

CS 532 Software Design

Instructor:

Frederick T. Sheldon

Computer Science Department

The University of Colorado at Colorado Springs

Bringing Design to Software

AGENDA

Course Overview

- System architecture, software architecture and software Design
- Software design axioms and principles

Role of Software Design

- Nature of the design process and the software design process
- Design in the software development process
- Design qualities
- Expressing ideas about a design

Design Representations

- Multiple views of the same systems
- The evolution of design practice

SOFTWARE ARCHITECTURE AND DESIGN

System architecture

- Highest level (proposed) system configuration including software and hardware (*requirements can be functional and/or non-functional*)
- Analysis, interface (components and human)
- Concepts of operations and analysis of mission scenarios

Software architecture

- High level partitioning of software into subsystems and their specification
- Includes SEE/CRS (languages, standards, configuration management, traceability, etc.)

Software Design

- Broadly, all phases of the Software Life Cycle prior to code
- Includes SW architecture, *more narrowly, those interactive design activities up to a detail which allows coding to commence in the target language*

WHY FOCUS ON DESIGN ... ?

Design errors are frequent

- 50% tractable in large systems to design error

Undetected design errors are costly

- New requirements emerge and cause the design to change and evolve

Initial design influences a system through its entire life cycle

- For Example,... Buick's touch display console

Other Problem specific factors

SOFTWARE CRISIS CAUSES AND CURES

Problem: (Root causes to the problems of complexity and cost . . .)

- Change is inevitable
- Ambiguities (attributable to natural language) in the agreed-upon specification
- New requirements uncovered later (than expected)
- Modest changes of requirements (in function, performance, etc.) often require major expensive changes in design
- Lack of conceptual integrity¹
- The SW Dev. Process is insensitive to the need to maintain intellectual control

Solutions

- Formal methods
- Use of information hiding and/or structured design
- Encapsulate data with operations using the ideas promoted in state-machines, abstract data typing, objects /Ada packages and object-based designs
- Specialize abstractions for handling parallel distributed systems
- Reinforce the notion malleability (see slides on design principles)

¹ Products often exhibit the absence of conceptual integrity. End users and maintainers are affected.

WHAT IS A MODEL?

A model, like an abstraction, highlights what's important for some investigation and suppresses what is irrelevant/superfluous.

- Inherent qualities/properties of a class of things
- Unlike an abstraction a model has form (structure essential to the class) and simplicity so as to facilitate study/analysis
- Benefits: intelligence amplifiers, detect problems and possible solutions (trade-offs) and the structure usually suggest submits/separation of concerns

WHAT IS A ABSTRACTION?

An abstraction is like a specification. Its a symbol (or expression) which represents the inherent qualities/properties of a class of “things.” Not including incidental qualities/aspects....

- A given abstraction always corresponds to many possible things (e.g., think of instances of the abstraction)
- Provides a medium for effective communication at a higher (language-independent) level **rather than** at a language level where different language paradigms contribute to loss effectiveness (e.g., a noisy low - level means of expression)

The guidelines for abstractions

- Express what must be achieved & suppress details regarding how
- Balance expressiveness with precision
- Review the abstraction with concerned parties (also, sometimes a independent review of the abstractions is highly desirable)
- Standardize the recording process helps to enforce conceptual integrity

STIMULUS-RESPONSE MECHANISM

- Actions are atomic (instantaneous/ discrete)
- No observable intermediate states, nor is an action, once starts affected by any external conditions
- Mechanism cycles

DOMINANT



ACTIVE

- Benefit : behavior specification and final results of execution output of time ordered sequence of steps used on design

TWO TYPES OF ABSTRACTIONS

Process: single action statements/clauses that summarize the net effect of a set of actions which would be described by a series of statements in the target language (page 41 & 42 of Witt for an example)

Data: represents a set of data elements and operations performed on the elements (see page 43 of Witt). Data has two components:

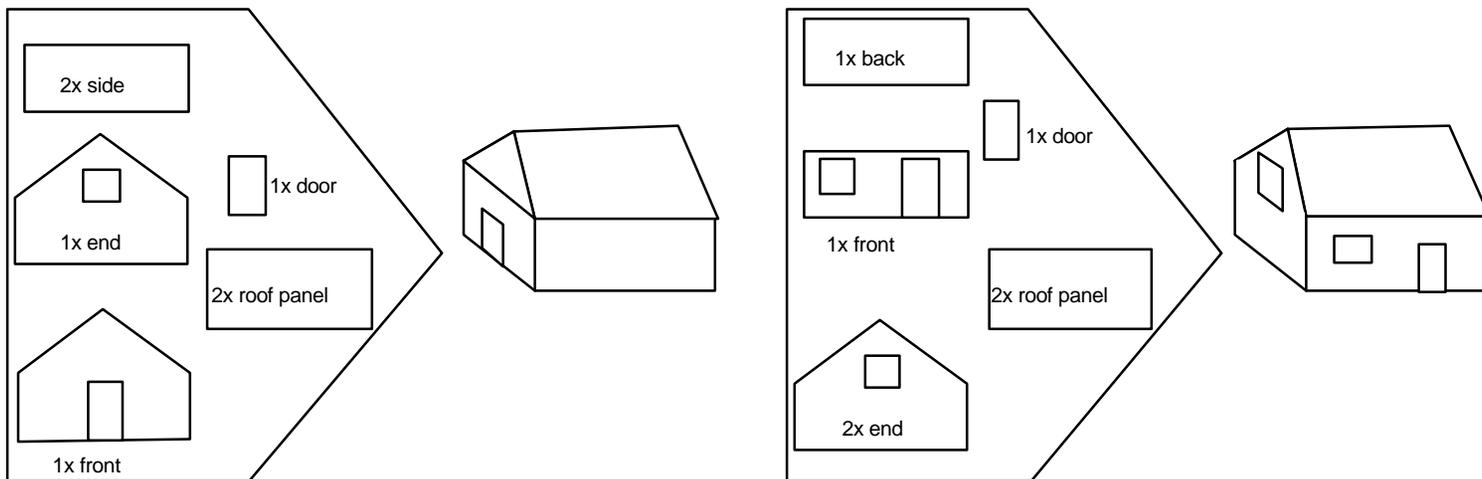
- *Abstract data type:* template defining data and its legal outputs (e.g., records, create, utilize, query, delete)
- *Abstract data object:* a specific named instance.....
 - **Public:** objects are visible /available to any client via to define operation
 - **Private:** each client holds the entire object or each invocation of a operation requires a key (e.g., fare card/ATM card w/pin)

WHAT IS DESIGN?

- **Software design:** (1) The process of defining the software architecture, components, modules, interfaces, test approach, and data for a software system to satisfy specified requirements. (2) The result of the design process. (IEEE Std Def.)
- **Design analysis:** (1) The evaluation of a **design** to determine correctness with respect to stated **requirements**, conformance to design standards, **system efficiency**, and other criteria. (2) The evaluation of alternative design approaches.
 - ✧ The fundamental problem is that designers are obliged to use current information to predict a future state that will not come about unless their predictions are correct. The final outcome of designing has to be assumed before the means of achieving it can be explored: The designers have to work backwards in time from an assumed effect upon the world to the beginning of a chain of events that will bring the effect about.
 - ✧ The design process is very different from that of the “analytical” technique that lies at the root of the scientific approach to problem-solving.
- System Architecture - totality of high-level system design HW & SW, requirements analysis, performance predictions, communications, computer-human interface.
- Software Architecture - high-level partitioning of software into major subsystems and the specification of such.
- Software Design - broadly it includes all phases of development, more narrowly its the iterative design activity of decomposing and refining the software components

OBJECTIVES FOR THE DESIGN ACTIVITY

- A blueprint
 - * Specify the best solution to the problem
 - * Describe how the solution is to be organized
 - * Produce the plans necessary for software production to proceed
 - * Assembling sheds from standard units.

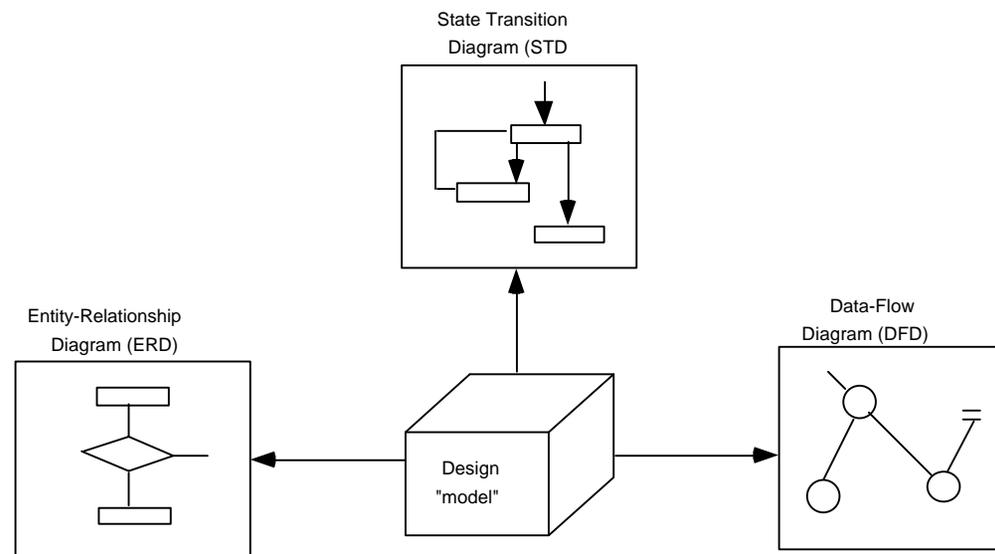


PLANS PROVIDE THE DETAIL

- Main task is to produce the plans necessary for software production to proceed
 - ✱ The static structure of the system, including any subprograms to be used and their hierarchy.
 - ✱ Any data objects to be used in the system.
 - ✱ The algorithms to be used.
 - ✱ The packaging of the system, in terms of how the components are grouped in compilation units.
 - ✱ Interactions between components, including the form these should take, and the nature of any casual links.

