Software Specification tools

Introduction

There are a variety of tools and techniques used in developing and representing software specifications. Most of these tools and techniques have been used at one time or another by practitioners, some with great success.

Software Specification Tools

- Data Dictionary
- Data Flow Diagrams
- Finite State Machine
- Petri Nets
- Operational Timelines
- Mathematical Logic
- Decision Support Tools

Data Dictionary

In the early 1970s system analysts realized that they needed more systematic and logical methods of describing systems. The proliferation of computers and the collection of large amounts of information in databases often led to a great deal of confusion and inconsistency. t is a composite collection of specifications about the nature of data and information. It is repository of descriptions of the form, style, and content of data as well as of the methods that will be used to process and report it. A data dictionary is in fact a database implementation as well as they contain data information about data.

It is typically structured in tables and views just like other data in a database. Most data dictionaries are central to a database and are very important tool for kinds of users from the data consumers to application designers to database developers and administrators.

A data dictionary is used when finding information about users, objects, schema and storage structures. Every time a data definition language (DDL) statement is issued, the data dictionary becomes modified.

The dictionary may also include standard tables of codes or words and their meanings, as well as alternative names or definitions.

For example, EMPLOYEE, NUMBER and SERIAL NUMBER might be equated and both might reference an employee's identification number.

The data dictionary is the foundation of structure systems analysis. It provides the standards and uniform format by which all elements or parts of a system are designed and coordinated. It places all information in a structure or hierarchy. At the top of the hierarchy is the data element, the data element is the smallest unit of a data that will be processed or become part of a record.

Data Flow Diagrams

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information systems, modelling its *process* aspects. Often they are a preliminary step used to create an overview of the system which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

A DFD shows what kinds of data will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of processes, or information about whether processes will operate in sequence or in parallel.

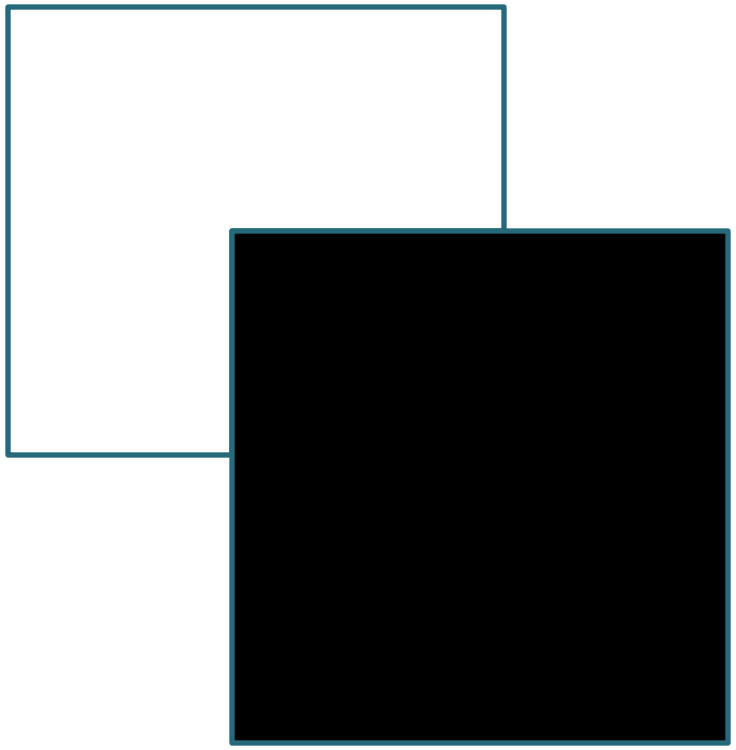
On the context diagram the system's interactions with the outside world are modelled purely in terms of data flows across the *system boundary*. The context diagram shows the entire system as a single process, and gives no clues as to its internal organization.

This context-level DFD is next "exploded", to produce a Level 0 DFD that shows some of the detail of the system being modeled. The Level 0 DFD shows how the system is divided into sub-systems (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present in order for the system to do its job, and shows the flow of data between the various parts of the system.

