmental systems also come within the purview of systems engineering. Systems engineering encourages the use of tools and methods to better comprehend and manage complexity in systems. Some examples of these tools can be seen here:

- *System model, Modeling, and Simulation*,
- System architecture,
- *Optimization*,
- *System dynamics*,
- *Systems analysis*,
- *Statistical analysis*,
- *Reliability analysis*, and
- *Decision making*

Taking an interdisciplinary approach to engineering systems is inherently complex since the behavior of and interaction among system components is not always immediately well defined or understood. Defining and characterizing such systems and subsystems and the interactions among them is one of the goals of systems engineering. In doing so, the gap that exists between informal requirements from users, operators, marketing organizations, and technical specifications is successfully bridged.

Scope



The scope of systems engineering activities

One way to understand the motivation behind systems engineering is to see it as a method, or practice, to identify and improve common rules that exist within a wide variety of systems. Keeping this in mind, the principles of systems engineering — holism, emergent behavior, boundary, et al. — can be applied to any system, complex or

otherwise, provided systems thinking is employed at all levels. Besides defense and aerospace, many information and technology based companies, software development firms, and industries in the field of electronics & communications require systems engineers as part of their team.

An analysis by the INCOSE Systems Engineering center of excellence (SECOE) indicates that optimal effort spent on systems engineering is about 15-20% of the total project effort. At the same time, studies have shown that systems engineering essentially leads to reduction in costs among other benefits. However, no quantitative survey at a larger scale encompassing a wide variety of industries has been conducted until recently. Such studies are underway to determine the effectiveness and quantify the benefits of systems engineering.

Systems engineering encourages the use of modeling and simulation to validate assumptions or theories on systems and the interactions within them.

Use of methods that allow early detection of possible failures, in safety engineering, are integrated into the design process. At the same time, decisions made at the beginning of a project whose consequences are not clearly understood can have enormous implications later in the life of a system, and it is the task of the modern systems engineer to explore these issues and make critical decisions. There is no method which guarantees that decisions made today will still be valid when a system goes into service years or decades after it is first conceived but there are techniques to support the process of systems engineering. Examples include the use of soft systems methodology, Jay Wright Forrester's System dynamics method and the Unified Modeling Language (UML), each of which are currently being explored, evaluated and developed to support the engineering decision making process.

Education

Education in systems engineering is often seen as an extension to the regular engineering courses, reflecting the industry attitude that engineering students need a foundational background in one of the traditional engineering disciplines (e.g. mechanical engineering, industrial engineering, computer engineering, electrical engineering) plus practical, real-world experience in order to be effective as systems engineers. Undergraduate university programs in systems engineering are rare.

INCOSE maintains a continuously updated Directory of Systems Engineering Academic Programs worldwide. As of 2006, there are about 75 institutions in United States that offer 130 undergraduate and graduate programs in systems engineering. Education in systems engineering can be taken as *SE-centric* or *Domain-centric*.

- *SE-centric* programs treat systems engineering as a separate discipline and all the courses are taught focusing on systems engineering practice and techniques.
- *Domain-centric* programs offer systems engineering as an option that can be exercised with another major field in engineering.

Both these patterns cater to educate the systems engineer who is able to oversee interdisciplinary projects with the depth required of a core-engineer.

Systems engineering topics

Systems engineering tools are strategies, procedures, and techniques that aid in performing systems engineering on a project or product. The purpose of these tools vary from database management, graphical browsing, simulation, and reasoning, to document production, neutral import/export and more.

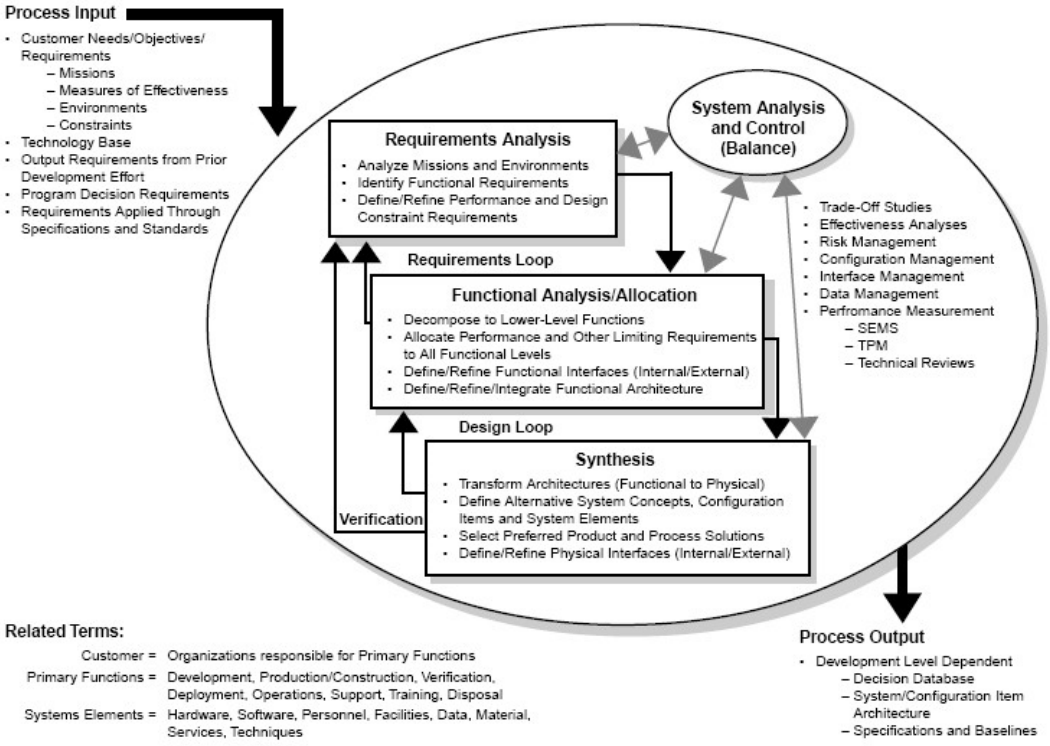
System

There are many definitions of what a system is in the field of systems engineering. Below are a few authoritative definitions:

- ANSI/EIA-632-1999: "An aggregation of end products and enabling products to achieve a given purpose."
- IEEE Std 1220-1998: "A set or arrangement of elements and processes that are related and whose behavior satisfies customer/operational needs and provides for life cycle sustainment of the products."
- ISO/IEC 15288:2008: "A combination of interacting elements organized to achieve one or more stated purposes."
- NASA Systems Engineering Handbook: "(1) The combination of elements that function together to produce the capability to meet a need. The elements include all hardware, software, equipment, facilities, personnel, processes, and procedures needed for this purpose. (2) The end product (which performs operational functions) and enabling products (which provide life-cycle support services to the operational end products) that make up a system."
- INCOSE Systems Engineering Handbook: "homogeneous entity that exhibits predefined behavior in the real world and is composed of heterogeneous parts that do not individually exhibit that behavior and an integrated configuration of components and/or subsystems."
- INCOSE: "A system is a construct or collection of different elements that together produce results not obtainable by the elements alone. The elements, or parts, can include people, hardware, software, facilities, policies, and documents; that is, all things required to produce systems-level results. The results include system level qualities, properties, characteristics, functions, behavior and performance. The value added by the system as a whole, beyond that contributed independently by the parts, is primarily created by the relationship among the parts; that is, how they are interconnected."

The systems engineering process

Depending on their application, tools are used for various stages of the systems engineering process:



Using models

Models play important and diverse roles in systems engineering. A model can be defined in several ways, including:

- An abstraction of reality designed to answer specific questions about the real world
- An imitation, analogue, or representation of a real world process or structure; or
- A conceptual, mathematical, or physical tool to assist a decision maker.

Together, these definitions are broad enough to encompass physical engineering models used in the verification of a system design, as well as schematic models like a functional flow block diagram and mathematical (i.e., quantitative) models used in the trade study process.

The main reason for using mathematical models and diagrams in trade studies is to provide estimates of system effectiveness, performance or technical attributes, and cost from a set of known or estimable quantities. Typically, a collection of separate models is needed to provide all of these outcome variables. The heart of any mathematical model is

a set of meaningful quantitative relationships among its inputs and outputs. These relationships can be as simple as adding up constituent quantities to obtain a total, or as complex as a set of differential equations describing the trajectory of a spacecraft in a gravitational field. Ideally, the relationships express causality, not just correlation.

Tools for graphic representations

Initially, when the primary purpose of a systems engineer is to comprehend a complex problem, graphic representations of a system are used to communicate a system's functional and data requirements. Common graphical representations include:

- Functional Flow Block Diagram (FFBD)
- VisSim
- Data Flow Diagram (DFD)
- N2 (N-Squared) Chart
- IDEF0 Diagram
- UML Use case diagram
- UML Sequence diagram
- USL Function Maps and Type Maps.
- Enterprise Architecture frameworks, like TOGAF, MODAF, Zachman Frameworks etc.

A graphical representation relates the various subsystems or parts of a system through functions, data, or interfaces. Any or each of the above methods are used in an industry based on its requirements. For instance, the N2 chart may be used where interfaces between systems is important. Part of the design phase is to create structural and behavioral models of the system.

Once the requirements are understood, it is now the responsibility of a systems engineer to refine them, and to determine, along with other engineers, the best technology for a job. At this point starting with a trade study, systems engineering encourages the use of weighted choices to determine the best option. A decision matrix, or Pugh method, is one way (QFD is another) to make this choice while considering all criteria that are important. The trade study in turn informs the design which again affects the graphic representations of the system (without changing the requirements). In an SE process, this stage represents the iterative step that is carried out until a feasible solution is found. A decision matrix is often populated using techniques such as statistical analysis, reliability analysis, system dynamics (feedback control), and optimization methods.

At times a systems engineer must assess the existence of feasible solutions, and rarely will customer inputs arrive at only one. Some customer requirements will produce no feasible solution. Constraints must be traded to find one or more feasible solutions. The customers' wants become the most valuable input to such a trade and cannot be assumed. Those wants/desires may only be discovered by the customer once the customer finds that he has overconstrained the problem. Most commonly, many feasible solutions can be found, and a sufficient set of constraints must be defined to produce an optimal solution.

This situation is at times advantageous because one can present an opportunity to improve the design towards one or many ends, such as cost or schedule. Various modeling methods can be used to solve the problem including constraints and a cost function.

Systems Modeling Language (SysML), a modeling language used for systems engineering applications, supports the specification, analysis, design, verification and validation of a broad range of complex systems.

Universal Systems Language (USL) is a systems oriented object modeling language with executable (computer independent) semantics for defining complex systems, including software.

Closely related fields

Many related fields may be considered tightly coupled to systems engineering. These areas have contributed to the development of systems engineering as a distinct entity.

Cognitive systems engineering
: Cognitive systems engineering (CSE) is a specific approach to the description and analysis of human-machine systems or sociotechnical systems. The three main themes of CSE are how humans cope with complexity, how work is accomplished by the use of artefacts, and how human-machine systems and socio-technical systems can be described as joint cognitive systems. CSE has since its beginning become a recognised scientific discipline, sometimes also referred to as Cognitive Engineering. The concept of a Joint Cognitive System (JCS) has in particular become widely used as a way of understanding how complex socio-technical systems can be described with varying degrees of resolution. The experience with CSE has been described in two books that summarises the field after more than 20 years of work, namely and.

Configuration Management
: Like systems engineering, Configuration Management as practiced in the defence and aerospace industry is a broad systems-level practice. The field parallels the taskings of systems engineering; where systems engineering deals with requirements development, allocation to development items and verification, Configuration Management deals with requirements capture, traceability to the development item, and audit of development item to ensure that it has achieved the desired functionality that systems engineering and/or Test and Verification Engineering have proven out through objective testing.

Control engineering
: Control engineering and its design and implementation of control systems, used extensively in nearly every industry, is a large sub-field of systems engineering. The cruise control on an automobile and the guidance system for a ballistic missile are two examples. Control systems theory is an active field of applied mathematics involving the investigation of solution spaces and the development of new methods for the analysis of the control process.

Industrial engineering

Industrial engineering is a branch of engineering that concerns the development, improvement, implementation and evaluation of integrated systems of people, money, knowledge, information, equipment, energy, material and process. Industrial engineering draws upon the principles and methods of engineering analysis and synthesis, as well as mathematical, physical and social sciences together with the principles and methods of engineering analysis and design to specify, predict and evaluate the results to be obtained from such systems.

Interface design

Interface design and its specification are concerned with assuring that the pieces of a system connect and inter-operate with other parts of the system and with external systems as necessary. Interface design also includes assuring that system interfaces be able to accept new features, including mechanical, electrical and logical interfaces, including reserved wires, plug-space, command codes and bits in communication protocols. This is known as extensibility. Human-Computer Interaction (HCI) or Human-Machine Interface (HMI) is another aspect of interface design, and is a critical aspect of modern systems engineering. Systems engineering principles are applied in the design of network protocols for local-area networks and wide-area networks.