SOFTWARE PLANS: A VARIETY OF REPRESENTATIONS FORMS

- Each form provides a different form *view* of a system:
 - ✱ The concept of tolerance (used in classical engineering) is perhaps less important (except with respect to timing synchronization) since software components must fit rather precisely.
 - ✱ The design must model and describe. . .
 - 1) Structure
 - 2) Function
 - 3) Behavior



DESIGN AS A PROBLEM SOLVING PROCESS

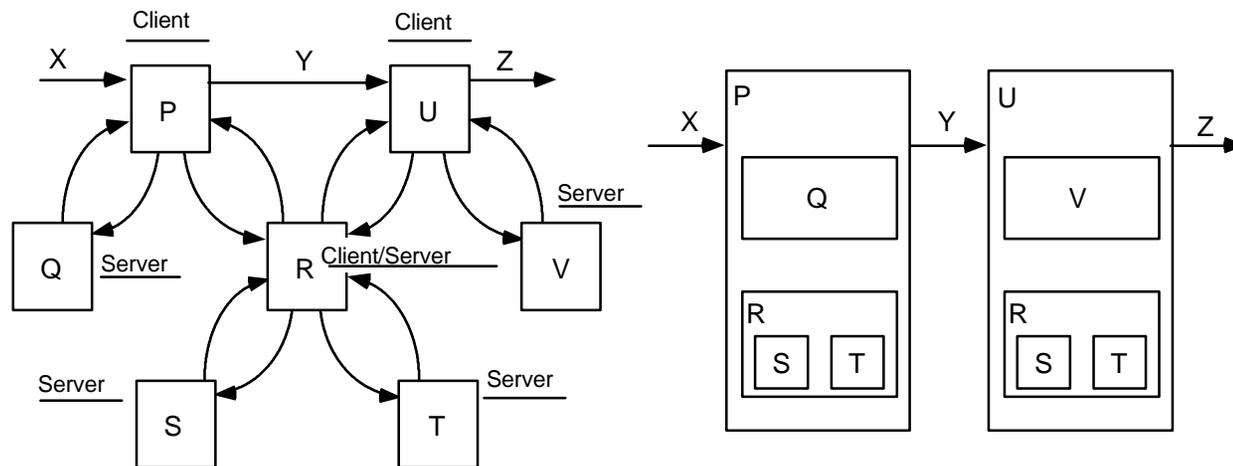
- Evaluate options
- Making choices using decision criteria
- Trade-offs:
 - ✱ Size
 - ✱ Speed
 - ✱ Ease of implementation / adaptation
 - ✱ Other problem specific factors

DESIGN AS A *WICKED* PROBLEM

- A problem whose form is such that a solution for one of its aspects reveals an even more complex problem beneath.
- The design process lacks any analytical form, with one important consequence being that there may well be a number of acceptable solutions!
- Properties:
 - ✱ There is no definitive formulation of a wicked problem.
 - ✱ Wicked problems have no stopping rule.
 - ✱ Solutions to wicked problems are not true or false, but good or bad.
 - ✱ Every wicked problem can be considered to be a symptom of another problem.

FOUR FUNDAMENTAL *UNDERLYING* AXIOMS

- The Axiom of Separation of Concerns
 - ✱ A complex problem can best be solved by initially devising and intermediate solution expressed in terms of simpler independent problems.
- The Axiom of Comprehension
 - ✱ The mind cannot easily manipulate more than about seven things at a time.
- The Axiom of Translation
 - ✱ Design correctness is unaffected by movement between equivalent contexts.



- The Axiom of Transformation²
 - ✱ Design correctness is unaffected by replacement of equivalent components.

² A high-level design unit may make references to lower-level unit specifications, but how those specifications are fulfilled is immaterial (transparent) to the higher-level design.

FIVE PRINCIPLES OF DESIGN³

Modular Designs: can be achieved by dividing large aggregates of components into units having loose inter-unit coupling and high internal cohesion, by abstracting each unit's behavior so that its collective purpose can be known, by recording each unit's interface so that it can be employed, and by hiding its design.

Portable Designs: can be achieved by employing abstract context interfaces.

Malleable Designs: can be achieved with designs that model the end-user's view of the external environment.

Intellectual Control: can be achieved by recording designs (after developing a design strategy) as hierarchies of increasingly detailed abstractions.

Conceptual Integrity: can be achieved by uniform application of a limited number of design forms.

³ Taken together, these form the basis and motivation for creating and employing a uniform set of design models, capable of being utilized throughout the architecture and design of complex systems.

PRINCIPLE OF MODULAR DESIGNS

- Designs should be modular
 - 1) Easily replaceable and self-contained assemblies of elementary parts which can limit the effect of design changes
 - 2) The high-level design (initial architecture) should be partitioned into independent activities
- Division into modular units
 - 1) A good design is characterized by loose coupling between units & high cohesion within each unit.
 - 2) Consider the problem in home building (pre-hung door).
- Public specification and hidden designs (Specification/Design Units)
 - 1) Behavior - a process abstraction summarizing the data transformations that will be performed.
 - 2) Interface - the name of the process and how it may be utilized (IN and OUT parameters)
 - 3) The design part of an SDU encapsulates all the details of implementation required to fulfill the specification.

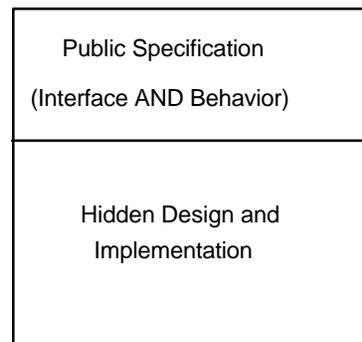
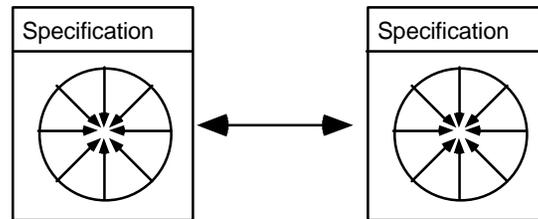
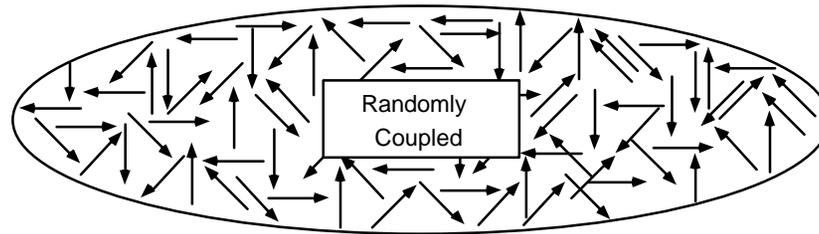
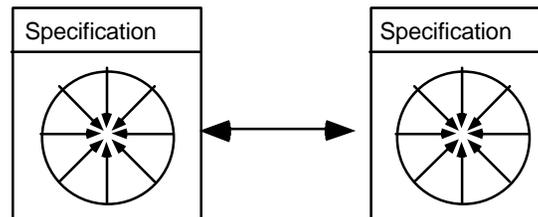


ILLUSTRATION OF COHESION AND COUPLING



Cohesive, Loosely Coupled



REVIEW: COHESION

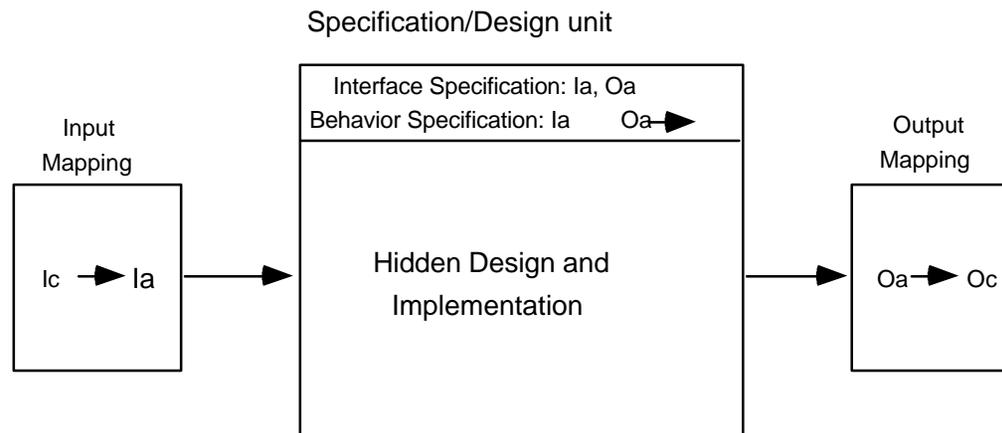
- Cohesion is a measure of how closely the parts of a component relate to each other.
 - 1) Coincidental cohesion - unrelated parts bundled together!
 - 2) Logical association - related components bundled together (e.g., input and error handling).
 - 3) Temporal cohesion - all elements are activated at a single time.
 - 4) Procedural cohesion - elements in a component make up a single control sequence.
 - 5) Communication cohesion - All elements of a component operate on the same input -> output.
 - 6) Sequential cohesion - Output from one element in the component serves as input for another element.
 - 7) Functional cohesion - each component part is necessary for the execution of a single function.