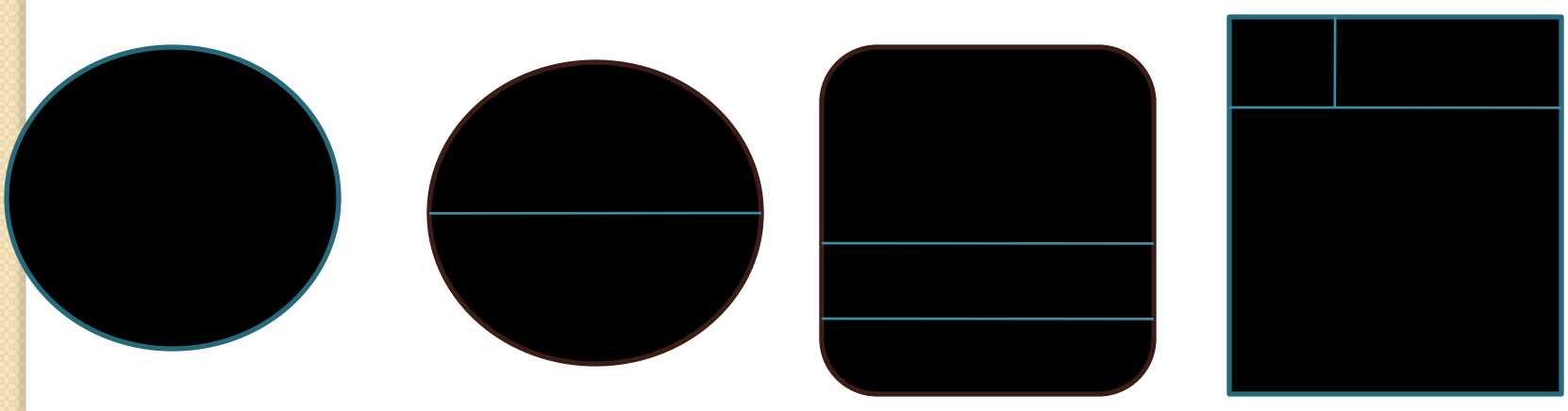
The sponsor of a project and the end users will need to be briefed and consulted throughout all stages of a system's evolution. With a data flow diagram, users are able to visualize how the system will operate, what the system will accomplish, and how the system will be implemented. The old system's dataflow diagrams can be drawn up and compared with the new system's data flow diagrams to draw comparisons to implement a more efficient system. Data flow diagrams can be used to provide the end user with a physical idea of where the data they input ultimately has an effect upon the structure of the whole system from order to dispatch to report. How any system is developed can be determined through a data flow diagram.