Mechatronic engineering

Mechatronic engineering, like Systems engineering, is a multidisciplinary field of engineering that uses dynamical systems modeling to express tangible constructs. In that regards it is almost indistinguishable from Systems Engineering, but what sets it apart is the focus on smaller details rather than larger generalizations and relationships. As such, both fields are distinguished by the scope of their projects rather than the methodology of their practice.

Operations research

Operations research supports systems engineering. The tools of operations research are used in systems analysis, decision making, and trade studies. Several schools teach SE courses within the operations research or industrial engineering department, highlighting the role systems engineering plays in complex projects. Operations research, briefly, is concerned with the optimization of a process under multiple constraints.

Performance engineering

Performance engineering is the discipline of ensuring a system will meet the customer's expectations for performance throughout its life. Performance is usually defined as the speed with which a certain operation is executed or the capability of executing a number of such operations in a unit of time. Performance may be degraded when an operations queue to be executed is throttled when the capacity is of the system is limited. For example, the performance of a packet-switched network would be characterised by the end-to-end packet transit delay or the number of packets switched within an hour. The design of high-performance systems makes use of analytical or simulation modeling, whereas the delivery of high-performance implementation involves thorough performance testing.

Performance engineering relies heavily on statistics, queueing theory and probability theory for its tools and processes.

Program management and project management.

Program management (or programme management) has many similarities with systems engineering, but has broader-based origins than the engineering ones of systems engineering. Project management is also closely related to both program management and systems engineering.

Proposal engineering

Proposal engineering is the application of scientific and mathematical principles to design, construct, and operate a cost-effective proposal development system. Basically, proposal engineering uses the "systems engineering process" to create a cost effective proposal and increase the odds of a successful proposal.

Reliability engineering

Reliability engineering is the discipline of ensuring a system will meet the customer's expectations for reliability throughout its life; i.e. it will not fail more frequently than expected. Reliability engineering applies to all aspects of the system. It is closely associated with maintainability, availability and logistics engineering. Reliability engineering is always a critical component of safety engineering, as in failure modes and effects analysis (FMEA) and hazard fault tree analysis, and of security engineering. Reliability engineering relies heavily onstatistics, probability theory and reliability theory for its tools and processes.

Safety engineering

The techniques of safety engineering may be applied by non-specialist engineers in designing complex systems to minimize the probability of safety-critical failures. The "System Safety Engineering" function helps to identify "safety hazards" in emerging designs, and may assist with techniques to "mitigate" the effects of (potentially) hazardous conditions that cannot be designed out of systems.

Security engineering

Security engineering can be viewed as an interdisciplinary field that integrates the community of practice for control systems design, reliability, safety and systems engineering. It may involve such sub-specialties as authentication of system users, system targets and others: people, objects and processes.

Software engineering

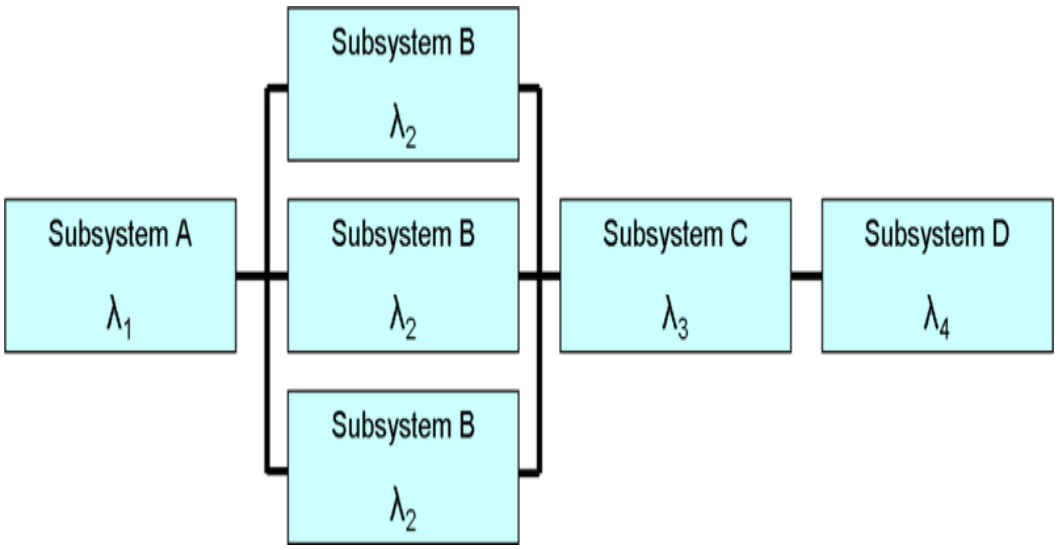
From its beginnings, software engineering has helped shape modern systems engineering practice. The techniques used in the handling of complexes of large software-intensive systems has had a major effect on the shaping and reshaping of the tools, methods and processes of SE.

Chapter-13

Reliability Engineering

Reliability engineering is an engineering field, that deals with the study of reliability: the ability of a system or component to perform its required functions under stated conditions for a specified period of time. It is often reported as a probability.

Overview



A Reliability Block Diagram

Reliability may be defined in several ways:

- The idea that something is fit for a purpose with respect to time;
- The capacity of a device or system to perform as designed;
- The resistance to failure of a device or system;
- The ability of a device or system to perform a required function under stated conditions for a specified period of time;
- The probability that a functional unit will perform its required function for a specified interval under stated conditions.
- The ability of something to "fail well" (fail without catastrophic consequences)

Reliability engineers rely heavily on statistics, probability theory, and reliability theory. Many engineering techniques are used in reliability engineering, such as reliability

prediction, Weibull analysis, thermal management, reliability testing and accelerated life testing. Because of the large number of reliability techniques, their expense, and the varying degrees of reliability required for different situations, most projects develop a reliability program plan to specify the reliability tasks that will be performed for that specific system.

The function of reliability engineering is to develop the reliability requirements for the product, establish an adequate reliability program, and perform appropriate analyses and tasks to ensure the product will meet its requirements. These tasks are managed by a reliability engineer, who usually holds an accredited engineering degree and has additional reliability-specific education and training. Reliability engineering is closely associated with maintainability engineering and logistics engineering. Many problems from other fields, such as security engineering, can also be approached using reliability engineering techniques. Here we, provides an overview of some of the most common reliability engineering tasks.

Many types of engineering employ reliability engineers and use the tools and methodology of reliability engineering. For example:

- System engineers design complex systems having a specified reliability
- Mechanical engineers may have to design a machine or system with a specified reliability
- Automotive engineers have reliability requirements for the automobiles (and components) which they design
- Electronics engineers must design and test their products for reliability requirements.
- In software engineering and systems engineering the **reliability engineering** is the subdiscipline of ensuring that a system (or a device in general) will perform its intended function(s) when operated in a specified manner for a specified length of time. Reliability engineering is performed throughout the entire life cycle of a system, including development, test, production and operation.

Reliability theory

Reliability theory is the foundation of reliability engineering. For engineering purposes, reliability is defined as:

> **the probability that a device will perform its intended function during a specified period of time under stated conditions.**

Mathematically, this may be expressed as,

$$R(t) = Pr\{T > t\} = \int_t^\infty f(x)\,dx$$

,

where $f(x)$ is the failure probability density function and *t* is the length of the period of time (which is assumed to start from time zero).

Reliability engineering is concerned with four key elements of this definition:

- First, reliability is a probability. This means that failure is regarded as a random phenomenon: it is a recurring event, and we do not express any information on individual failures, the causes of failures, or relationships between failures, except that the likelihood for failures to occur varies over time according to the given probability function. Reliability engineering is concerned with meeting the specified probability of success, at a specified statistical confidence level.
- Second, reliability is predicated on "intended function:" Generally, this is taken to mean operation without failure. However, even if no individual part of the system fails, but the system as a whole does not do what was intended, then it is still charged against the system reliability. The system requirements specification is the criterion against which reliability is measured.
- Third, reliability applies to a specified period of time. In practical terms, this means that a system has a specified chance that it will operate without failure before time t. Reliability engineering ensures that components and materials will meet the requirements during the specified time. Units other than time may sometimes be used. The automotive industry might specify reliability in terms of miles, the military might specify reliability of a gun for a certain number of rounds fired. A piece of mechanical equipment may have a reliability rating value in terms of cycles of use.
- Fourth, reliability is restricted to operation under stated conditions. This constraint is necessary because it is impossible to design a system for unlimited conditions. A Mars Rover will have different specified conditions than the family car. The operating environment must be addressed during design and testing. Also, that same rover, may be required to operate in varying conditions requiring additional scrutiny.

Reliability program plan

Many tasks, methods, and tools can be used to achieve reliability. Every system requires a different level of reliability. A commercial airliner must operate under a wide range of conditions. The consequences of failure are grave, but there is a correspondingly higher budget. A pencil sharpener may be more reliable than an airliner, but has a much different set of operational conditions, insignificant consequences of failure, and a much lower budget.

A reliability program plan is used to document exactly what tasks, methods, tools, analyses, and tests are required for a particular system. For complex systems, the reliability program plan is a separate document. For simple systems, it may be combined with the systems engineering management plan or integrated Logistics Support management plan. The reliability program plan is essential for a successful reliability program and is developed early during system development. It specifies not only what the

reliability engineer does, but also the tasks performed by others. The reliability program plan is approved by top program management.

Reliability requirements

For any system, one of the first tasks of reliability engineering is to adequately specify the reliability requirements. Reliability requirements address the system itself, test and assessment requirements, and associated tasks and documentation. Reliability requirements are included in the appropriate system/subsystem requirements specifications, test plans, and contract statements.

System reliability parameters

Requirements are specified using reliability parameters. The most common reliability parameter is the mean time between failures (MTBF), which can also be specified as the failure rate or the number of failures during a given period. These parameters are very useful for systems that are operated frequently, such as most vehicles, machinery, and electronic equipment. Reliability increases as the MTBF increases. The MTBF is usually specified in hours, but can also be used with other units of measurement such as miles or cycles.

In other cases, reliability is specified as the probability of mission success. For example, reliability of a scheduled aircraft flight can be specified as a dimensionless probability or a percentage. refer to system safety engineering.

A special case of mission success is the single-shot device or system. These are devices or systems that remain relatively dormant and only operate once. Examples include automobile airbags, thermal batteries and missiles. Single-shot reliability is specified as a probability of success, or is subsumed into a related parameter. Single-shot missile reliability may be incorporated into a requirement for the probability of hit.

For such systems, the probability of failure on demand (PFD) is the reliability measure. This PFD is derived from failure rate and mission time for non-repairable systems. For repairable systems, it is obtained from failure rate and mean-time-to-repair (MTTR) and test interval. This measure may not be unique for a given system as this measure depends on the kind of demand. In addition to system level requirements, reliability requirements may be specified for critical subsystems. In all cases, reliability parameters are specified with appropriate statistical confidence intervals.

Reliability modelling

Reliability modelling is the process of predicting or understanding the reliability of a component or system. Two separate fields of investigation are common: The physics of failure approach uses an understanding of the failure mechanisms involved, such as crack propagation or chemical corrosion; The parts stress modelling approach is an empirical

method for prediction based on counting the number and type of components of the system, and the stress they undergo during operation.

For systems with a clearly defined failure time (which is sometimes not given for systems with a drifting parameter), the empirical distribution function of these failure times can be determined. This is done in general in an accelerated experiment with increased stress. These experiments can be divided into two main categories:

Early failure rate studies determine the distribution with a decreasing failure rate over the first part of the bathtub curve. Here in general only moderate stress is necessary. The stress is applied for a limited period of time in what is called a censored test. Therefore, only the part of the distribution with early failures can be determined.

In so-called zero defect experiments, only limited information about the failure distribution is acquired. Here the stress, stress time, or the sample size is so low that not a single failure occurs. Due to the insufficient sample size, only an upper limit of the early failure rate can be determined. At any rate, it looks good for the customer if there are no failures.

In a study of the intrinsic failure distribution, which is often a material property, higher stresses are necessary to get failure in a reasonable period of time. Several degrees of stress have to be applied to determine an acceleration model. The empirical failure distribution is often parametrised with a Weibull or a log-normal model.

It is a general praxis to model the early failure rate with an exponential distribution. This less complex model for the failure distribution has only one parameter: the constant failure rate. In such cases, the Chi-square distribution can be used to find the goodness of fit for the estimated failure rate. Compared to a model with a decreasing failure rate, this is quite pessimistic. Combined with a zero-defect experiment this becomes even more pessimistic. The effort is greatly reduced in this case: one does not have to determine a second model parameter (e.g., the shape parameter of a Weibull distribution) or its confidence interval (e.g., by an MLE / Maximum likelihood approach) - and the sample size is much smaller.