REVIEW: COUPLING AND MAINTAINABILITY

- Coupling is a measure of the strength of component interconnections. Designers should aim to produce strongly cohesive and weakly coupled design.
 - 1) Tightly coupled modules use shared variables or exchange control information (common and control coupling).
 - 2) Loose coupling is achieved by ensuring that details of the data representation are held within a component.
 - 3) Component interface with other components through a parameter list.
 - 4) If shared information is necessary, the sharing should be limited to those components which need access to the information.
 - 5) Globally accessible information should be avoided when ever possible.
- Maintainability is an important design quality attribute. Maximizing cohesion and minimizing the coupling between modules / components makes them easier to change. Understandability and adapatability are also enhanced in this way.

PRINCIPLE OF PORTABLE DESIGNS

- Designs should be portable
 - 1) Capable of reuse in different operational environments
- Design correctness is unaffected by movement into an equivalent context ⁴
 - 1) We are concerned about the portability of modular software and software whose internal design is isolated and therefore by its very nature unaffected by the move.
- Employing abstract context interfaces:
 - 1) If the SDU's input and output interfaces are specified abstractly (I_a and O_a), then mappings can be implemented between the SDU and the concrete input and output (I_c and O_c) available from the usable new context.
 - 2) Construct the design defensively, in a way that facilitates a later mapping between the interface requirements of the design and the actual interface available from, and usable by, some unknown new context.



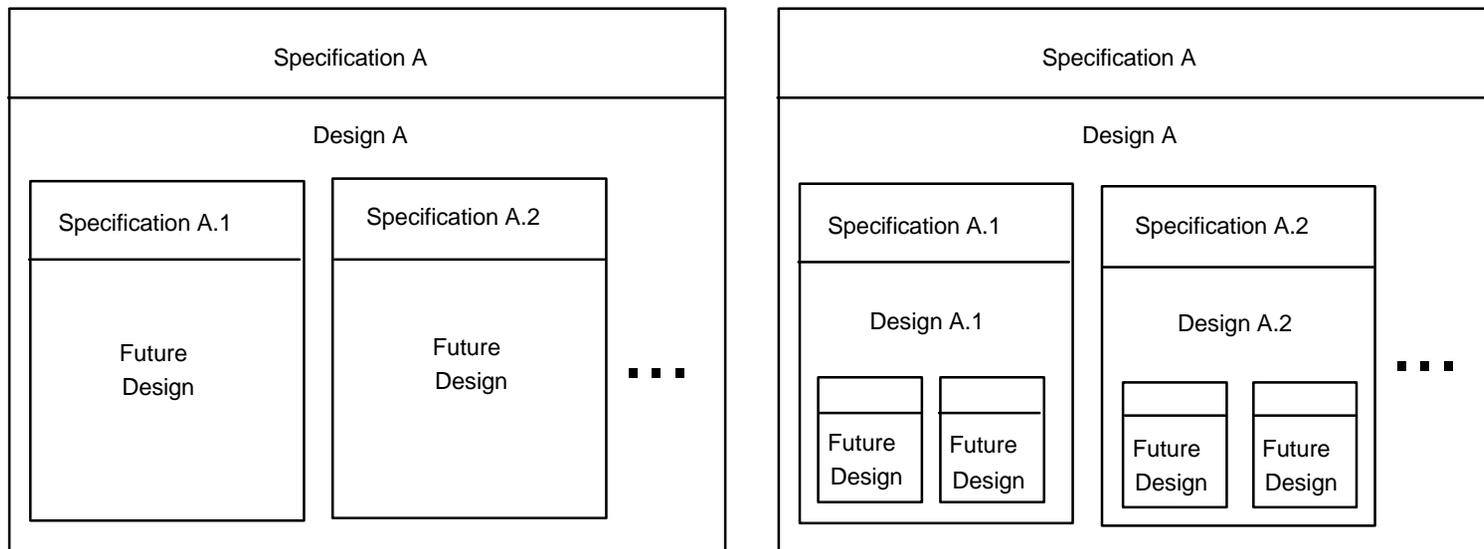
⁴ This principle is a direct consequence of the Axiom of Translation.

PRINCIPLE OF MALLEABLE DESIGNS

- This principle describes a way to increase the likelihood that a design can absorb changes in user requirements with minimal impacts.
- Malleable refers to the ability of a design to accommodate changes to behavior requirements
- Elements in our designs must correspond to elements in the end-user's view of the world.
 - 1) Modularity ensures only that if the behavior and interface specifications of a unit remain unchanged, then the unit can be replaced. This protects the bulk of the system design from changes in the representation of the systems input and output.
 - 2) If behavior requirements change, then modularity alone does not provide a sufficient shield against change.
- Modeling the user's world
 - 1) Thus, we maximize the dependencies on stable requirements and minimize the dependencies on unstable ones. This is the starting point for organizing the design.
 - 2) The uncertainties in a design should be encapsulated, both during design and over its useful lifetime, so that they may readily be changed as requirements are eventually firmed up or are altered by circumstance.

PRINCIPLE OF INTELLECTUAL CONTROL

- Design process should be under intellectual control
 - 1) Parts must interrelate
 - 2) Rationale and criticality of design choices *should be* understood
 - 3) Effect of proposed changes *must be* understood
 - 4) The design should exhibit correctness prior to implementation
- The need for hierarchy to support comprehension
 - 1) The root SDU of an evolving design contains the root specification (Specification A) and the corresponding high level design (Design A) which in turn is expressed in terms of lower-level specifications.



PRINCIPLE OF INTELLECTUAL CONTROL

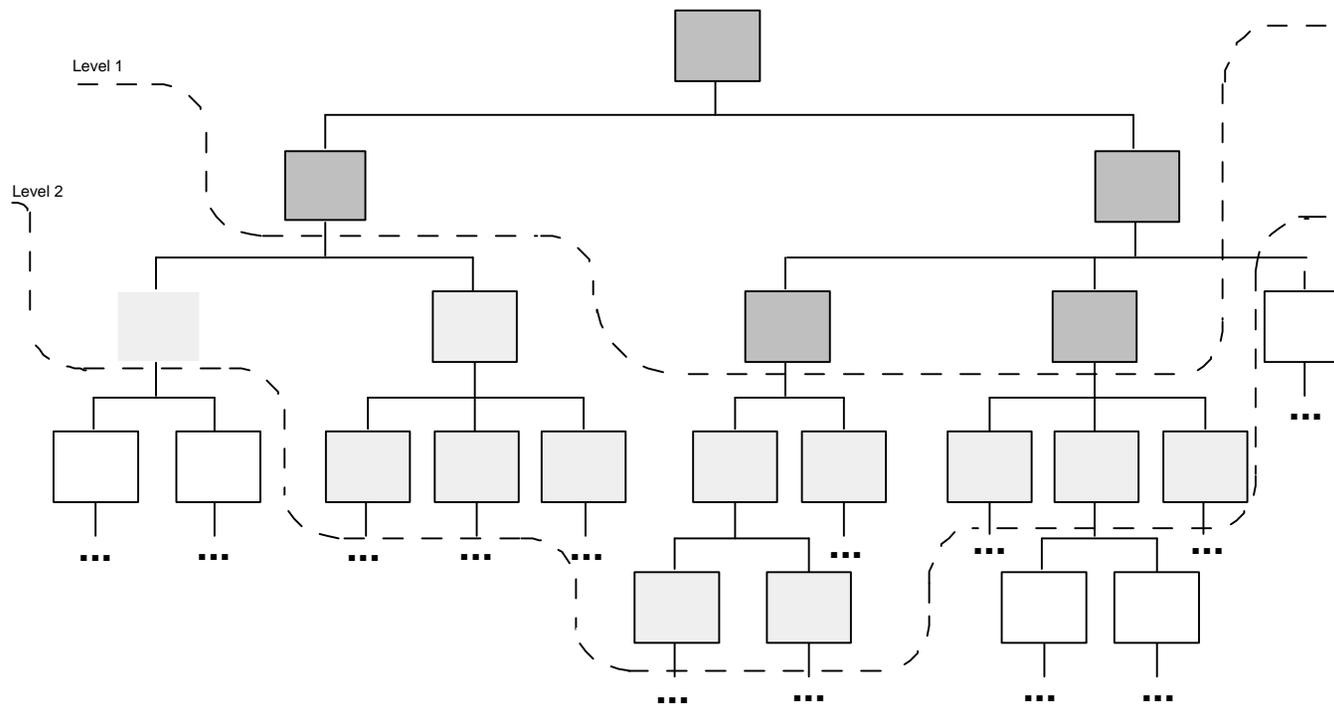
- The need to record specifications with designs ⁵
 - ✳ Design units always have two components: the summary or specification and the design itself.
 - ✳ The hierarchical structure of the presentation is obvious from the indentation style and the BEGIN/END delimiters

```
[ Get to work in the AM ] =  
  BEGIN  
    [ Get ready to Leave house ] =  
      BEGIN  
        Wake up and get out of bed;  
        Shower, brush teeth, comb hair, ...;  
        Dress for work;  
        Eat breakfast;  
      END  
    [ Drive to work ] =  
      BEGIN  
        Drive to freeway  
        Drive to work exit;  
        Drive to parking;  
        Park and enter building;  
      END  
    [ Go to the office ] =  
      BEGIN  
        Get elevator;  
        Ride elevator to office floor;  
        Walk to office;  
      END  
    END  
  END
```

⁵ Notice that (theoretically) the designer was able to focus solely on what was needed to complete the design of the root, and set aside temporarily the question of how each of the three components would be expanded.