Data Flow Diagrams

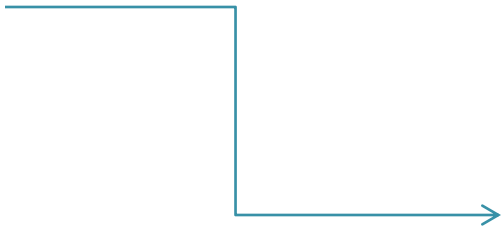
- External Entity



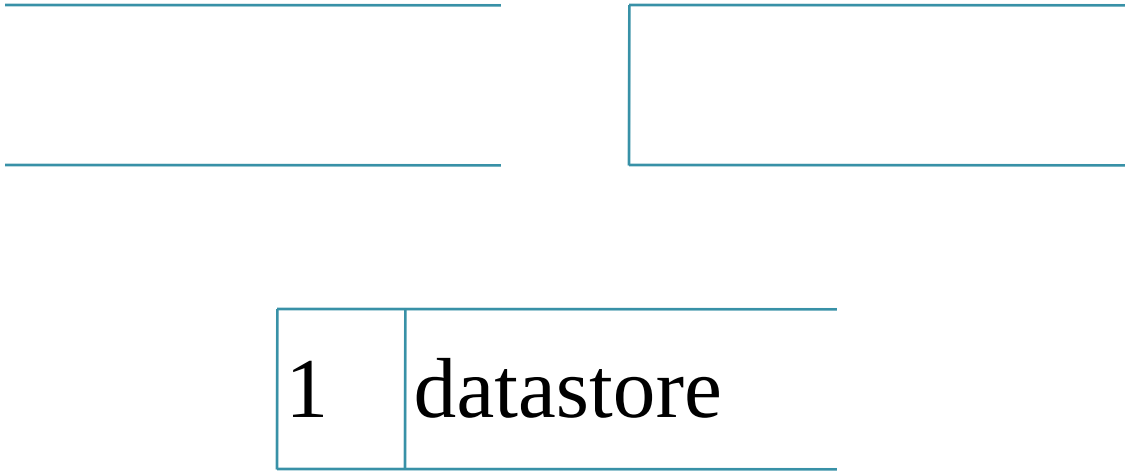
- Process



- Data flow



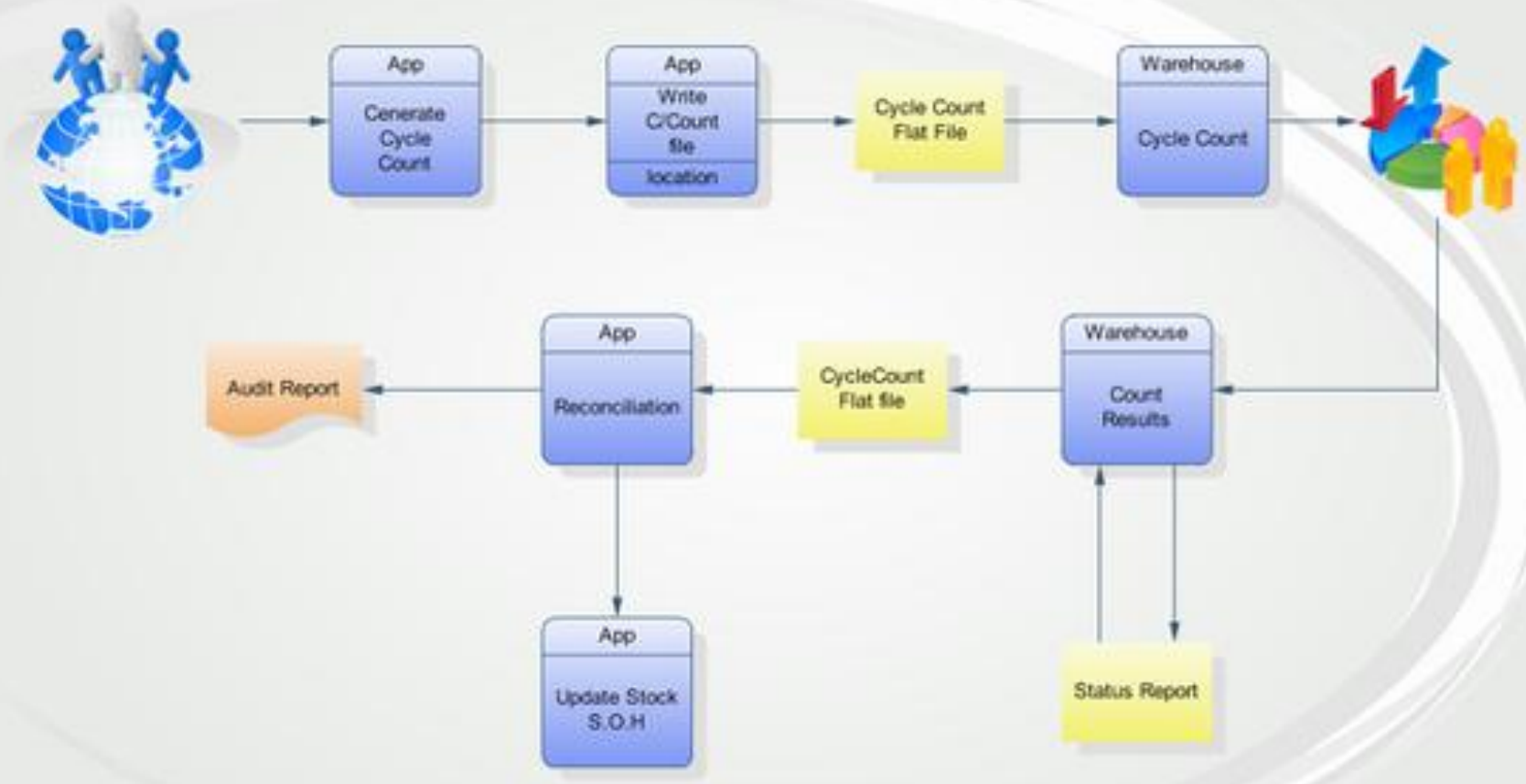
- Data Store



1	datastore

- Resource flow

Data Flow Model Diagram



Finite State Machine

A **finite-state machine** (**FSM**) or **finite-state automaton** (plural: *automata*), or simply a **state machine**, is a mathematical abstraction sometimes used to design digital logic or computer programs. It is a behavior model composed of a finite number of states, transitions between those states, and actions, similar to a flow graph in which one can inspect the way logic runs when certain conditions are met. It has finite internal memory, an input feature that reads symbols in a sequence, one at a time without going backward; and an output feature, which may be in the form of a user interface, once the model is implemented. The operation of an FSM begins from one of the states (called a *start state*), goes through transitions depending on input to different states and can end in any of those available, however only a certain set of states mark a successful flow of operation (called *accept states*).