Reliability test requirements

Because reliability is a probability, even highly reliable systems have some chance of failure. However, testing reliability requirements is problematic for several reasons. A single test is insufficient to generate enough statistical data. Multiple tests or long-duration tests are usually very expensive. Some tests are simply impractical. Reliability engineering is used to design a realistic and affordable test program that provides enough evidence that the system meets its requirement. Statistical confidence levels are used to address some of these concerns. A certain parameter is expressed along with a corresponding confidence level: for example, an MTBF of 1000 hours at 90% confidence level. From this specification, the reliability engineer can design a test with explicit

criteria for the number of hours and number of failures until the requirement is met or failed.

The combination of reliability parameter value and confidence level greatly affects the development cost and the risk to both the customer and producer. Care is needed to select the best combination of requirements. Reliability testing may be performed at various levels, such as component, subsystem, and system. Also, many factors must be addressed during testing, such as extreme temperature and humidity, shock, vibration, and heat. Reliability engineering determines an effective test strategy so that all parts are exercised in relevant environments. For systems that must last many years, reliability engineering may be used to design an accelerated life test as well.

Reliability prediction

A prediction of reliability is an important element in the process of selecting equipment for use by telecommunications service providers and other buyers of electronic equipment. Reliability is a measure of the frequency of equipment failures as a function of time. Reliability has a major impact on maintenance and repair costs and on the continuity of service. Reliability predictions:

- Help assess the effect of product reliability on the maintenance activity and on the quantity of spare units required for acceptable field performance of any particular system. For example, predictions of the frequency of unit level maintenance actions can be obtained. Reliability prediction can be used to size spare populations.
- Provide necessary input to system-level reliability models. System-level reliability models can subsequently be used to predict, for example, frequency of system outages in steady-state, frequency of system outages during early life, expected downtime per year, and system availability.
- Provide necessary input to unit and system-level Life Cycle Cost Analyses. Life cycle cost studies determine the cost of a product over its entire life. Therefore, how often a unit will have to be replaced needs to be known. Inputs to this process include unit and system failure rates. This includes how often units and systems fail during the first year of operation as well as in later years.
- Assist in deciding which product to purchase from a list of competing products. As a result, it is essential that reliability predictions be based on a common procedure.
- Can be used to set factory test standards for products requiring a reliability test. Reliability predictions help determine how often the system should fail.
- Are needed as input to the analysis of complex systems such as switching systems and digital cross-connect systems. It is necessary to know how often different parts of the system are going to fail even for redundant components.
- Can be used in design trade-off studies. For example, a supplier could look at a design with many simple devices and compare it to a design with fewer devices that are newer but more complex. The unit with fewer devices is usually more reliable.

- Can be used to set achievable in-service performance standards against which to judge actual performance and stimulate action.

The telecommunications industry has devoted much time over the years to concentrate on developing reliability models for electronic equipment. One such tool is the Automated Reliability Prediction Procedure (ARPP), which is an Excel-spreadsheet software tool that automates the reliability prediction procedures in SR-332, Reliability Prediction Procedure for Electronic Equipment. FD-ARPP-01 provides suppliers and manufacturers with a tool for making Reliability Prediction Procedure (RPP) calculations. It also provides a means for understanding RPP calculations through the capability of interactive examples provided by the user.

The RPP views electronic systems as hierarchical assemblies. Systems are constructed from units that, in turn, are constructed from devices. The methods presented predict reliability at these three hierarchical levels:

1. *Device*: A basic component (or part)
2. *Unit*: Any assembly of devices. This may include, but is not limited to, circuit packs, modules, plug-in units, racks, power supplies, and ancillary equipment. Unless otherwise dictated by maintenance considerations, a unit will usually be the lowest level of replaceable assemblies/devices. The RPP is aimed primarily at reliability prediction of units.
3. *Serial System*: Any assembly of units for which the failure of any single unit will cause a failure of the system.

Requirements for reliability tasks

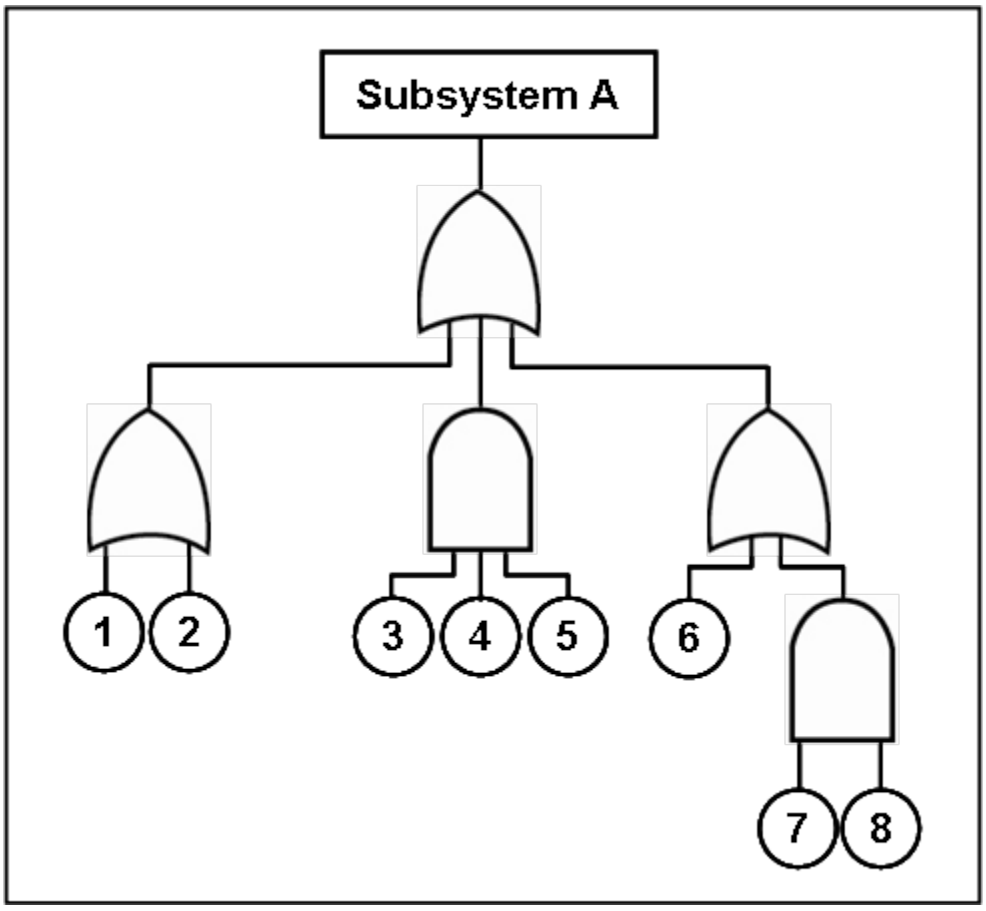
Reliability engineering must also address requirements for various reliability tasks and documentation during system development, test, production, and operation. These requirements are generally specified in the contract statement of work and depend on how much leeway the customer wishes to provide to the contractor. Reliability tasks include various analyses, planning, and failure reporting. Task selection depends on the criticality of the system as well as cost. A critical system may require a formal failure reporting and review process throughout development, whereas a non-critical system may rely on final test reports. The most common reliability program tasks are documented in reliability program standards, such as MIL-STD-785 and IEEE 1332. Failure reporting analysis and corrective action systems are a common approach for product/process reliability monitoring.

Design for reliability

Design For Reliability (DFR), is an emerging discipline that refers to the process of designing reliability into products. This process encompasses several tools and practices and describes the order of their deployment that an organization needs to have in place to drive reliability into their products. Typically, the first step in the DFR process is to set the system’s reliability requirements. Reliability must be "designed in" to the system.

During system design, the top-level reliability requirements are then allocated to subsystems by design engineers and reliability engineers working together.

Reliability design begins with the development of a model. Reliability models use **block diagrams** and **fault trees** to provide a graphical means of evaluating the relationships between different parts of the system. These models incorporate predictions based on parts-count failure rates taken from historical data. While the predictions are often not accurate in an absolute sense, they are valuable to assess relative differences in design alternatives.



A Fault Tree Diagram

One of the most important design techniques is **redundancy**. This means that if one part of the system fails, there is an alternate success path, such as a backup system. An automobile brake light might use two light bulbs. If one bulb fails, the brake light still operates using the other bulb. Redundancy significantly increases system reliability, and is often the only viable means of doing so. However, redundancy is difficult and expensive, and is therefore limited to critical parts of the system. Another design technique, **physics of failure**, relies on understanding the physical processes of stress, strength and failure at a very detailed level. Then the material or component can be re-designed to reduce the probability of failure. Another common design technique is

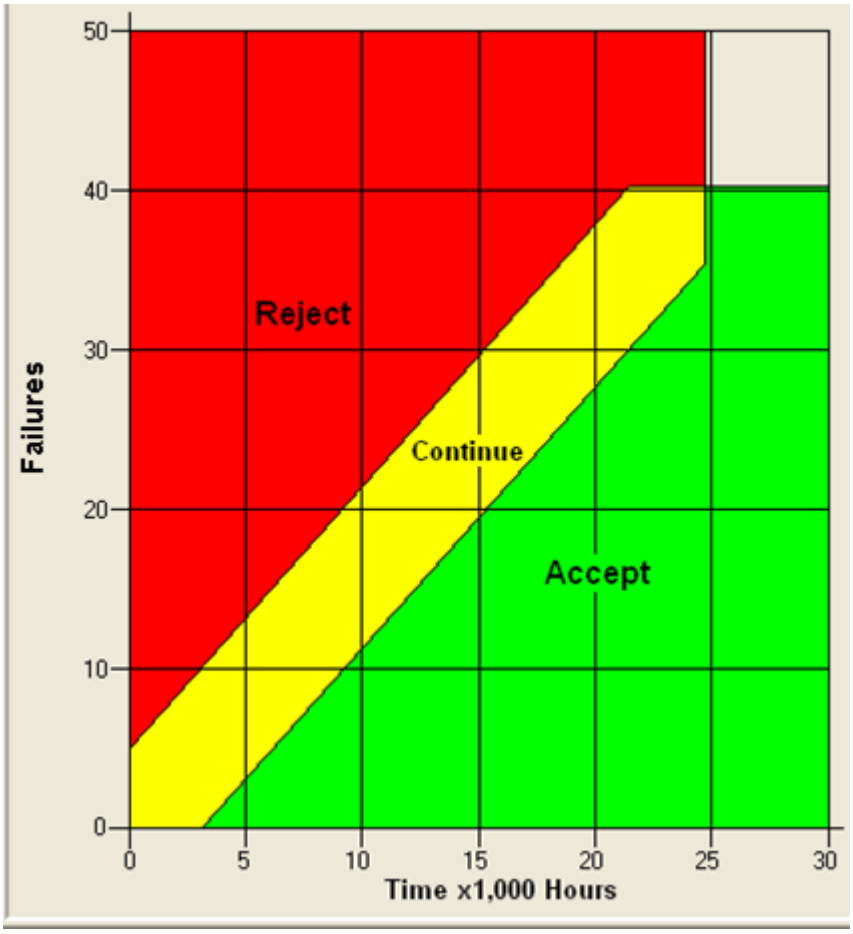
component derating: Selecting components whose tolerance significantly exceeds the expected stress, as using a heavier gauge wire that exceeds the normal specification for the expected electrical current.

Many tasks, techniques and analyses are specific to particular industries and applications. Commonly these include:

- Built-in test (BIT)
- Failure mode and effects analysis (FMEA)
- Reliability simulation modeling
- Thermal analysis
- Reliability Block Diagram analysis
- Fault tree analysis
- Root cause analysis
- Sneak circuit analysis
- Accelerated Testing
- Reliability Growth analysis
- Weibull analysis
- Electromagnetic analysis
- Statistical interference
- Avoid Single Point of Failure

Results are presented during the system design reviews and logistics reviews. Reliability is just one requirement among many system requirements. Engineering trade studies are used to determine the optimum balance between reliability and other requirements and constraints.

Reliability testing



A Reliability Sequential Test Plan

The purpose of reliability testing is to discover potential problems with the design as early as possible and, ultimately, provide confidence that the system meets its reliability requirements.

Reliability testing may be performed at several levels. Complex systems may be tested at component, circuit board, unit, assembly, subsystem and system levels. (The test level nomenclature varies among applications.) For example, performing environmental stress screening tests at lower levels, such as piece parts or small assemblies, catches problems before they cause failures at higher levels. Testing proceeds during each level of integration through full-up system testing, developmental testing, and operational testing, thereby reducing program risk. System reliability is calculated at each test level. Reliability growth techniques and failure reporting, analysis and corrective active systems (FRACAS) are often employed to improve reliability as testing progresses. The drawbacks to such extensive testing are time and expense. Customers may choose to accept more risk by eliminating some or all lower levels of testing.