PRINCIPLE OF INTELLECTUAL CONTROL (CONTINUED)

- The need to verify equivalence
 - ✳ The replacement of specifications by designs is a very error-prone activity in software development. Must arrive at a reasonable conviction that the design and specification are equivalent.
- The need for strategy: Planning to levels of quiescence
 - ✳ Hierarchical recording of designs facilitates comprehension, and should be stubbornly adhered to
 - ✳ A practical approach to the balance between thinking and recording is to delay recording until the design ideas stabilize at some *level of quiescence* ...



PRINCIPLE OF CONCEPTUAL INTEGRITY

- This principle describes the basis for our ability to distinguish order from chaos
 - 1) Conceptual integrity is the most important consideration in system design. It is better to have a system omit certain anomalous features and improvements, than to have one that contains many good but independent and uncoordinated ideas.
 - 2) Every part must reflect the same philosophies and the same balancing of desiderata. Every part must even use the same techniques in syntax and analogous notions in semantics. Ease of use, then, dictates unity of design and conceptual integrity.
- Requires a shared vision
 - 1) Designer(s) must be interchangeable in their ability to add details once the “vision” has been articulated.
 - 2) The “vision” should be one that balances capabilities supplied to users with ease of use and maintainability of those capabilities
 - 3) The “vision” should be one that balances the design concept with the implementation, testing and tool building associated with its construction.
- Design models are the principal contributor to conceptual integrity
 - 1) Other design forms (representation forms) play a role such as standards and common approaches for addressing common design situations
- How can we ensure uniform application of the models, standards, and approaches?⁶

⁶ a) Products exhibit harmony, symmetry and predictability; b) The system should appear to reflect the mind of a single person and to faithfully adhere to a single concept; c) There should be no surprises for its user or its maintainer; d) Knowledge gained in one use or change should be immediately transferable to the next

SOFTWARE DESIGN ABSTRACTIONS AND MODELS

- Abstraction
 - ✧ A symbol (often an expression) that represents the inherent qualities or properties of a class of things, rather than those qualities of the things deemed incidental for some given purpose.
- Process abstraction
 - ✧ Ideally are single-action statements or single-action clauses that summarize the net effect of a set of actions which can only be described by a series of statements in the target language.
- Data abstraction
 - ✧ Represent a set of data elements and operations that can be performed on those elements, neither of which are available in the target language.
- Model
 - ✧ Like an abstraction, highlights what's important for some investigation and suppresses what is irrelevant.

MODELS ASSIST FINDING DESIGN SOLUTIONS

- Intelligence amplifiers
 - ✧ Tools that help the understanding based on analysis and documentation. By constructing the model the new and unknown can be compared with old and familiar to similarities and differences.
- Help identify problem areas
 - ✧ The existence of problems becomes clear as well as possible solutions. Because models can be analyzed in some meaningful way, often yielding quantitative results, we can use them to predict problem areas and propose rules for problem avoidance; as well as hypothesize alternate solutions to trade-off the costs and benefits.
- Help in defining and capturing structure
 - ✧ Models suggest components to be included in designs, so the designs need not be reinvented each time.

THE SOFTWARE DESIGN PROCESS

- Building models
 - ✧ A design method provides the necessary set of transformations that lead from the initial model to a detailed description of the eventual solution corresponding to a particular model.
- Structuring the design process
 - ✧ **Representation part** provides a set of descriptive forms used for building models of the problem.
 - ✧ **Process part** describes transformations between representation forms.
 - ✧ **Set of heuristics** guide the activities defined in the process part for specific classes of problems.
- Constraints upon the design process
 - ✧ Problem specific and includes such things as hardware / platform, OS and programming language.
 - ✧ Process part describes transformations between representation forms.
- Recording design decisions
 - ✧ Needed for maintenance and quality assurance.
- Designing with others
 - ✧ How to split the design task among the team and integrate the individual contributions to the design.

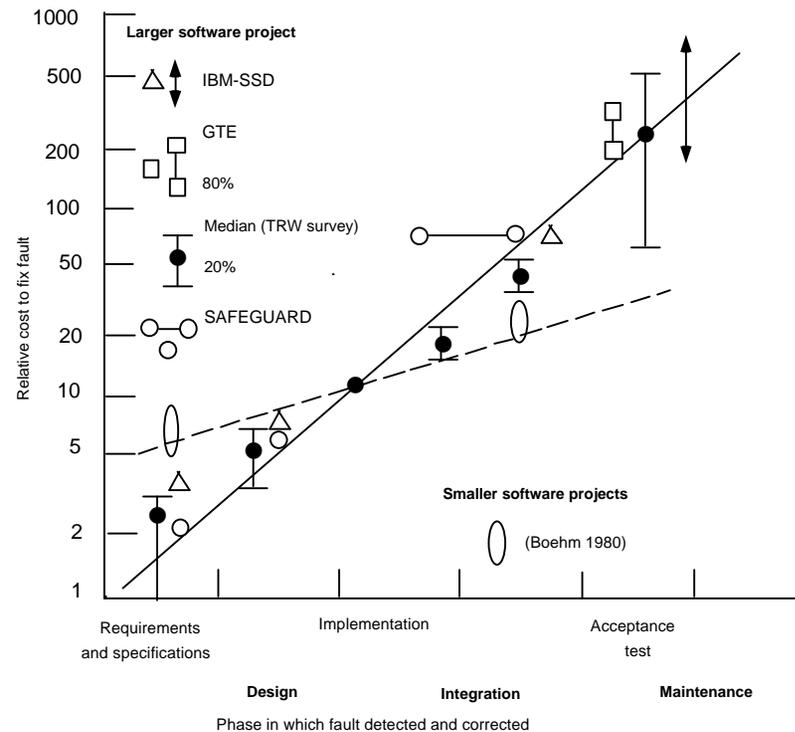
DESIGN IN THE SOFTWARE DEVELOPMENT PROCESS

- A context for design

The software process model addresses the software project questions

- What should we do next?
- How long should we continue to do it?

- Economic factors



- Software production models and their influence
- Prototyping roles and forms

DESIGN QUALITIES

- The quality concept

Software quality concepts are concerned with assessing both the static structure and the dynamic behavior of the eventual system.

- Assessing design quality

Ultimate goal is of quality must be that of fitness for purpose, although the criteria for determining whether this is achieved will be both problem-dependent and domain dependent.

- Quality attributes of the design product

While the use of abstraction is an important tool for the designer, it makes it difficult to make any direct product measurements during the design process.

- Assessing the design process

Technical design reviews can provide a valuable means of obtaining and using domain knowledge to aid with assessing the design product as well as method knowledge to aid with assessing the design process.

EXPRESSING IDEAS ABOUT DESIGN

- Representing abstract ideas

The roles of representation in capturing, explaining and checking design information.

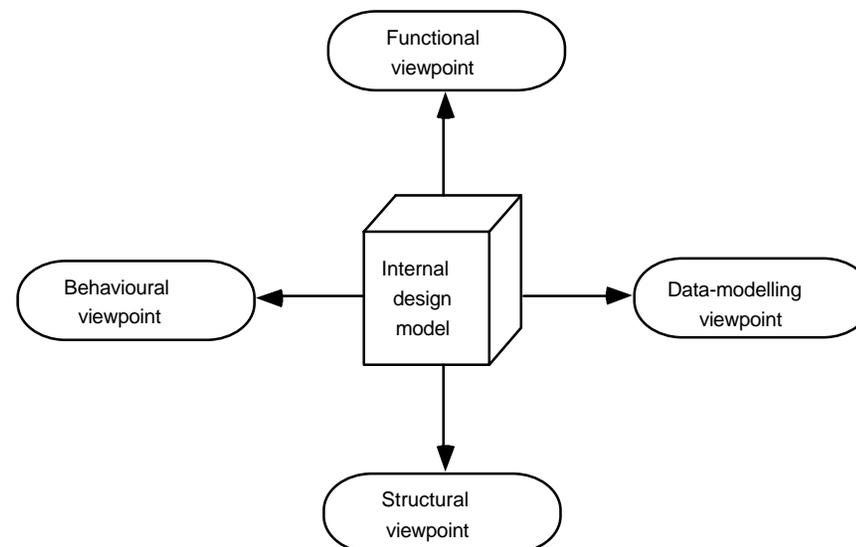
- Capturing the designer's ideas for a solution
- Explaining the designer's ideas to others (customers, implementors and managers).
- Checking for consistency and completeness in a solution.

- Design viewpoints for software

A means of capturing a particular set of design attributes, and as projected through the use of a representation

- Forms of notation

The principal classes of direct design viewpoint – the structural, behavioral, functional and data-modeling forms.