Finite-state machines can solve a large number of problems, among which are electronic design automation, communication protocol design, parsing and other engineering applications. In biology and artificial intelligence research, state machines or hierarchies of state machines are sometimes used to describe neurological systems and in linguistics—to describe the grammars of natural languages.

State Transition Table

Current state → Input ↓	State A	State B	State C
Input X	...	...	...
Input Y	...	State C	...
Input Z	...	...	...

Petri Nets

A **Petri net** (also known as a **place/transition net** or **P/T net**) is one of several mathematical modeling languages for the description of distributed systems. A Petri net is a directed bipartite graph, in which the nodes represent transitions (i.e. events that may occur, signified by bars) and places (i.e. conditions, signified by circles). The directed arcs describe which places are pre- and/or postconditions for which transitions (signified by arrows). Some sources state that Petri nets were invented in August 1939 by Carl Adam Petri – at the age of 13 – for the purpose of describing chemical processes.

Like industry standards such as UML activity diagrams, BPMN and EPCs, Petri nets offer a graphical notation for stepwise processes that include choice, iteration, and concurrent execution. Unlike these standards, Petri nets have an exact mathematical definition of their execution semantics, with a well-developed mathematical theory for process analysis.

Mathematical Properties of Petri Nets

One thing that makes Petri nets interesting is that they provide a balance between modeling power and analyzability: many things one would like to know about concurrent systems can be automatically determined for Petri nets, although some of those things are very expensive to determine in the general case. Several subclasses of Petri nets have been studied that can still model interesting classes of concurrent systems, while these problems become easier.

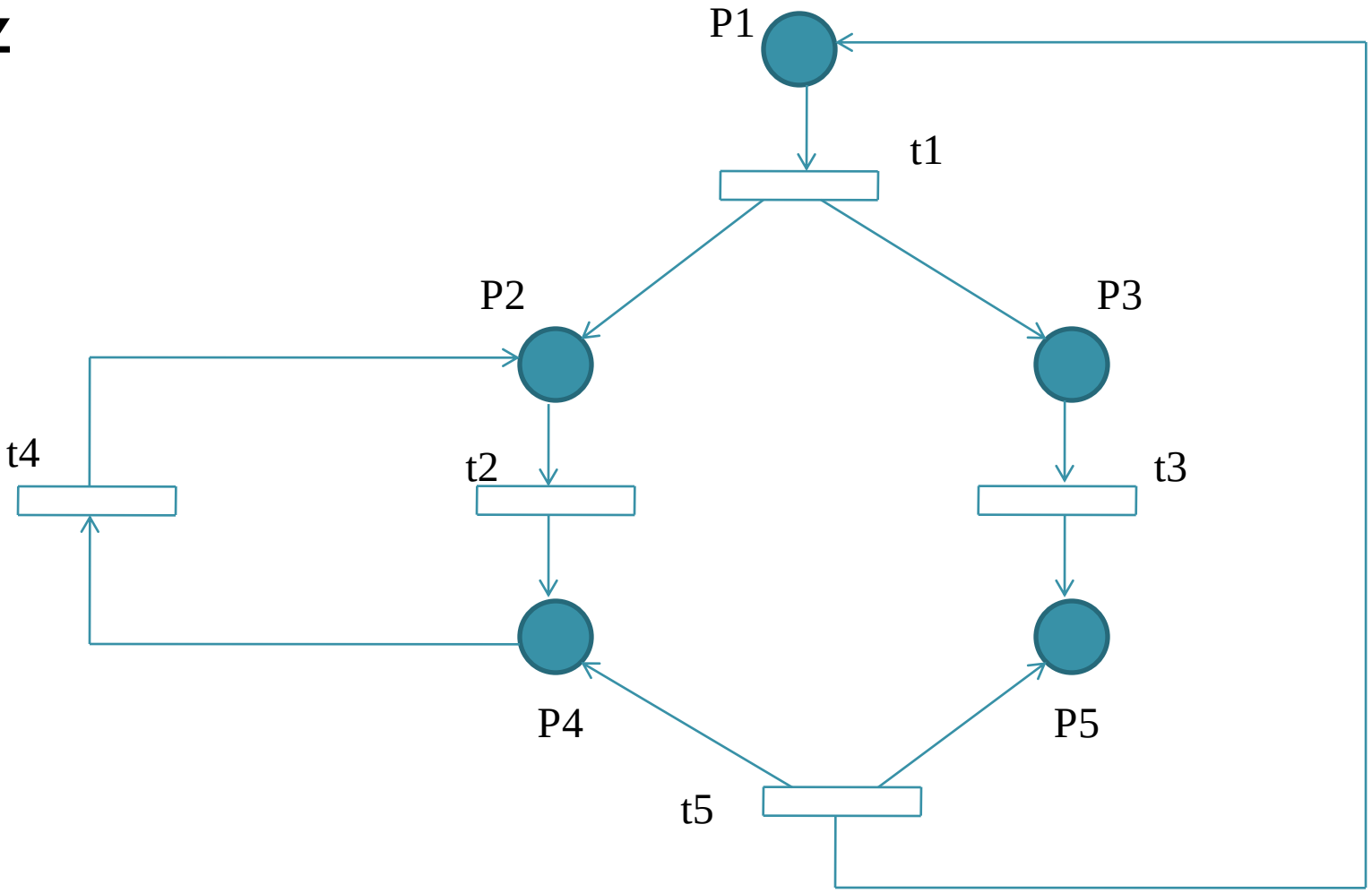
An overview of such decision problems, with decidability and complexity results for Petri nets and some subclasses, can be found in Esparza and Nielsen (1995).