It is not always feasible to test all system requirements. Some systems are prohibitively expensive to test; some failure modes may take years to observe; some complex interactions result in a huge number of possible test cases; and some tests require the use of limited test ranges or other resources. In such cases, different approaches to testing can be used, such as accelerated life testing, design of experiments, and simulations.

The desired level of statistical confidence also plays an important role in reliability testing. Statistical confidence is increased by increasing either the test time or the number of items tested. Reliability test plans are designed to achieve the specified reliability at the specified confidence level with the minimum number of test units and test time. Different test plans result in different levels of risk to the producer and consumer. The desired reliability, statistical confidence, and risk levels for each side influence the ultimate test plan. Good test requirements ensure that the customer and developer agree in advance on how reliability requirements will be tested.

A key aspect of reliability testing is to define "failure". Although this may seem obvious, there are many situations where it is not clear whether a failure is really the fault of the system. Variations in test conditions, operator differences, weather, and unexpected situations create differences between the customer and the system developer. One strategy to address this issue is to use a **scoring conference** process. A scoring conference includes representatives from the customer, the developer, the test organization, the reliability organization, and sometimes independent observers. The scoring conference process is defined in the statement of work. Each test case is considered by the group and "scored" as a success or failure. This scoring is the official result used by the reliability engineer.

As part of the requirements phase, the reliability engineer develops a test strategy with the customer. The test strategy makes trade-offs between the needs of the reliability organization, which wants as much data as possible, and constraints such as cost, schedule, and available resources. Test plans and procedures are developed for each reliability test, and results are documented in official reports.

Accelerated testing

The purpose of accelerated life testing is to induce field failure in the laboratory at a much faster rate by providing a harsher, but nonetheless representative, environment. In such a test the product is expected to fail in the lab just as it would have failed in the field—but in much less time. The main objective of an accelerated test is either of the following:

- To discover failure modes
- To predict the normal field life from the high stress lab life

An **Accelerated testing** program can be broken down into the following steps:

- Define objective and scope of the test

- Collect required information about the product
- Identify the stress(es)
- Determine level of stress(es)
- Conduct the Accelerated test and analyse the accelerated data.

Common way to determine a life stress relationship are

- Arrhenius Model
- Eyring Model
- Inverse Power Law Model
- Temperature-Humidity Model
- Temperature Non-thermal Model

Software reliability

Software reliability is a special aspect of reliability engineering. System reliability, by definition, includes all parts of the system, including hardware, software, operators and procedures. Traditionally, reliability engineering focuses on critical hardware parts of the system. Since the widespread use of digital integrated circuit technology, software has become an increasingly critical part of most electronics and, hence, nearly all present day systems. There are significant differences, however, in how software and hardware behave. Most hardware unreliability is the result of a component or material failure that results in the system not performing its intended function. Repairing or replacing the hardware component restores the system to its original unfailed state. However, software does not fail in the same sense that hardware fails. Instead, software unreliability is the result of unanticipated results of software operations. Even relatively small software programs can have astronomically large combinations of inputs and states that are infeasible to exhaustively test. Restoring software to its original state only works until the same combination of inputs and states results in the same unintended result. Software reliability engineering must take this into account.

Despite this difference in the source of failure between software and hardware — software doesn't wear out — some in the software reliability engineering community believe statistical models used in hardware reliability are nevertheless useful as a measure of software reliability, describing what we experience with software: the longer you run software, the higher the probability you'll eventually use it in an untested manner and find a latent defect that results in a failure (Shooman 1987), (Musa 2005), (Denney 2005).

As with hardware, software reliability depends on good requirements, design and implementation. Software reliability engineering relies heavily on a disciplined software engineering process to anticipate and design against unintended consequences. There is more overlap between software quality engineering and software reliability engineering than between hardware quality and reliability. A good software development plan is a key aspect of the software reliability program. The software development plan describes the

design and coding standards, peer reviews, unit tests, configuration management, software metrics and software models to be used during software development.

A common reliability metric is the number of software faults, usually expressed as faults per thousand lines of code. This metric, along with software execution time, is key to most software reliability models and estimates. The theory is that the software reliability increases as the number of faults (or fault density) goes down. Establishing a direct connection between fault density and mean-time-between-failure is difficult, however, because of the way software faults are distributed in the code, their severity, and the probability of the combination of inputs necessary to encounter the fault. Nevertheless, fault density serves as a useful indicator for the reliability engineer. Other software metrics, such as complexity, are also used.

Testing is even more important for software than hardware. Even the best software development process results in some software faults that are nearly undetectable until tested. As with hardware, software is tested at several levels, starting with individual units, through integration and full-up system testing. Unlike hardware, it is inadvisable to skip levels of software testing. During all phases of testing, software faults are discovered, corrected, and re-tested. Reliability estimates are updated based on the fault density and other metrics. At system level, mean-time-between-failure data are collected and used to estimate reliability. Unlike hardware, performing exactly the same test on exactly the same software configuration does not provide increased statistical confidence. Instead, software reliability uses different metrics such as test coverage.

Eventually, the software is integrated with the hardware in the top-level system, and software reliability is subsumed by system reliability. The Software Engineering Institute's Capability Maturity Model is a common means of assessing the overall software development process for reliability and quality purposes. However, actual software reliability is served through SAE standards JA1002 and JA1003.

Reliability Operational Assessment

After a system is produced, reliability engineering monitors, assesses, and corrects deficiencies. Monitoring includes electronic and visual surveillance of critical parameters identified during the fault tree analysis design stage. The data are constantly analyzed using statistical techniques, such as Weibull analysis and linear regression, to ensure the system reliability meets requirements. Reliability data and estimates are also key inputs for system logistics. Data collection is highly dependent on the nature of the system. Most large organizations have quality control groups that collect failure data on vehicles, equipment, and machinery. Consumer product failures are often tracked by the number of returns. For systems in dormant storage or on standby, it is necessary to establish a formal surveillance program to inspect and test random samples. Any changes to the system, such as field upgrades or recall repairs, require additional reliability testing to ensure the reliability of the modification. Since it is not possible to anticipate all the failure modes of a given system, especially ones with a human element, failures will occur. The reliability program also includes a systematic root cause analysis that

identifies the causal relationships involved in the failure such that effective corrective actions may be implemented. When possible, system failures and corrective actions are reported to the reliability engineering organization.

One of the most common methods to apply a Reliability Operational Assessment are Failure Reporting, Analysis and Corrective Action Systems (FRACAS). This systematic approach develops a reliability, safety and logistics assessment based on Failure / Incident reporting, management, analysis and corrective/preventive actions. Organizations today are adopting this method and utilize commercial systems such as a Web based FRACAS application enabling and organization to create a failure/incident data repository from which statistics can be derived to view accurate and genuine reliability, safety and quality performances.

Some of the common outputs from a FRACAS system includes: Field MTBF, MTTR, Spares Consumption, Reliability Growth, Failure/Incidents distribution by type, location, part no., serial no, symptom etc.

Reliability organizations

Systems of any significant complexity are developed by organizations of people, such as a commercial company or a government agency. The reliability engineering organization must be consistent with the company's organizational structure. For small, non-critical systems, reliability engineering may be informal. As complexity grows, the need arises for a formal reliability function. Because reliability is important to the customer, the customer may even specify certain aspects of the reliability organization.

There are several common types of reliability organizations. The project manager or chief engineer may employ one or more reliability engineers directly. In larger organizations, there is usually a product assurance or specialty engineering organization, which may include reliability, maintainability, quality, safety, human factors, logistics, etc. In such case, the reliability engineer reports to the product assurance manager or specialty engineering manager.

In some cases, a company may wish to establish an independent reliability organization. This is desirable to ensure that the system reliability, which is often expensive and time consuming, is not unduly slighted due to budget and schedule pressures. In such cases, the reliability engineer works for the project day-to-day, but is actually employed and paid by a separate organization within the company.

Because reliability engineering is critical to early system design, it has become common for reliability engineers, however the organization is structured, to work as part of an integrated product team.

Certification

The American Society for Quality has a program to become a Certified Reliability Engineer, CRE. Certification is based on education, experience, and a certification test: periodic recertification is required. The body of knowledge for the test includes: reliability management, design evaluation, product safety, statistical tools, design and development, modeling, reliability testing, collecting and using data, etc.

Another highly respected certification program is the CRP (Certified Reliability Professional). To achieve certification, candidates must complete a series of courses focused on important Reliability Engineering topics, successfully apply the learned body of knowledge in the workplace and publicly present this expertise in an industry conference or journal.

Reliability engineering education

Some Universities offer graduate degrees in Reliability Engineering (e.g, University of Tennessee, Knoxville, University of Maryland, College Park, Concordia University, Montreal, Canada, Monash University, Australia and Tampere University of Technology, Tampere, Finland). Other reliability engineers typically have an engineering degree, which can be in any field of engineering, from an accredited university or college program. Many engineering programs offer reliability courses, and some universities have entire reliability engineering programs. A reliability engineer may be registered as a Professional Engineer by the state, but this is not required by most employers. There are many professional conferences and industry training programs available for reliability engineers. Several professional organizations exist for reliability engineers, including the IEEE Reliability Society, the American Society for Quality (ASQ), and the Society of Reliability Engineers (SRE).

Chapter-14

Tools of Systems Engineering

System model

A **system model** is the conceptual model that describes and represents a system. A system comprises multiple views such as planning, requirement, design, implementation, deployment, operational, structure, behavior, input data, and output data views. A system model is required to describe and represent all these multiple views.

The system model describes and represents the multiple views possibly adopting two different approaches. The first one is the non-architectural approach and the second one is the architectural approach.

The non-architectural approach respectively picks a model for each view. For example, Structured Systems Analysis and Design Method (SSADM), picking the Structure Chart (SC) for structure description and the Data Flow Diagram (DFD) for behavior description, is categorized into the non-architectural approach.

The architectural approach, instead of picking many heterogeneous and unrelated models, will use only one single coalescence model. For example, System architecture, using the Architecture Description Language (ADL) for both structure and behavior descriptions, is categorized into the architectural approach.

System model may refer to:

- Community Climate System Model, a coupled Global Climate Model
- Human visual system model, a human visual system model used by image processing, video processing and computer vision
- Internal Family Systems Model, an integrative approach to psychotherapy, relationship counseling and family therapy
- Solar system model, a model that illustrate the relative positions and motions of the planets and stars
- Viable System Model, a model of the organisational structure of any viable or autonomous system

Systems architecture

A **system architecture** or **systems architecture** is the conceptual model that defines the structure, behavior, and more views of a system.

An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structure of the system which comprises system components, the externally visible properties of those components, the relationships (e.g. the behavior) between them, and provides a plan from which products can be procured, and systems developed, that will work together to implement the overall system. The language for architecture description is called the architecture description language (ADL).

Overview

There is no universally agreed definition of which aspects constitute a system architecture, and various organizations define it in different ways, including:

- The fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.
- The composite of the design architectures for products and their life cycle processes.
- A representation of a system in which there is a mapping of functionality onto hardware and software components, a mapping of the software architecture onto the hardware architecture, and human interaction with these components.
- An allocated arrangement of physical elements which provides the design solution for a consumer product or life-cycle process intended to satisfy the requirements of the functional architecture and the requirements baseline.
- An architecture is the most important, pervasive, top-level, strategic inventions, decisions, and their associated rationales about the overall structure (i.e., essential elements and their relationships) and associated characteristics and behavior.
- A description of the design and contents of a computer system. If documented, it may include information such as a detailed inventory of current hardware, software and networking capabilities; a description of long-range plans and priorities for future purchases, and a plan for upgrading and/or replacing dated equipment and software.
- A formal description of a system, or a detailed plan of the system at component level to guide its implementation.
- The structure of components, their interrelationships, and the principles and guidelines governing their design and evolution over time.

Systems architecture can best be thought of as a representation of an existent (or To Be Created) system, and the process and discipline for effectively implementing the design(s) for such a system. The set of relations (that is, embedded information) which an

architecture describes may be expressed in hardware, software, or something else (for example, organizational management describes many architectures whose nodes are people, knowledge management systems use nodes of metadescription).

- It is a *representation* because it is used to convey the informational content of the elements comprising a system, the relationships among those elements, and the rules governing those relationships.
- It is a *process* because a sequence of steps is prescribed to produce or change the architecture, and/or a design from that architecture, of a system within a set of constraints.
- It is a *discipline* because a body of knowledge is used to inform practitioners as to the most effective way to design the system within a set of constraints.

A systems architecture is primarily concerned with the internal interfaces among the system's components or subsystems, and the interface between the system and its external environment, especially the user. (In the specific case of computer systems, this latter, special interface, is known as the computer human interface, *AKA* human computer interface, or CHI; formerly called the man-machine interface.)

History

It is important to keep in mind that the modern systems architecture did not appear out of nowhere. Systems architecture depends heavily on practices and techniques which were developed over thousands of years in many other fields most importantly being, perhaps, civil architecture.