SOME DESIGN REPRESENTATIONS

- The Data-Flow Diagram
- The Entity-Relationship Diagram
- The Structure Chart
- The Structure Graph
- The Jackson Structure Diagram
- Pseudocode
- The State Transition Diagram
- The State Chart
- The Petri net

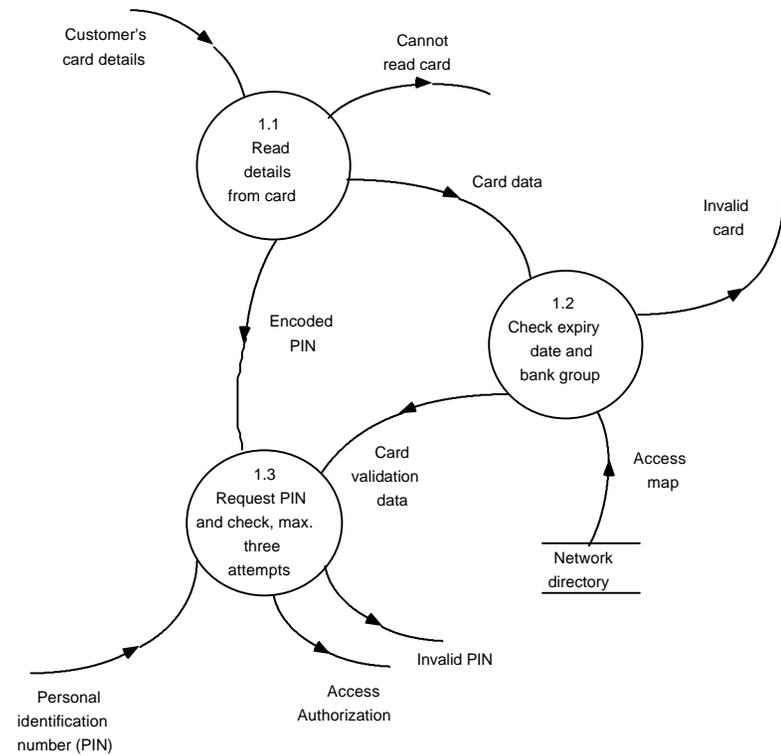
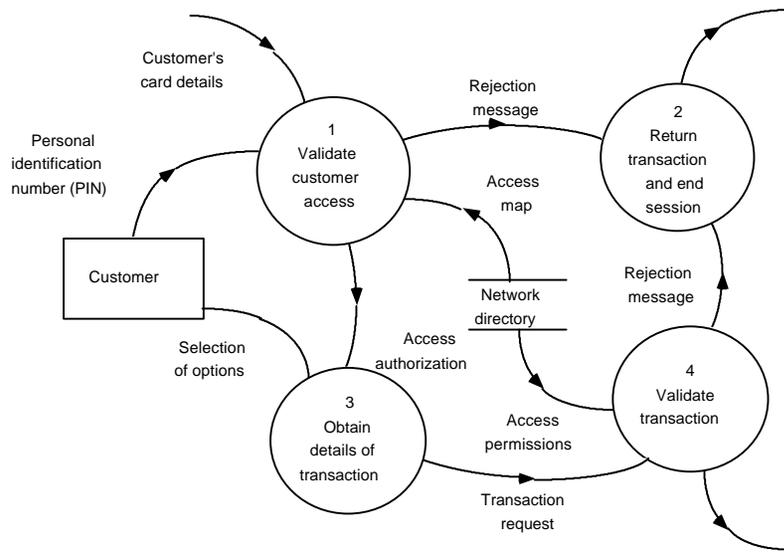
SELECTING THE CORRECT DESIGN REPRESENTATION

- Form Includes textual, diagrammatical and mathematical forms of notation.
- Viewpoint Structural, behavioral, functional and data-modeling viewpoints.
- Use In terms of the form's role during the phases of design, the type of problem domain in which it might be appropriate, and the extent to which it is used.

<i>Representation form</i>	<i>Viewpoints</i>	<i>Design attributes</i>
Data-Flow Diagram	Functional	Information flow, dependency of operation on other operations, relation with data stores
Entity-Relationship Diagram	Data modeling	Static relationships between design entities
Structure Chart	Functional and structural	Invocation hierarchy between procedures, decomposition into procedures
Structure Graph	Structural	Packaging (information-hiding), uses relationship, concurrency
Structure Diagram	Functional data modeling, behavioral	Algorithm form Sequence of data components Sequencing of actions
Pseudocode	Functional	Algorithm form
State Transition Diagram	Functional	State-machine model of an entity
StateChart	Behavioral	System-wide state model, including parallelism (orthogonality), hierarchy and abstraction
Petri Net Graph	Behavioral	Interaction between parallel threads

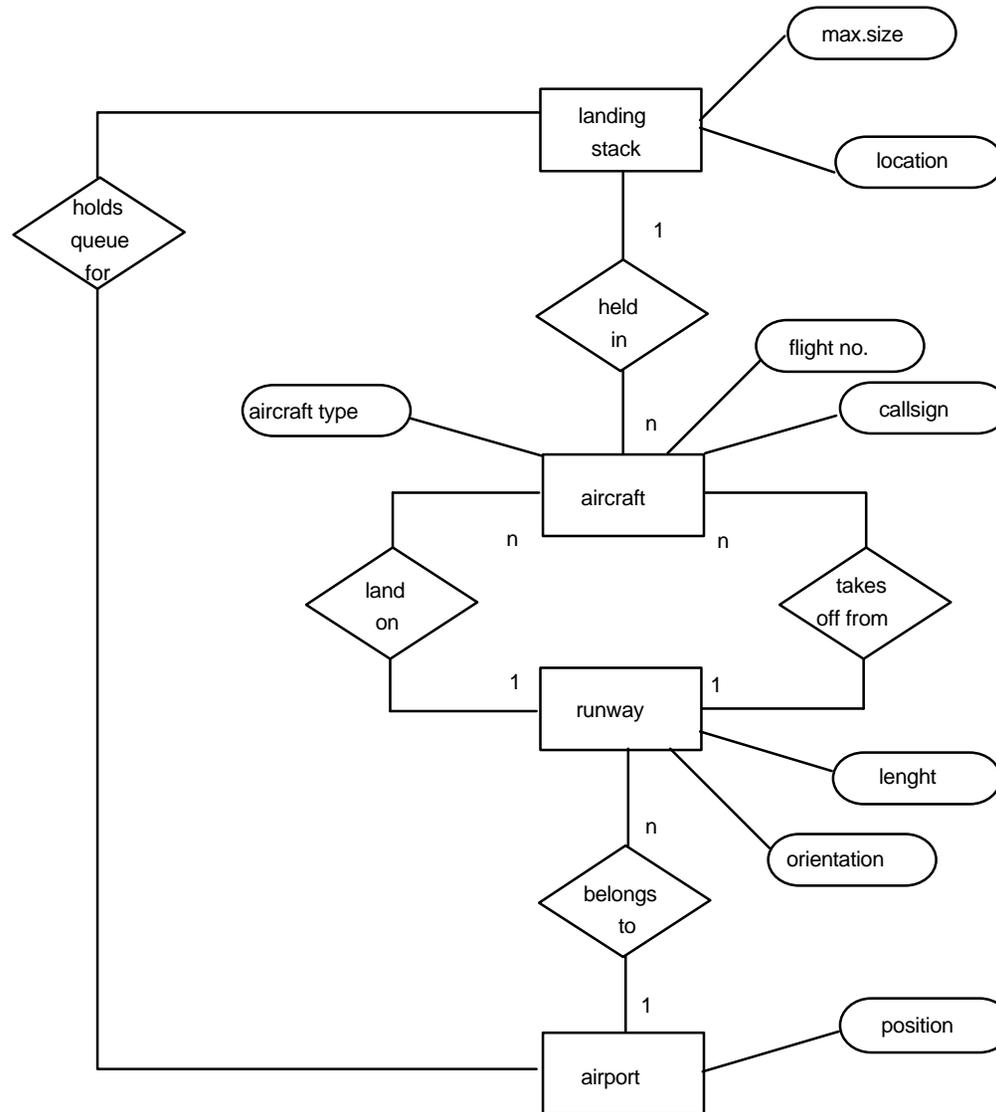
SOME DESIGN REPRESENTATIONS

The Data-Flow Diagram



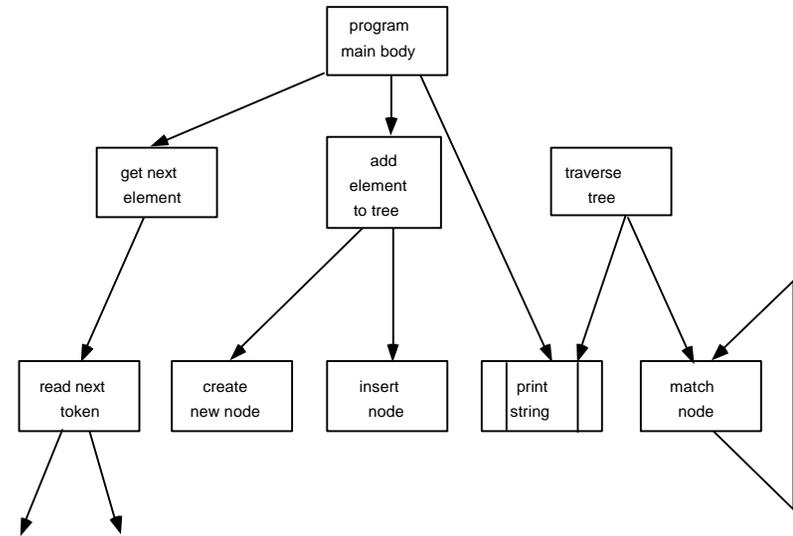
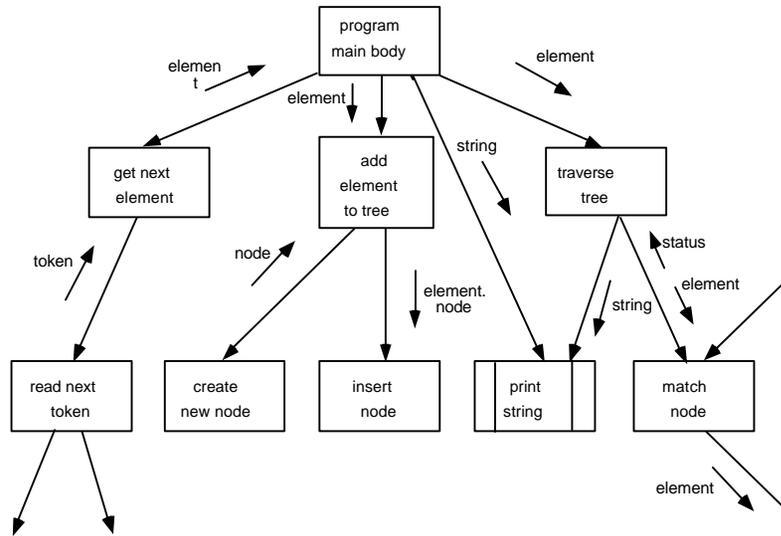
SOME DESIGN REPRESENTATIONS