For example, if in the net *N*, both places are assigned capacity 2, we obtain a Petri net with place capacities, say *N2*; its reachability graph is displayed on the right.

A two-bounded Petri net, obtained by extending *N* with "counter-places".

Alternatively, places can be made bounded by extending the net. To be exact, a place can be made *k*-bounded by adding a "counter-place" with flow opposite to that of the place, and adding tokens to make the total in both places *k*.

Z



$P=\{P_1,P_2,P_3,P_4,P_5\}$

$T=\{t_1,t_2\ ,t_3\ ,t_4\ ,t_5\}$

$I\ (t_1) =\{P_1\}O\ (t_1) = \{P_1,P_3\}$

$I\ (t_2) =\{P_2\}O\ (t_2) = \{P_4\}$

$I\ (t_3) =\{P_3\}O\ (t_2) = \{P_5\}$

$I\ (t_4) =\{P_4\}O\ (t_4) = \{P_2\}$

$I\ (t_5) =\{P_4,P_5\}O\ (t_5) = \{P_1\}$

$M_1 = (1,0,0,0,0)$

Mathematical Logic

Mathematical logic (also known as **symbolic logic**) is a subfield of mathematics with close connections to foundations of mathematics, theoretical computer science and philosophical logic.The field includes both the mathematical study of logic and the applications of formal logic to other areas of mathematics. The unifying themes in mathematical logic include the study of the expressive power of formal systems and the deductive power of formal proof systems.

- Mathematical logic is often divided into the fields of set theory, model theory, recursion theory, and proof theory. These areas share basic results on logic, particularly first-order logic, and definability. In computer science (particularly in the ACM Classification) mathematical logic encompasses additional topics not detailed in this article; see logic in computer science for those.

Since its inception, mathematical logic has contributed to, and has been motivated by, the study of foundations of mathematics. This study began in the late 19th century with the development of axiomatic frameworks for geometry, arithmetic, and analysis. In the early 20th century it was shaped by David Hilbert's program to prove the consistency of foundational theories.

Operational Timelines

Operational timelines are an ordered sequence of steps required to execute a system function. They are used referred to as operational scenario's, Operational procedures and operational flows.

The most effective time to formulate these timelines is during the development of software requirements. It is a very effective tool for the timely development of requirements and design specification.

Preparation of operational timelines involves:-

1) Understanding operational environment
2) Knowledge of the entire system
3) Imagination
4) Assistance of systems operators

Decision Support Tools

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.

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