Prior to the advent of digital computers, the electronics and other engineering disciplines used the term system as it is still commonly used today. However, with the arrival of digital computers and the development of software engineering as a separate discipline, it was often necessary to distinguish among engineered hardware artifacts, software artifacts, and the combined artifacts. A programmable hardware artifact, or computing machine, that lacks its software program is impotent; even as a software artifact, or program, is equally impotent unless it can be used to alter the sequential states of a suitable (hardware) machine. However, a hardware machine and its software program can be designed to perform an almost illimitable number of abstract and physical tasks. Within the computer and software engineering disciplines (and, often, other engineering disciplines, such as communications), then, the term system came to be defined as containing all of the elements necessary (which generally includes both hardware and software) to perform a useful function.

Consequently, within these engineering disciplines, a system generally refers to a programmable hardware machine and its included program. And a systems engineer is defined as one concerned with the complete device, both hardware and software and, more particularly, all of the interfaces of the device, including that between hardware and software, and especially between the complete device and its user (the CHI). The hardware engineer deals (more or less) exclusively with the hardware device; the

software engineer deals (more or less) exclusively with the software program; and the systems engineer is responsible for seeing that the software program is capable of properly running within the hardware device, and that the system composed of the two entities is capable of properly interacting with its external environment, especially the user, and performing its intended function.

By analogy, then, a systems architecture makes use of elements of both software and hardware and is used to enable design of such a composite system. A good architecture may be viewed as a 'partitioning scheme,' or algorithm, which partitions all of the system's present and foreseeable requirements into a workable set of cleanly bounded subsystems with nothing left over. That is, it is a partitioning scheme which is exclusive, inclusive, and exhaustive. A major purpose of the partitioning is to arrange the elements in the sub systems so that there is a minimum of communications needed among them. In both software and hardware, a good sub system tends to be seen to be a meaningful "object". Moreover, a good architecture provides for an easy mapping to the user's requirements and the validation tests of the user's requirements. Ideally, a mapping also exists from every least element to every requirement and test.

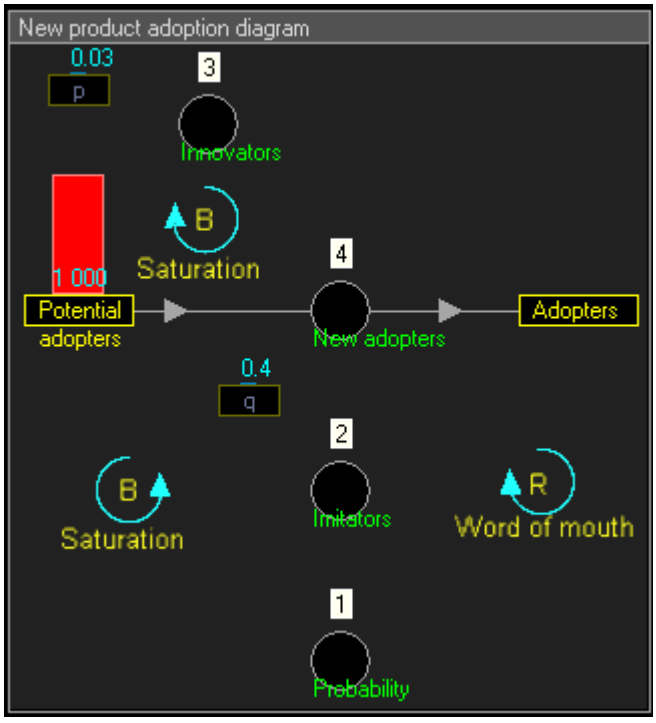
A *robust architecture* is said to be one that exhibits an optimal degree of fault-tolerance, backward compatibility, forward compatibility, extensibility, reliability, maintainability, availability, serviceability, usability, and such other quality attributes as necessary and/or desirable.

Types of systems architectures

Several types of systems architectures (underlain by the same fundamental principles) have been identified as follows:

- Collaborative Systems (such as the Internet, intelligent transportation systems, and joint air defense systems)
- Manufacturing Systems
- Social Systems
- Software and Information Technology Systems
- Strategic Systems Architecture

System dynamics



Dynamic stock and flow diagram of model *New product adoption* (model from article by John Sterman 2001)

System dynamics is an approach to understanding the behaviour of complex systems over time. It deals with internal feedback loops and time delays that affect the behaviour of the entire system. What makes using system dynamics different from other approaches to studying complex systems is the use of feedback loops and stocks and flows. These elements help describe how even seemingly simple systems display baffling nonlinearity.

Overview

System dynamics is a methodology and computer simulation modeling technique for framing, understanding, and discussing complex issues and problems. Originally developed in the 1950s to help corporate managers improve their understanding of industrial processes, system dynamics is currently being used throughout the public and private sector for policy analysis and design.

System dynamics is an aspect of systems theory as a method for understanding the dynamic behavior of complex systems. The basis of the method is the recognition that the structure of any system — the many circular, interlocking, sometimes time-delayed relationships among its components — is often just as important in determining its behavior as the individual components themselves. Examples are chaos theory and social dynamics. It is also claimed that because there are often properties-of-the-whole which

cannot be found among the properties-of-the-elements, in some cases the behavior of the whole cannot be explained in terms of the behavior of the parts.

History

System dynamics was created during the mid-1950s by Professor Jay Forrester of the Massachusetts Institute of Technology. In 1956, Forrester accepted a professorship in the newly-formed MIT School of Management. His initial goal was to determine how his background in science and engineering could be brought to bear, in some useful way, on the core issues that determine the success or failure of corporations. Forrester's insights into the common foundations that underlie engineering and management, which led to the creation of system dynamics, were triggered, to a large degree, by his involvement with managers at General Electric (GE) during the mid-1950s. At that time, the managers at GE were perplexed because employment at their appliance plants in Kentucky exhibited a significant three-year cycle. The business cycle was judged to be an insufficient explanation for the employment instability. From hand simulations (or calculations) of the stock-flow-feedback structure of the GE plants, which included the existing corporate decision-making structure for hiring and layoffs, Forrester was able to show how the instability in GE employment was due to the internal structure of the firm and not to an external force such as the business cycle. These hand simulations were the beginning of the field of system dynamics.

During the late 1950s and early 1960s, Forrester and a team of graduate students moved the emerging field of system dynamics from the hand-simulation stage to the formal computer modeling stage. Richard Bennett created the first system dynamics computer modeling language called SIMPLE (Simulation of Industrial Management Problems with Lots of Equations) in the spring of 1958. In 1959, Phyllis Fox and Alexander Pugh wrote the first version of DYNAMO (DYNAmic MOdels), an improved version of SIMPLE, and the system dynamics language became the industry standard for over thirty years. Forrester published the first, and still classic, book in the field titled Industrial Dynamics in 1961.

From the late 1950s to the late 1960s, system dynamics was applied almost exclusively to corporate/managerial problems. In 1968, however, an unexpected occurrence caused the field to broaden beyond corporate modeling. John Collins, the former mayor of Boston, was appointed a visiting professor of Urban Affairs at MIT. The result of the Collins-Forrester collaboration was a book titled Urban Dynamics. The Urban Dynamics model presented in the book was the first major non-corporate application of system dynamics.

The second major noncorporate application of system dynamics came shortly after the first. In 1970, Jay Forrester was invited by the Club of Rome to a meeting in Bern, Switzerland. The Club of Rome is an organization devoted to solving what its members describe as the "predicament of mankind" -- that is, the global crisis that may appear sometime in the future, due to the demands being placed on the Earth's carrying capacity (its sources of renewable and nonrenewable resources and its sinks for the disposal of pollutants) by the world's exponentially growing population. At the Bern meeting,

Forrester was asked if system dynamics could be used to address the predicament of mankind. His answer, of course, was that it could. On the plane back from the Bern meeting, Forrester created the first draft of a system dynamics model of the world's socioeconomic system. He called this model WORLD1. Upon his return to the United States, Forrester refined WORLD1 in preparation for a visit to MIT by members of the Club of Rome. Forrester called the refined version of the model WORLD2. Forrester published WORLD2 in a book titled World Dynamics.

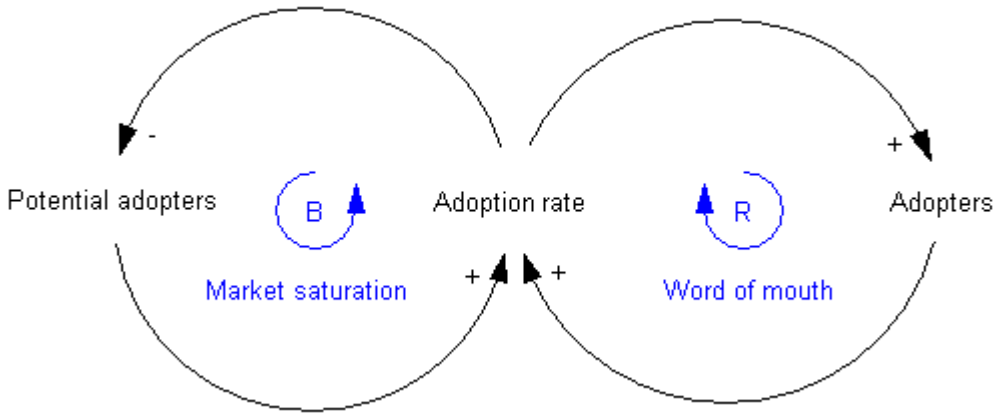
Topics in systems dynamics

The elements of system dynamics diagrams are feedback, accumulation of flows into stocks and time delays.

As an illustration of the use of system dynamics, imagine an organisation that plans to introduce an innovative new durable consumer product. The organisation needs to understand the possible market dynamics in order to design marketing and production plans.

Causal loop diagrams

A causal loop diagram is a visual representation of the feedback loops in a system. The causal loop diagram of the new product introduction may look as follows:

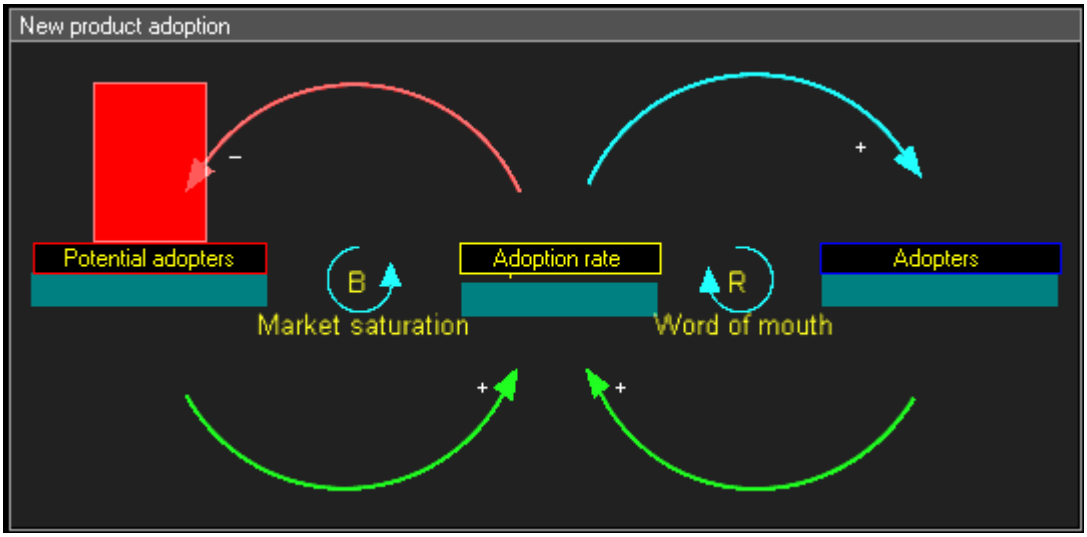


Causal loop diagram of *New product adoption* model

There are two feedback loops in this diagram. The positive reinforcement (labeled R) loop on the right indicates that the more people have already adopted the new product, the stronger the word-of-mouth impact. There will be more references to the product, more demonstrations, and more reviews. This positive feedback should generate sales that continue to grow.

The second feedback loop on the left is negative reinforcement (or "balancing" and hence labeled B). Clearly growth can not continue forever, because as more and more people adopt, there remain fewer and fewer potential adopters.

Both feedback loops act simultaneously, but at different times they may have different strengths. Thus one would expect growing sales in the initial years, and then declining sales in the later years.