The Entity-Relationship Diagram



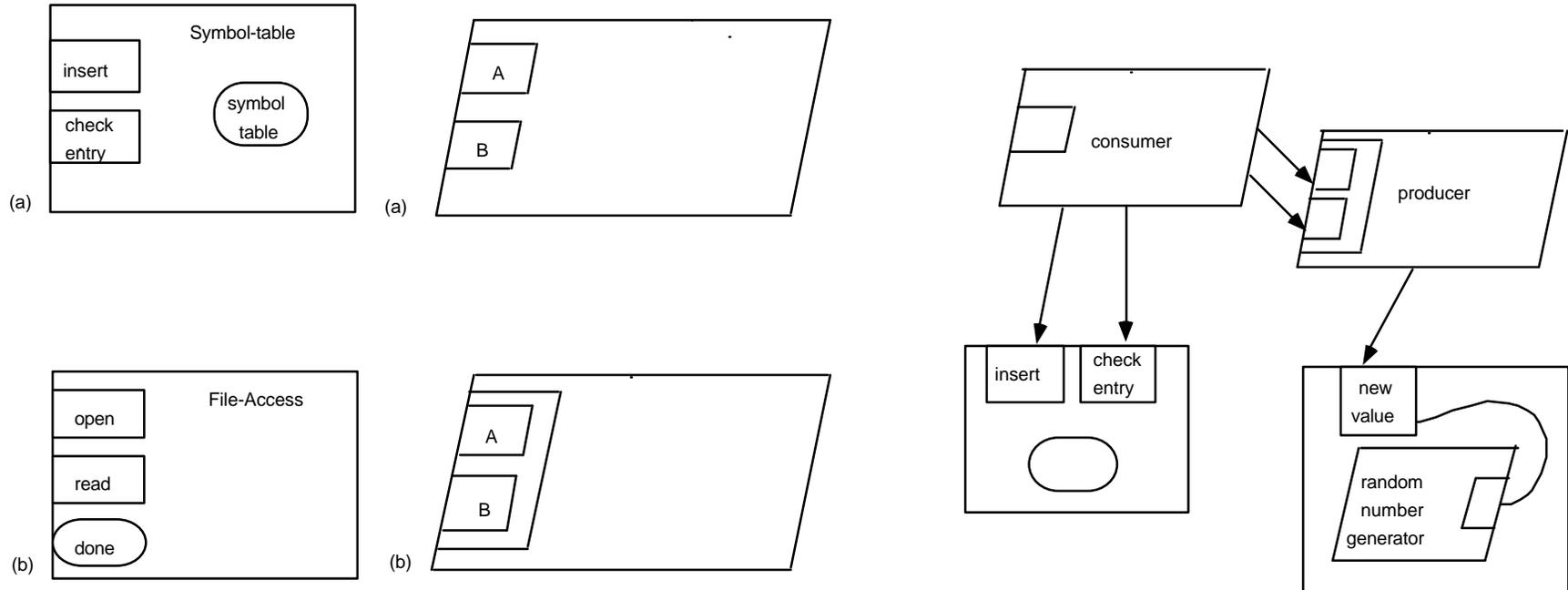
SOME DESIGN REPRESENTATIONS

The Structure Chart



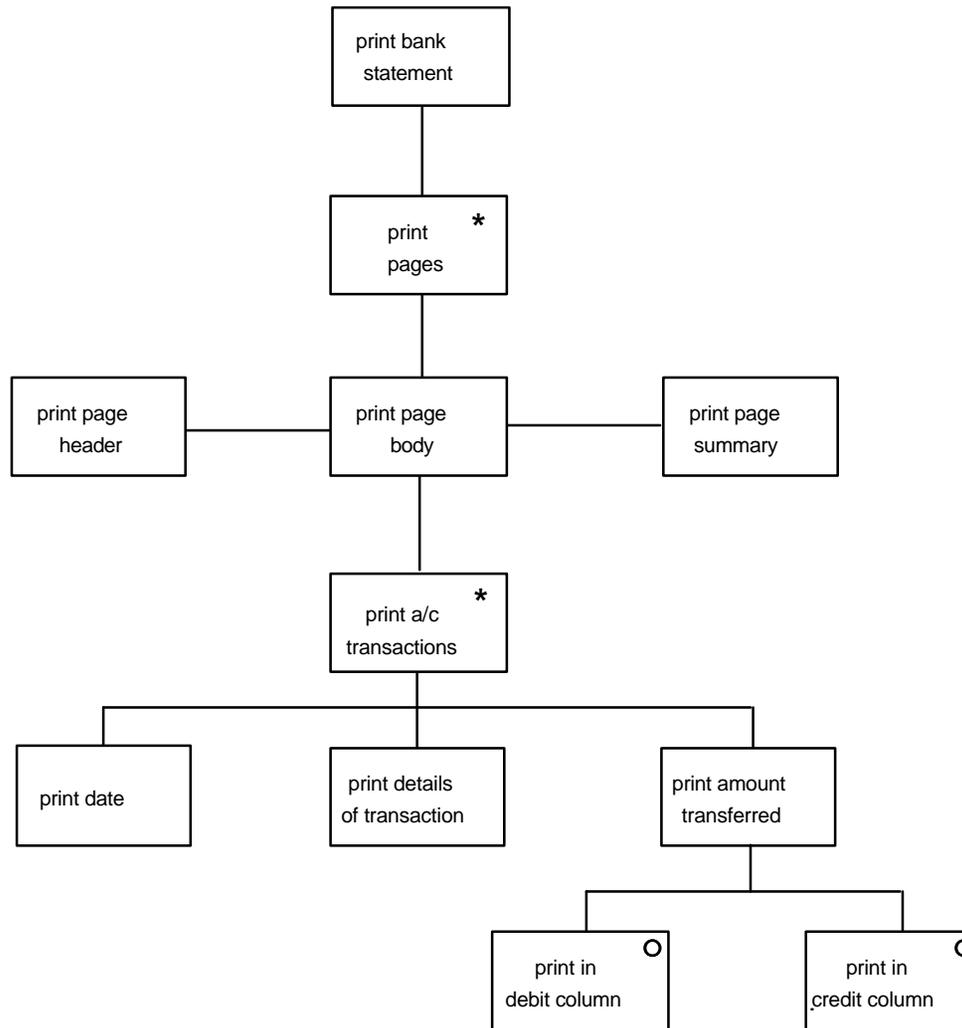
SOME DESIGN REPRESENTATIONS

The Structure Graph



SOME DESIGN REPRESENTATIONS

The Jackson Structure Diagram



SOME DESIGN REPRESENTATIONS

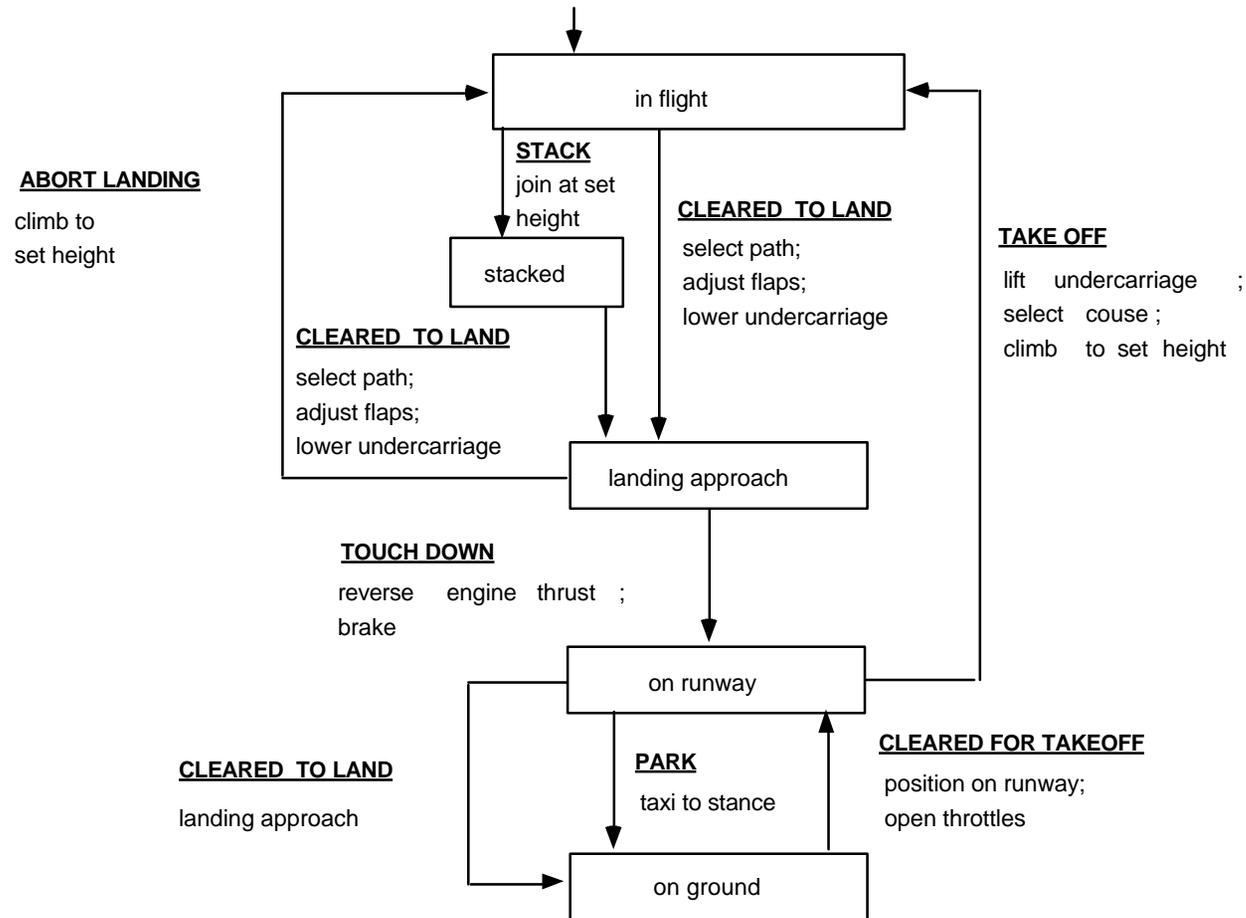
Pseudocode

boil water;
pour some water into teapot;
empty teapot;
REPEAT
 place spoonful of tea in pot
UNTIL enough tea for no. of drinkers;
REPEAT
 pour water
UNTIL enough water for no. of drinkers;

INITIALIZE line buffer;
READ first character from keyboard;
WHILE not the end of line **DO**
 IF character is terminator of a word
 THEN
 mark end of word in buffer;
 SKIP any trailing word separators
 ELSE
 copy character to buffer
 END IF ;
 READ next character;
END WHILE;

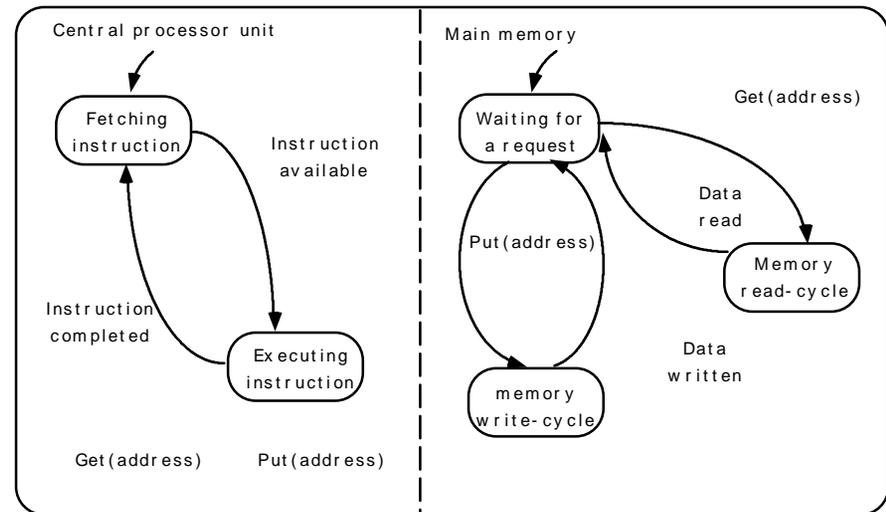
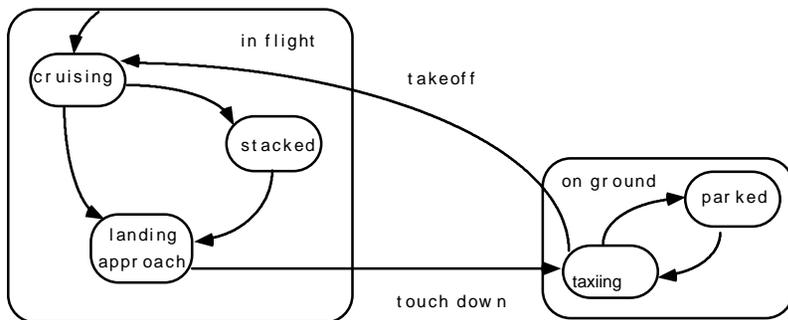
SOME DESIGN REPRESENTATIONS

The State Transition Diagram



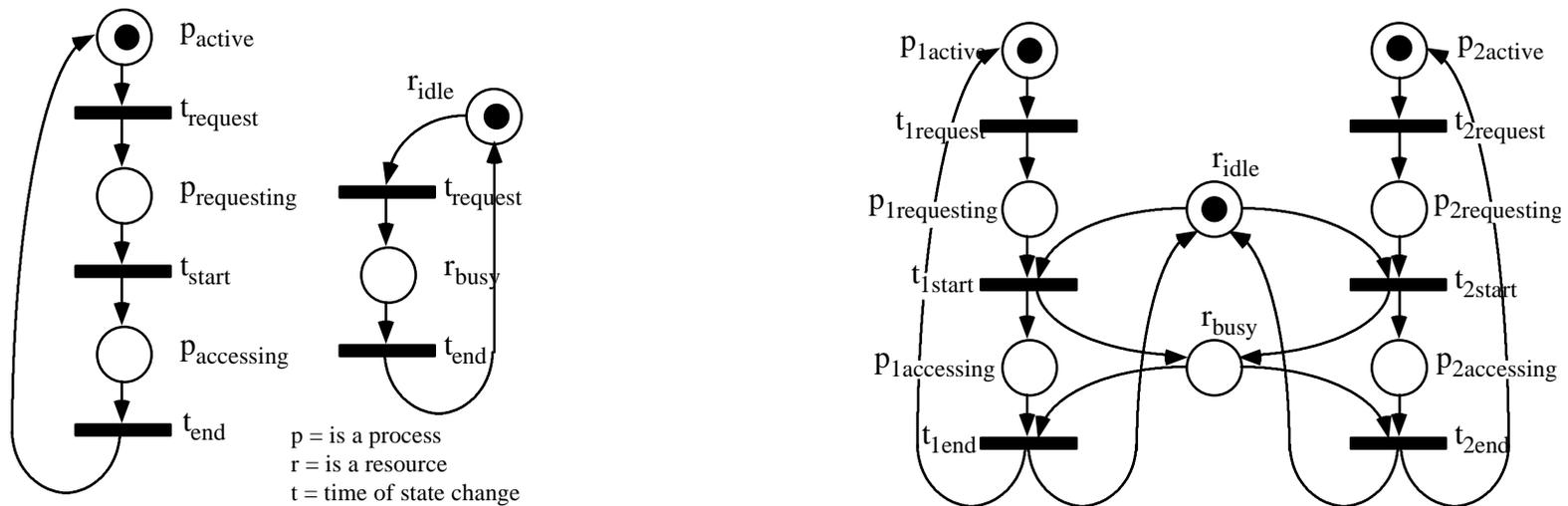
SOME DESIGN REPRESENTATIONS

The State Chart



SOME DESIGN REPRESENTATIONS

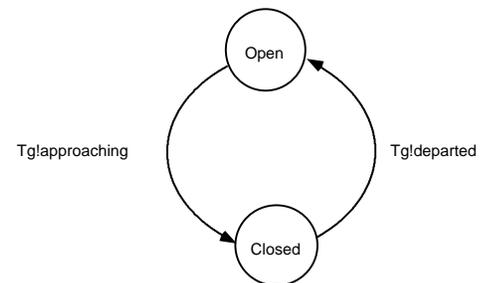
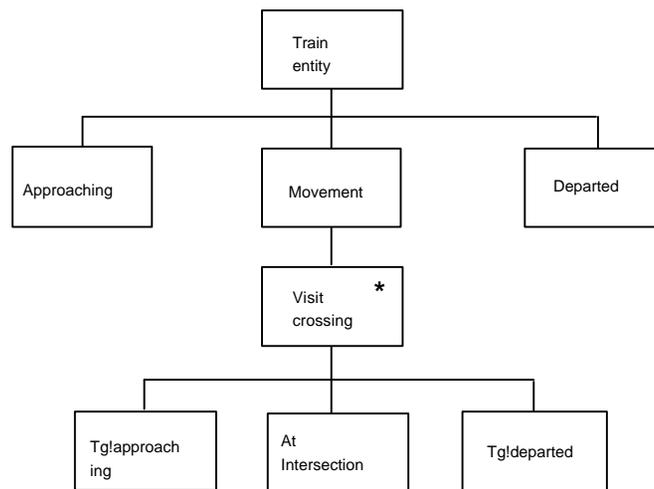
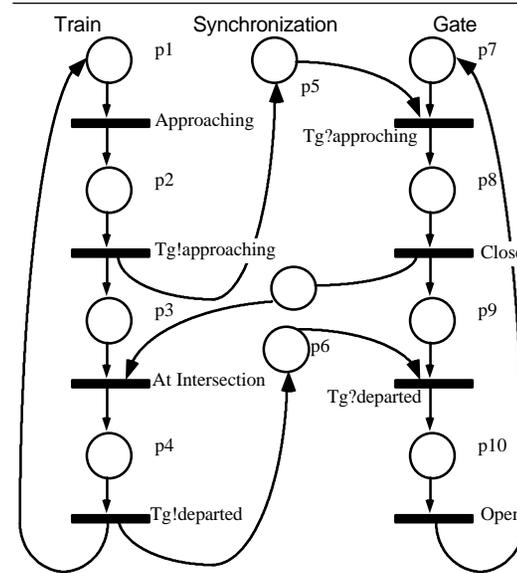
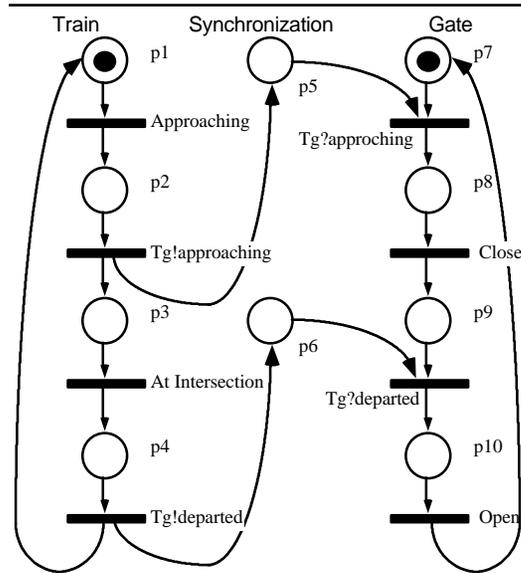
The Petri Net



One of the classical problems in computer science is the orderly access to a shared resource. This may result because a CPU must address a memory location, or a packet must be processed by a protocol. Lets consider the individual states of the resource and of its user separately. The resource can be in either an idle or a busy condition (alternately). Merge the two nets shown below via superposition of the appropriate transitions.