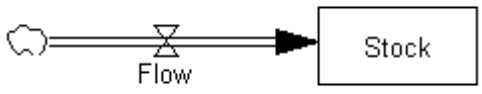
Causal loop diagram of *New product adoption* model with nodes values after calculus

In this dynamic causal loop diagram :

- step1 : (+) green arrows show that *Adoption rate* is function of *Potential Adopters* and *Adopters*
- step2 : (-) red arrow shows that *Potential adopters* decreases by *Adoption rate*
- step3 : (+) blue arrow shows that *Adopters* increases by *Adoption rate*

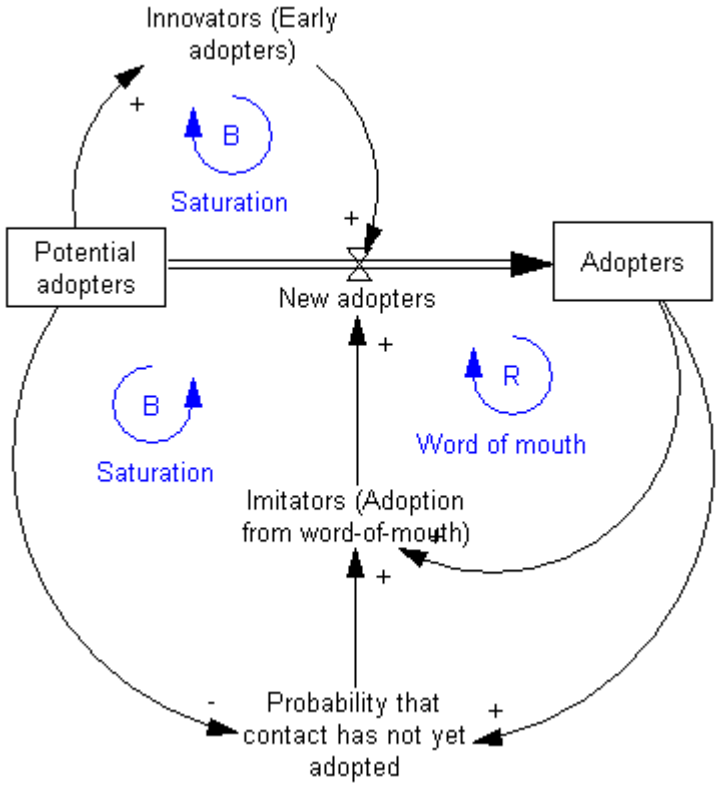
Stock and flow diagrams

The next step is to create what is termed a stock and flow diagram. A stock is the term for any entity that accumulates or depletes over time. A flow is the rate of change in a stock.



A flow is the rate of accumulation of the stock

In our example, there are two stocks: Potential adopters and Adopters. There is one flow: New adopters. For every new adopter, the stock of potential adopters declines by one, and the stock of adopters increases by one.



Stock and flow diagram of *New product adoption* model

Equations

The real power of system dynamics is utilised through simulation. Although it is possible to perform the modeling in a spreadsheet, there are a variety of software packages that have been optimised for this.

The steps involved in a simulation are:

- Define the problem boundary
- Identify the most important stocks and flows that change these stock levels
- Identify sources of information that impact the flows
- Identify the main feedback loops
- Draw a causal loop diagram that links the stocks, flows and sources of information
- Write the equations that determine the flows
- Estimate the parameters and initial conditions. These can be estimated using statistical methods, expert opinion, market research data or other relevant sources of information.
- Simulate the model and analyse results

In this example, the equations that change the two stocks via the flow are:

$$\text{Potential adopters} = \int_0^t \text{-New adopters } dt$$

$$\text{Adopters} = \int_0^t \text{New adopters } dt$$

List of all the equations, in their order of execution in each year, from year 1 to 15:

$$\text{1) Probability that contact has not yet adopted} = \frac{\text{Potential adopters}}{\text{Potential adopters} + \text{Adopters}}$$

$$\text{2) Imitators} = q \cdot \text{Adopters} \cdot \text{Probability that contact has not yet adopted}$$

$$\text{3) Innovators} = p \cdot \text{Potential adopters}$$

$$\text{4) New adopters} = \text{Innovators} + \text{Imitators}$$

$$\text{4.1) Potential adopters} - = \text{New adopters}$$

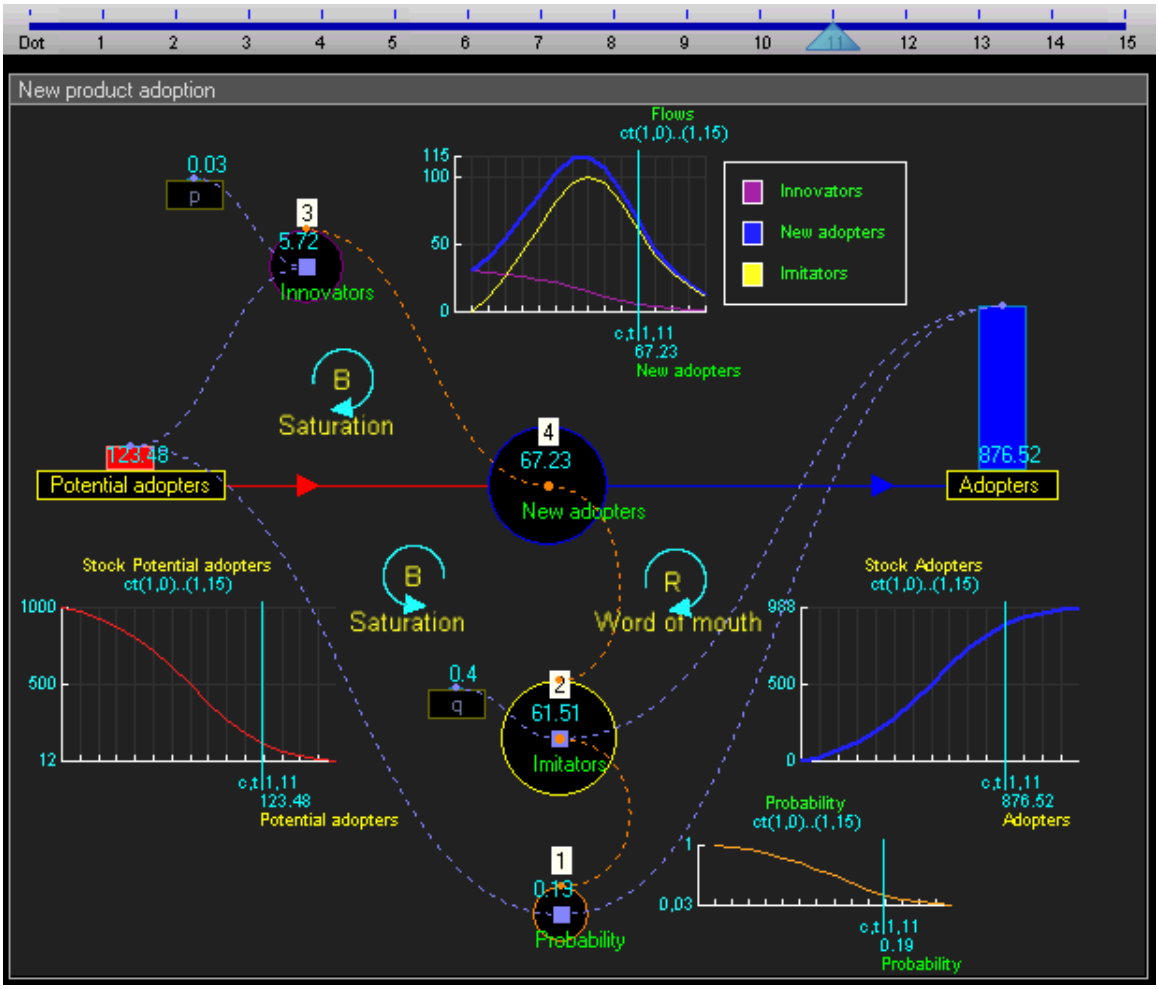
$$\text{4.2) Adopters} + = \text{New adopters}$$

$$p = 0.03$$

$$q = 0.4$$

Dynamic simulation results

The dynamic simulation results show that the behaviour of the system would be to have growth in **Adopters** that follows a classical s-curve shape.
The increase in **Adopters** is very slow initially, then exponential growth for a period, followed ultimately by saturation.



Dynamic stock and flow diagram of *New product adoption* model

Year	Probability	Imitators	Innovators	New adopters	Potential adopters	Adopters
0	0,00	0,00	0,00	0,00	1 000,00	0,00
1	1,00	0,00	30,00	30,00	970,00	30,00
2	0,97	11,64	29,10	40,74	929,26	70,74
3	0,93	26,32	27,88	54,20	875,06	124,94
4	0,88	43,98	26,25	70,23	804,83	195,17
5	0,80	62,45	24,14	86,59	718,24	281,76
6	0,72	81,15	21,55	102,70	615,54	384,46
7	0,62	95,35	18,47	113,82	501,72	498,28
8	0,50	99,66	15,05	114,71	387,01	612,99
9	0,39	95,63	11,61	107,24	279,77	720,23
10	0,28	80,67	8,39	89,06	190,71	809,29
11	0,19	61,51	5,72	67,23	123,48	876,52
12	0,12	42,07	3,70	45,77	77,71	922,29
13	0,08	29,51	2,33	31,84	45,87	954,13
14	0,05	19,08	1,38	20,46	25,41	974,59
15	0,03	11,70	0,76	12,46	12,95	987,05

Stocks and flows values for years = 0 to 15

Application

System dynamics has found application in a wide range of areas, for example population, ecological and economic systems, which usually interact strongly with each other.

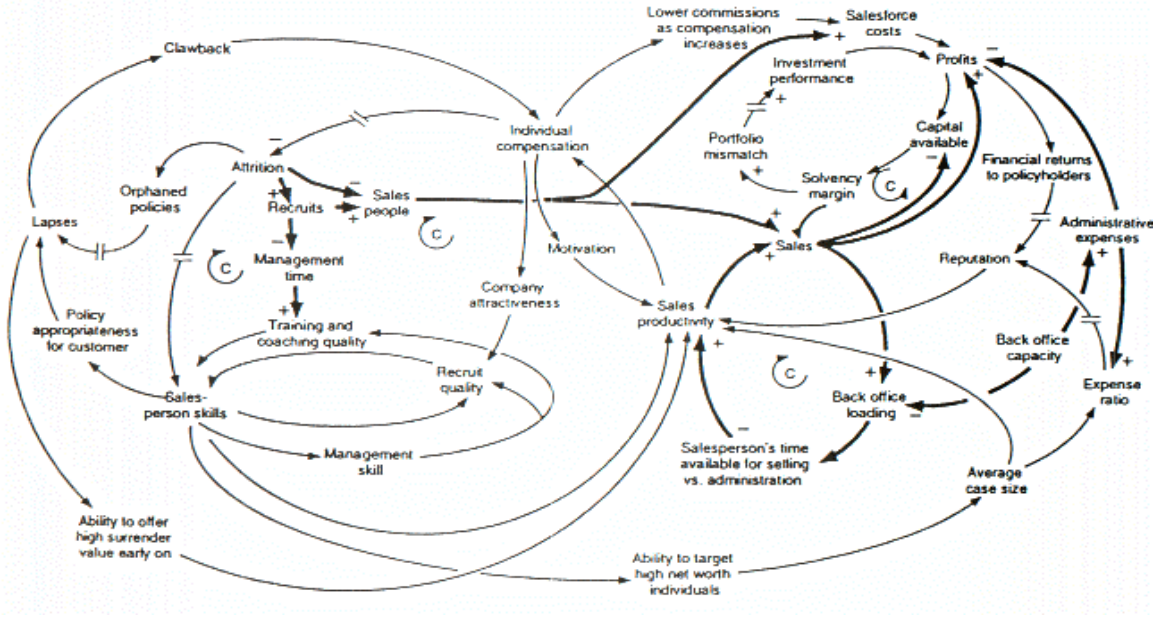
System dynamics have various "back of the envelope" management applications. They are a potent tool to:

- Teach system thinking reflexes to persons being coached
- Analyze and compare assumptions and mental models about the way things work
- Gain qualitative insight into the workings of a system or the consequences of a decision
- Recognize archetypes of dysfunctional systems in everyday practice

Computer software is used to simulate a system dynamics model of the situation being studied. Running "what if" simulations to test certain policies on such a model can greatly aid in understanding how the system changes over time. System dynamics is very similar to systems thinking and constructs the same causal loop diagrams of systems with feedback. However, system dynamics typically goes further and utilises simulation to study the behaviour of systems and the impact of alternative policies.

System dynamics has been used to investigate resource dependencies, and resulting problems, in product development.

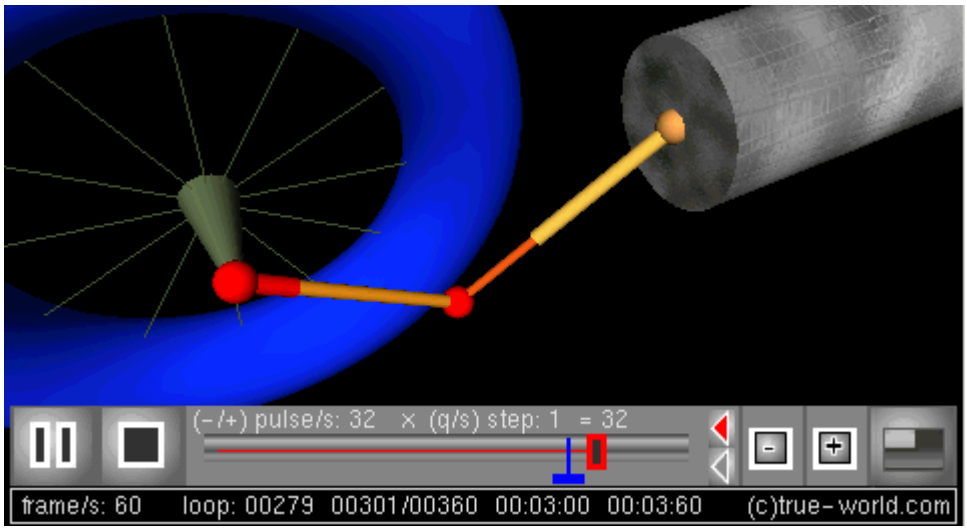
Example



Causal loop diagram of a model examining the growth or decline of a life insurance company.

The figure above is a causal loop diagram of a system dynamics model created to examine forces that may be responsible for the growth or decline of life insurance companies in the United Kingdom. A number of this figure's features are worth mentioning. The first is that the model's negative feedback loops are identified by "C's," which stand for "Counteracting" loops. The second is that double slashes are used to indicate places where there is a significant delay between causes (i.e., variables at the tails of arrows) and effects (i.e., variables at the heads of arrows). This is a common causal loop diagramming convention in system dynamics. Third, is that thicker lines are used to identify the feedback loops and links that author wishes the audience to focus on. This is also a common system dynamics diagramming convention. Last, it is clear that a decision maker would find it impossible to think through the dynamic behavior inherent in the model, from inspection of the figure alone.

Example of 4D piston motion



Piston motion equations

This animation was made with the 3D modeler of a system dynamics software.
The calculated values are associated with parameters of the rod and crank.
In this example the crank is driving, we vary both the speed of rotation, its radius and the length of the rod, the piston follows.

Systems analysis

Systems analysis is the study of sets of interacting entities, including computer systems analysis. This field is closely related to operations research. It is also "an explicit formal inquiry carried out to help someone (referred to as the decision maker) identify a better course of action and make a better decision than he might otherwise have made."

Overview

The terms analysis and synthesis come from Greek where they mean respectively "to take apart" and "to put together". These terms are in scientific disciplines from mathematics and logic to economy and psychology to denote similar investigative procedures. Analysis is defined as the procedure by which we break down an intellectual or substantial whole into parts or . Synthesis is defined as the : to basant combine separate elements or components in order to form a coherent whole. Systems analysis researchers apply methodology to the analysis of systems involved to form an overall picture.

Information technology

The development of a computer-based information system includes a systems analysis phase which produces or enhances the data model which itself is a precursor to creating or enhancing a database. There are a number of different approaches to system analysis. When a computer-based information system is developed, systems analysis (according to the Waterfall model) would constitute the following steps:

- The development of a feasibility study, involving determining whether a project is economically, socially, technologically and organizationally feasible.
- Conducting fact-finding measures, designed to ascertain the requirements of the system's end-users. These typically span interviews, questionnaires, or visual observations of work on the existing system.
- Gauging how the end-users would operate the system (in terms of general experience in using computer hardware or software), what the system would be used for etc.

Another view outlines a phased approach to the process. This approach breaks systems analysis into 5 phases:

- Scope definition
- Problem analysis
- Requirements analysis
- Logical design
- Decision analysis

Use cases are a widely-used systems analysis modeling tool for identifying and expressing the functional requirements of a system. Each use case is a business scenario or event for which the system must provide a defined response. Use cases evolved out of object-oriented analysis; however, their use as a modeling tool has become common in many other methodologies for system analysis and design.

Practitioners

Practitioners of systems analysis are often called up to dissect systems that have grown haphazardly to determine the current components of the system. This was shown during the year 2000 re-engineering effort as business and manufacturing processes were examined as part of the Y2K automation upgrades. Employment utilizing systems analysis include systems analyst, business analyst, manufacturing engineer, enterprise architect, etc.

While practitioners of systems analysis can be called upon to create new systems they often modify, expand or document existing systems (processes, procedures and methods). A set of components interact with each other to accomplish some specific purpose. Systems are all around us. Our body is itself a system. A business is also a system. Men, money, machine, market and material are the components of business system that work together that achieve the common goal of the organization.

Characteristics: >> Systems are on-going never ending and continuous process. >> A system may be closed or open. All on-going system are open thus closed systems exist only as a concept. >> A system must have: Standard for acceptable performance

```
      + A method of comparing actual performance with standard
performance
      + A method of measuring actual performance.
      + A method for feedback.
```

Chapter-15

Primary Areas of Product Lifecycle Management

Project portfolio management

Project Portfolio Management (**PPM**) is a term used by project managers and project management (PM) organizations, (or PMO's), to describe methods for analyzing and collectively managing a group of current or proposed projects based on numerous key characteristics. The fundamental objective of PPM is to determine the optimal mix and sequencing of proposed projects to best achieve the organization's overall goals - typically expressed in terms of hard economic measures, business strategy goals, or technical strategy goals - while honoring constraints imposed by management or external real-world factors. Typical attributes of projects being analyzed in a PPM process include each project's total expected cost, consumption of scarce resources (human or otherwise) expected timeline and schedule of investment, expected nature, magnitude and timing of benefits to be realized, and relationship or inter-dependencies with other projects in the portfolio.

The key challenge to implementing an effective PPM process is typically securing the mandate to do so. Many organizations are culturally inured to an informal method of making project investment decisions, which can be compared to political processes observable in the U.S. legislature. However this approach to making project investment decisions has led many organizations to unsatisfactory results, and created demand for a more methodical and transparent decision making process. That demand has in turn created a commercial marketplace for tools and systems which facilitate such a process.

Some commercial vendors of PPM software emphasize their products' ability to treat projects as part of an overall investment portfolio. PPM advocates see it as a shift away from one-off, ad hoc approaches to project investment decision making. Most PPM tools and methods attempt to establish a set of values, techniques and technologies that enable visibility, standardization, measurement and process improvement. PPM tools attempt to enable organizations to manage the continuous flow of projects from concept to completion.

Treating a set of projects as a portfolio would be, in most cases, an improvement on the ad hoc, one-off analysis of individual project proposals. The relationship between PPM

techniques and existing investment analysis methods is a matter of debate. While many are represented as "rigorous" and "quantitative", few PPM tools attempt to incorporate established financial portfolio optimization methods like modern portfolio theory or Applied Information Economics, which have been applied to project portfolios, including even non-financial issues.

Controversy over the "investment discipline" of PPM

Developers of PPM tools see their solutions as borrowing from the financial investment world. However, other than using the word "portfolio", few can point to any specific portfolio optimization methods implemented in their tools.

A project can be viewed as a composite of resource investments such as skilled labour and associated salaries, IT hardware and software, and the opportunity cost of deferring other project work. As project resources are constrained, business management can derive greatest value by allocating these resources towards project work that is objectively and relatively determined to meet business objectives more so than other project opportunities. Thus, the decision to invest in a project can be made based upon criteria that measures the relative benefits (eg. supporting business objectives) and its relative costs and risks to the organization.

In principle, PPM attempts to address issues of resource allocation, e.g., money, time, people, capacity, etc. In order for it to truly borrow concepts from the financial investment world, the portfolio of projects and hence the PPM movement should be grounded in some financial objective such as increasing shareholder value, top line growth, etc. Equally important, risks must be computed in a statistically, actuarially meaningful sense. Optimizing resources and projects without these in mind fails to consider the most important resource any organization has and which is easily understood by people throughout the organization whether they be IT, finance, marketing, etc and that resource is money.

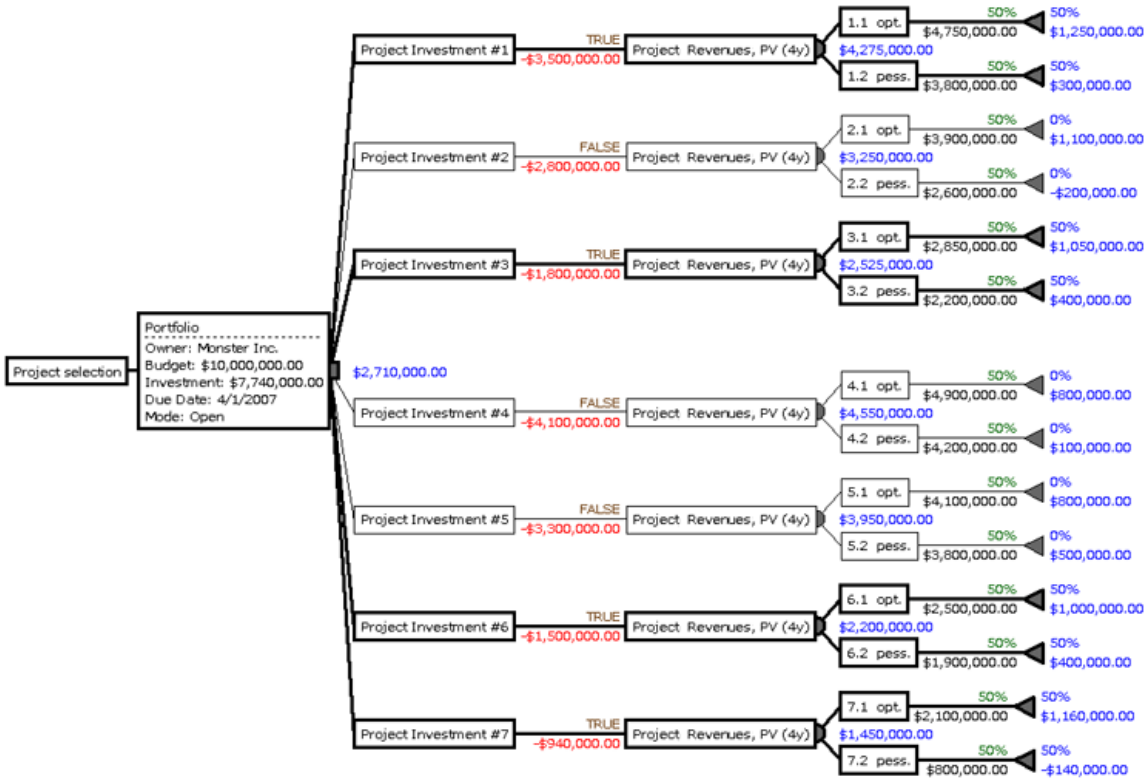
While being tied largely to IT and fairly synonymous with IT portfolio management, PPM is ultimately a subset of corporate portfolio management and should be exportable/utilized by any group selecting and managing discretionary projects. However, most PPM methods and tools opt for various subjective weighted scoring methods, not quantitatively rigorous methods based on options theory, modern portfolio theory, Applied Information Economics or operations research.

Beyond the project investment decision, PPM aims to support ongoing measurement of the project portfolio so each project can be monitored for its relative contribution to business goals. If a project is either performing below expectations (cost overruns, benefit erosion) or is no longer highly aligned to business objectives (which change with natural market and statutory evolution), management can choose to decommit from a project and redirect its resources elsewhere. This analysis, done periodically, will "refresh" the portfolio to better align with current states and needs.

Historically, many organizations were criticized for focusing on "doing the wrong things well." PPM attempts to focus on a fundamental question: "Should we be doing this project or this portfolio of projects at all?" One litmus test for PPM success is to ask "Have you ever canceled a project that was on time and on budget?" With a true PPM approach in place, it is much more likely that the answer is "yes." As goals change so should the portfolio mix of what projects are funded or not funded no matter where they are in their individual lifecycles. Making these portfolio level business investment decisions allows the organization to free up resources, even those on what were before considered "successful" projects, to then work on what is really important to the organization.

Optimizing for payoff

One method PPM tools or consultants might use is the use of decision trees with decision nodes that allow for multiple options and optimize against a constraint. The organization in the following example has options for 7 projects but the portfolio budget is limited to $10,000,000. The selection made are the projects 1, 3, 6 and 7 with a total investment of $7,740,000 - the optimum under these conditions. The portfolio's payoff is $2,710,000.



Presumably, all other combinations of projects would either exceed the budget or yield a lower payoff. However, this is an extremely simplified representation of risk and is unlikely to be realistic. Risk is usually a major differentiator among projects but it is difficult to quantify risk in a statistically and actuarially meaningful manner (with probability theory, Monte Carlo Method, statistical analysis, etc.). This places limits on

the deterministic nature of the results of a tool such as a decision tree (as predicted by modern portfolio theory).