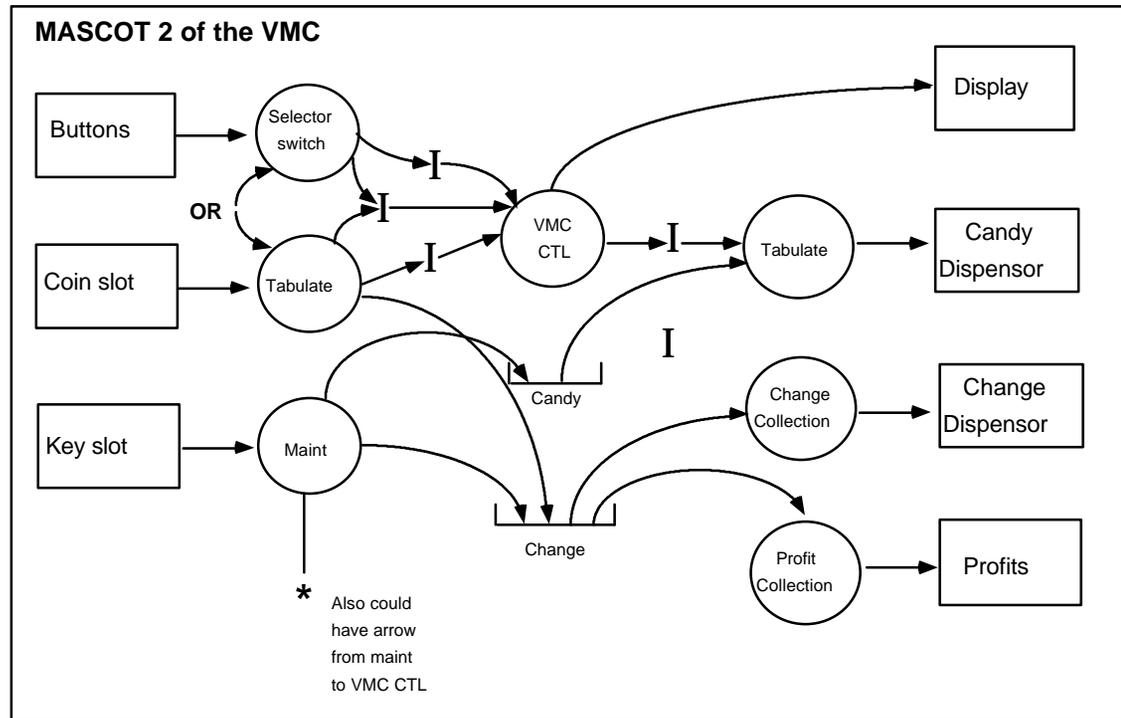
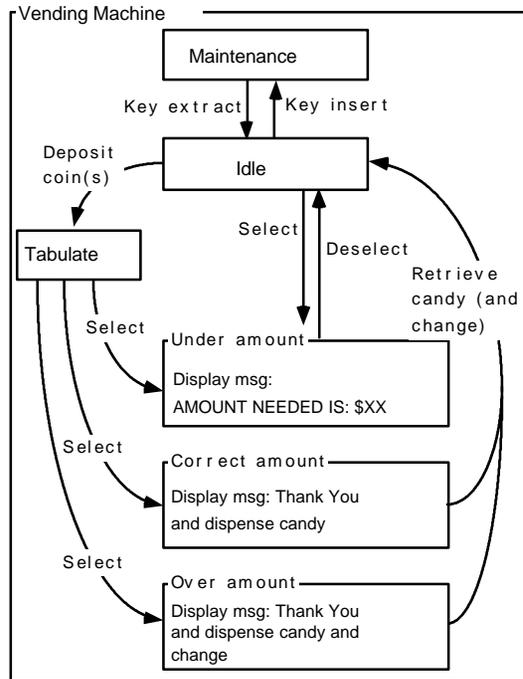
MULTIPLE VIEWS OF THE SAME SYSTEM

Train / Gate Railroad Crossing



MULTIPLE VIEWS OF THE SAME SYSTEM

Vending Machine

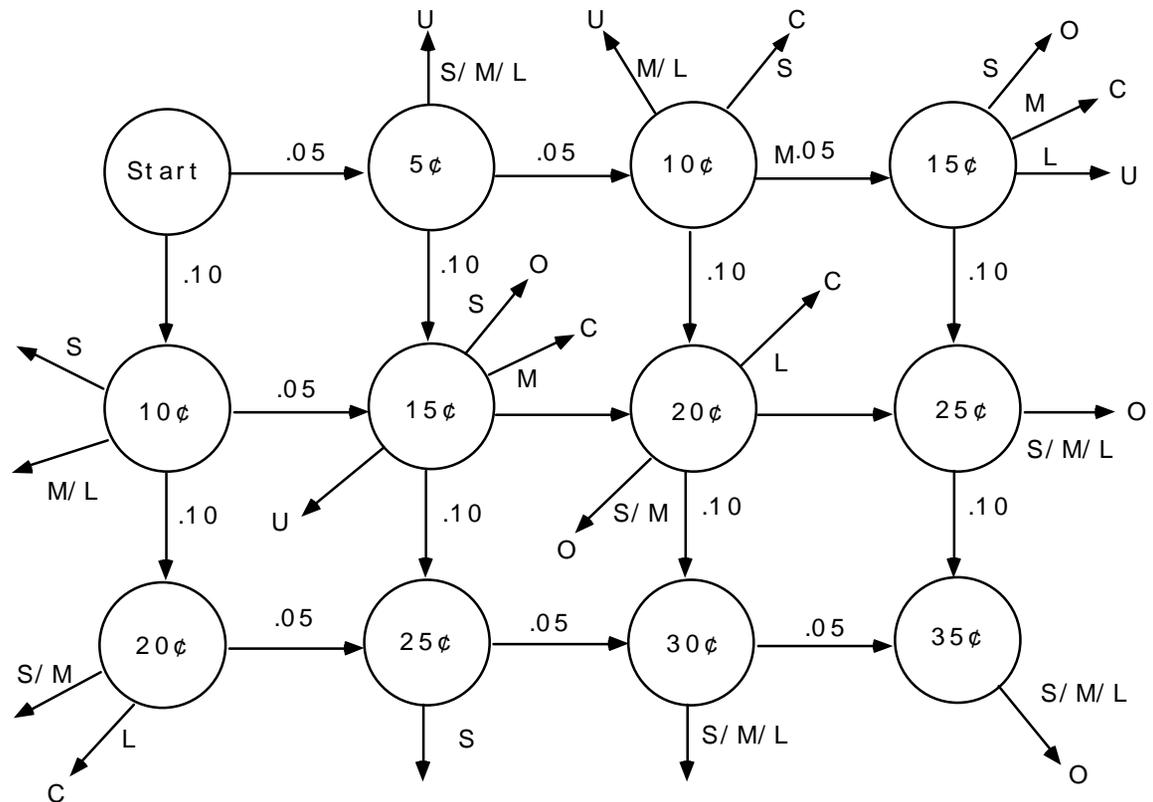
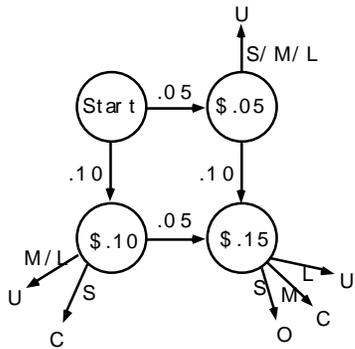


MULTIPLE VIEWS OF THE SAME SYSTEM

Vending Machine (continued)

Tabulate

U = Under amount
C = Correct amount
O = Over amount
S = Select small candy
M = Select medium candy
L = Select large candy



THE EVOLUTION OF DESIGN PRACTICES

Conclusions

- The software design process is intrinsically a complex undertaking by its nature.
- The complexity forces limitations for any design method.
- Procedures and notations can vary widely between different methods. The selection of a particular method will be based on numerous factors (organizational constraints, domain of specialized application (e.g., real-time systems)).
- Formal descriptions can provide a very powerful aid to developing design especially when issues such as consistency and verification are considered in terms of safety (or ultra-cost) critical systems. There is evidence for increasing use of such methods for the development of high-integrity systems (or those parts of such)
- Future methods may need to move away from procedural forms to achieve any significant step forward.

TITLE

•

•

•

•

•

•

•

•

•

•

•