An approach to include uncertainty and risk in portfolio optimization takes a decision centric view of the project portfolio optimization process . In this view there are five key decisions that must be made:

- Decision D1: Decide what strategic initiatives, benefits, and resource limitation criteria (i.e. measures) to use for project filtering and portfolio ranking.
- Decision D2: Decide which criteria are most important to achieve.
- Decision D3: Decide which project ideas or needs are worth developing into business cases?
- Decision D4: Decide which Business Cases should be considered as part of the portfolio
- Decision D5: Decide which projects to fund

For each decision is imperative to fuse both qualitative and quantitative evaluations of alternatives where each measure includes uncertainty. It is the uncertainty in the strategic alignment, value benefits and resources demand that cause risk and thus drives all five decisions.

Resource allocation

Resource allocation is a critical component of PPM. Once it is determined that one or many projects meet defined objectives, the available resources of an organization must be evaluated for its ability to meet project demand (aka as a demand "pipeline" discussed below). Effective resource allocation typically requires an understanding of existing labor or funding resource commitments (in either business operations or other projects) as well as the skills available in the resource pool. Project investment should only be made in projects where the necessary resources are available during a specified period of time.

Resources may be subject to physical constraints. For example, IT hardware may not be readily available to support technology changes associated with ideal implementation timeframe for a project. Thus, a holistic understanding of all project resources and their availability must be conjoined with the decision to make initial investment or else projects may encounter substantial risk during their lifecycle when unplanned resource constraints arise to delay achieving project objectives.

Beyond the project investment decision, PPM involves ongoing analysis of the project portfolio so each investment can be monitored for its relative contribution to business goals versus other portfolio investments. If a project is either performing below expectations (cost overruns, benefit erosion) or is no longer aligned to business objectives (which change with natural market and statutory evolution), management can choose to decommit from a project to stem further investment and redirect resources towards other projects that better fit business objectives. This analysis can typically be performed on a periodic basis (eg. quarterly or semi-annually) to "refresh" the portfolio for optimal

business performance. In this way both new and existing projects are continually monitored for their contributions to overall portfolio health. If PPM is applied in this manner, management can more clearly and transparently demonstrate its effectiveness to its shareholders or owners.

Implementing PPM at the enterprise level faces a challenge in gaining enterprise support because investment decision criteria and weights must be agreed to by the key stakeholders of the organization, each of whom may be incentivised to meet specific goals that may not necessarily align with those of the entire organization. But if enterprise business objectives can be manifested in and aligned with the objectives of its distinct business unit sub-organizations, portfolio criteria agreement can be achieved more easily. (Assadourian 2005)

From a requirements management perspective Project Portfolio Management can be viewed as the upper-most level of business requirements management in the company, seeking to understand the business requirements of the company and what portfolio of projects should be undertaken to achieve them. It is through portfolio management that each individual project should receive its allotted business requirements (Denney 2005).

Pipeline management

In addition to managing the mix of projects in a company, Project Portfolio Management must also determine whether (and how) a set of projects in the portfolio can be executed by a company in a specified time, given finite development resources in the company. This is called *pipeline management*. Fundamental to pipeline management is the ability to measure the planned allocation of development resources according to some strategic plan. To do this, a company must be able to estimate the effort planned for each project in the portfolio, and then roll the results up by one or more strategic project types e.g., effort planned for research projects. (Cooper et al. 1998); (Denney 2005) discusses project portfolio and pipeline management in the context of use case driven development.

Organizational applicability

The complexity of PPM and other approaches to IT projects (e.g., treating them as a capital investment) may render them not suitable for smaller or younger organizations. An obvious reason for this is that a few IT projects doesn't make for much of a portfolio selection. Other reasons include the cost of doing PPM—the data collection, the analysis, the documentation, the education, and the change to decision-making processes.

Product design

Example of designed product - Roomba robotic vacuum cleaner.

Product design is concerned with the efficient and effective generation and development of ideas through a process that leads to new products.

Product Designers conceptualize and evaluate ideas, making them tangible through products in a more systematic approach. Their role is to combine art, science and technology to create tangible three-dimensional goods. This evolving role has been facilitated by digital tools that allow designers to communicate, visualize and analyze ideas in a way that would have taken greater manpower in the past.

Product design is sometimes confused with industrial design, industrial design is concerned with the aspect of that process that brings that sort of artistic form and usability usually associated with craft design to that of mass produced goods.

Process

Product designers follow various methodology that requires a specific skill set to complete.

Initial Stage

- **Idea Generation** can be from imagination, observation, or research.
- **Need Based Generation** can be from the need to solve a problem, the need to follow the popular trends, or the need for a product to do a specific task.

Mid Stage

- **Design Solutions** arise from meeting user needs, concept development, form exploration, ergonomics, prototyping, materials, and technology.
- **Production** involves fabrication and manufacturing the design.

Final Stage

- **Marketing** involves selling the product. It can either be client based which mean the a client buys the design and manufactures it and then sells it to customers. Or it can be user based where the product is sold directly to the user by the designer.

Application

Product design ranges from furniture, electronics, lighting, tools, toys, and general everyday objects.

Manufacturing process management

Manufacturing process management (**MPM**) is a collection of technologies and methods used to define how products are to be manufactured. MPM differs from ERP/MRP which is used to plan the ordering of materials and other resources, set manufacturing schedules, and compile cost data.

A cornerstone of MPM is the central repository for the integration of all these tools and activities aids in the exploration of alternative production line scenarios; making assembly lines more efficient with the aim of reduced lead time to product launch, shorter product times and reduced work in progress (WIP) inventories as well as allowing rapid response to product or product changes.

Topics and technology

- Production process planning
    - Manufacturing concept planning
    - Factory layout planning and analysis
        - work flow simulation.
        - walk-path assembly planning
        - plant design optimization

    - Mixed model line balancing.
    - Workloads on multiple stations.
    - Process simulation tools e.g. die press lines, manufacturing lines
    - Ergonomic simulation and assessment of production assembly tasks
    - Resource planning
- Computer-aided manufacturing (CAM)
    - Numerical control CNC
    - Direct Numerical Control (DNC)
    - Tooling/equipment/fixtures development
    - Tooling and Robot work-cell setup and offline programming (OLP)
- Generation of shop floor work instructions
- Time and cost estimates
    - ABC - Manufacturing activity-based costing
    - Production, costs, and pricing
- Quality Computer-aided quality assurance (CAQ)
    - FMEA Failure mode and effects analysis
    - SPC Statistical process control
    - Computer aided inspection with coordinate-measuring machine (CMM)
    - Tolerance stack-up analysis using PMI models.
- Success Measurements
    - Overall Equipment Effectiveness (OEE),
- Communication with other systems
    - Enterprise resource planning (ERP)
    - Manufacturing Operations Management (MOM)
    - Product Data Management (PDM)
    - SCADA (Supervisory Control and Data Acquisition) real time process monitoring and control
    - Human-machine interface (HMI) (or *man-machine interface* (MMI))
    - Distributed control system (DCS)

Product data management

Product data management (**PDM**) is the business function often within product lifecycle management that is responsible for the creation, management and publication of product data.

Introduction

Product data management **(PDM)** is the use of software or other tools to track and control data related to a particular product. The data tracked usually involves the technical specifications of the product, specifications for manufacture and development, and the types of materials that will be required to produce goods. The use of product data

management allows a company to track the various costs associated with the creation and launch of a product. Product data management is part of product life cycle management, and is primarily used by engineers.

Within PDM the focus is on managing and tracking the creation, change and archive of all information related to a product. The information being stored and managed (on one or more file servers) will include engineering data such as Computer-aided design (CAD) models, drawings and their associated documents.

Product data management (PDM) serves as a central knowledge repository for process and product history, and promotes integration and data exchange among all business users who interact with products — including project managers, engineers, sales people, buyers, and quality assurance teams.

The central database will also manage metadata such as owner of a file and release status of the components. The package will: control check-in and check-out of the product data to multi-user; carry out engineering change management and release control on all versions/issues of components in a product; build and manipulate the product structure bill of materials (BOM) for assemblies; and assist in configurations management of product variants.

This enables automatic reports on product costs, etc. Furthermore, PDM enables companies producing complex products to spread product data into the entire PLM launch-process. This significantly enhances the effectiveness of the launch process.

Product data management is focused on capturing and maintaining information on products and/or services through its development and useful life. Typical information managed in the PDM module include

- Part number
- Part description
- Supplier/vendor
- Vendor part number and description
- Unit of measure
- Cost/price
- Schematic or CAD drawing
- Material data sheets

PDM Advantages:

- Track and manage all changes to product related data
- Accelerate return on investment with easy setup;
- Spend less time organizing and tracking design data;
- Improve productivity through reuse of product design data;
- Enhance collaboration.

History of PDM

PDM stems from traditional engineering design activities that created product drawings and schematics on paper and using CAD tools to create parts lists (Bills of Material structures - BOM). The PDM and BOM data is used in enterprise resource planning (ERP) systems to plan and coordinate all transactional operations of a company (sales order management, purchasing, cost accounting, logistics, etc.)

PDM is a subset of a larger concept of product lifecycle management (PLM). PLM encompasses the processes needed to launch new products (NPI), manage changes to existing products (ECN/ECO) and retire products at the end of their life (End of Life) .

Capabilities

PDM systems vary in their functionality, but some of their common capabilities are described below.

- Access control

Access control to each element in the product definition data base can be specified. Read only access can be given to personnel not directly involved with the design, development and planning process. Creation and maintenance access can be given to the individuals responsible for product and process design. As Product Data Management systems evolve towards Collaborative Product Commerce (CPC) systems which are used across multiple enterprises in a supply chain, access control becomes more critical and requires control to limit access to specific projects, products or parts for a specific supplier or customer

- Component / Material Classification

Components and materials can be classified and organized and attributes assigned. This supports standardization by identifying similar components/materials, eliminating redundancy, and establishing a preferred parts list. Establishing classes and subclasses with attributes allows a designer to search and select a needed material, component or assembly with minimal effort thereby avoiding having to re-specifying an existing or similar component or material

- Product Structure

Since the relationship of a product's parts is a logical one maintained by the information system rather than a fixed physical relationship as represented on a drawing, it is possible to readily maintain more than one relationship. This will allow different views of part relationships in assemblies to correspond to the various departmental needs (e.g., engineering and manufacturing product structures), while maintaining rigor and consistency of the product's definition through this single data base. Thus, this one logical data base can support product and process design requirements as well as maintain part relationships to serve as a manufacturing bill of materials for MRP II/ERP. In other

words, PDM provides the ability to hold not just the physical relationships between parts in an assembly but also other kinds of structures; for instance, manufacturing, financial, maintenance or document relationships. So, it is possible for specialist team members to see the product structured from their point of view. Product data can be accessed via this complete Bill of Materials. This access includes assemblies, parts and related documents.An integrated approach to developing, organizing and maintaining part and product definition data facilitates the design process, makes design data more readily usable and enhances integration with process requirements

- Engineering Changes

Engineering changes can be facilitated with this configuration management and administrative control embedded within the system. CAE/CAD tools will enable engineering changes to be more thoroughly developed and analyzed to better define change impact. Once a design has been created, it can be checked-out electronically to a workstation for engineering changes. When the changes have been made, it can be returned to the central database and placed in a queue or an email notification sent for approval by designated parties. In this manner, a Change Control Board (CCB) can even "convene" and provide individual member's input electronically. In addition to supporting engineering analysis, information related to procurement, inventory, manufacturing and cost is available for members of the CCB to evaluate, designate the effectivity of the change and determine the disposition of existing items.

- Process Management and Workflow

PDM systems support process management by defining process steps related to the development, distribution and use of product data. The process is defined in the form of specified process steps and release or promotion levels that the data must achieve. The manner in which the process is defined varies with every PDM system. Within a project, responsibilities are defined for the process steps - who needs to approve the data or work on the data before it moves to the next release or promotion level. While, the current process is defined in a company's configuration management or engineering change procedures and in its new product development process, often changes have to be made to take advantage of the communication and coordination capabilities of the PDM system. This new data is moved to the next person's "in basket" within PDM or an email notification is sent.

- Collaboration

Collaboration can be supported in several ways. First, a PDM system may be the gateway that a team uses to access the information under discussion avoiding the need to copy and distribute a series of paper documents. Second, the PDM system may provide a synchronous or asynchronous collaboration environment for team members to access, present, review and produce feedback on product and process information. Further, this collaboration tool may incorporate viewing and mark-up capability, as well as provide the ability to store marked-up files or documents submitted by collaborators. Third, what

are now described as collaborative product commerce systems (CPC), provide extended PDM functionality and access control outside the enterprise for customers, suppliers and interested third parties (e.g., regulatory agencies). This speeds the distribution of information, enhances coordination, and speeds the capture of feedback.

- Storage and retrieval of product information
- Product structure modeling and management
    - Bill of materials
    - Product configurations
    - Product variations
    - Product versions
- Project tracking
- Resource planning