Software Design
- Theoretical Approaches

By

Vasanth Krishnamoorthy

Sanchit Karve

Overview

- Introduction
- Features of the problems of Program Design
- Knowledge Centred Approach
- Strategy Centred Approach
- Organization Centred Approach
- Modelling the Expert
- Making Tools more suitable for Programmers
- Future Research

Introduction

- **Software design** is a process of problem solving and planning for a software solution.

- **Need for theoretical approaches**
    - Problems to be solved are “ill-defined”
    - Distinction needs to be drawn between problems of producing programs and problems of producing results.

- **Theoretical Approaches**
    - Knowledge-Centred Approach
    - Strategy-Centred Approach
    - Organization-Centred Approach

Features of the problems of Program Design

- **Program design** is mainly the framework of research into the problem solving activities.

Ill Defined Problems

- **Problem Solving Phases**
  - Understanding the problem.
  - Research and Development of the solution.
  - Coding the solutions.
- **Ill Defined problem characteristics**
  - Some specification of the problem is missing.
  - Part of the solution introduces new constraints.
  - There exist several acceptable solutions for the same problem, which cannot be evaluated using a single criterion.

Features of the problems of Program Design

Program Design Activity

- **Domain knowledge.**
  - Designers use both application (or problem) domain and the computing domain, between which they establish a mapping.
  - Programmers construct a mental model of
    - Problem and its solution in terms of application domain.
    - Problem and its solution in terms of computing terms.
  - Part of the work consists of passing info from one model to another.
  - Depending on the features of the design situation, the distance between the models will be made bigger or smaller.

Features of the problems of Program Design

- **Problems of Program Production.**

  Distinguished based on type of solution produced.

  - Problem solving situations :
    - Finding the procedure to obtain the result.
    - Obtaining the result itself.
  - In a result producing situation
    - Subject concentrates on the result, procedure is secondary.
  - In a program producing situation (harder)
    - Subject concentrates on working out a procedure.
  - Example: Task of sorting names is easier to find the result when compared to finding the procedure to do so.

Knowledge-Centred approaches

- **Identifying and formalizing the knowledge of expert programmers**

  - 3 types of knowledge that serve to distinguish experts from novice.
    - Syntactic knowledge
      - Define the syntactic and lexical elements of a language. Eg. ‘If’ condition
    - Semantic knowledge
      - Refers to the concepts that make it possible to understand what happens when a line of code is executed. eg. Notion of a variable.
    - Schematic knowledge
      - Refers to programming schemas that represent generic solutions.

Knowledge-Centred approaches

Theory of Schemas

It is the theory of organization of knowledge in memory and of the processes for making use of this knowledge.

- **Schema**
  - A schema is a data structure which represents generic concepts stored in memory (knowledge structure).
  - A structure of variables to which is associated a set of possible values.
- **Solution Plan**
  - A sequence of actions in a program which will achieve the goal of the program.
  - For experts, a special case or an instance of a program schema

Knowledge-Centred approaches

Programming Schemas

- **Elementary programming schemas** represent knowledge about control structures and variables.
- **Algorithmic schemas** represent knowledge about structure of algorithms.

Classification based on degree of **dependence on programming language**.

- **Tactical and strategic schemas** which are independent of programming language.
- **Implementation schemas** which are dependent on particular programming language.

The notion of 'focus' or 'focal line' is that part of the schema that directly implements its goal. Eg. Incrementing the counter in a 'count' loop.

Knowledge-Centred approaches

Other types of Schema

Structural Schema

- Programming schemas which are rich in knowledge and content.

  Eg. In the studies of understanding text and grammar.

- Programmers ability to write or understand programs depends on their familiarity with this schema.

Domain Specific Schema

- Developed by experts familiar with a problem domain.
- They are knowledge schemas representing their knowledge of certain types of problems.

  Eg. An expert in invoicing and sales application will have a schema for discount structures.

Knowledge-Centred approaches

Rules of Discourse

- Control the construction of programs and instantiation of schemas during design.
  - Experts retrieve suitable programming schemas from memory and instantiate them according to the particular problem they are solving.

    Eg. The name of a variable must reflect its function.
  - In professional software engineering, rules of discourse are usually formalized in to coding standards.

Limitations of Schemas

- Understanding a program consists, in part of activating schemas in memory and then inferring information from it.
  - This approach is limited because it takes little account of other processes found in these activities which are bottom up and more constructive.

Strategy-Centred approaches

The expert is characterized not only by more abstract, hierarchically organized knowledge but also by a broader range of more versatile strategies.

Design strategies are chosen based on :-

- Familiarity of the situation.
- Characteristics of the application task.
- Notational features of the language.

Novice often experience difficulty due to :-

- Lack of adequate knowledge
- Lack of suitable strategies for responding to a specific situation.
- Incapability to use the necessary knowledge they have.

Strategy-Centred approaches

Classification of Strategies

- Top Down Vs Bottom Up
- Forward Vs Backward Development
- Breadth-first vs Depth-first
- Procedural Vs Declarative
- Mental Simulation

Strategy-Centred approaches

Top Down Vs Bottom Up

Top Down :

- Programmer develops the solution at an abstract level and then refines it.
- Usually associated with experts

Bottom Up :

- The solution is developed at every detailed level before more abstract structure is identified.
- **Novices usually try to develop bottom-up by writing in the final programming language and then building the abstract structure of the solution.**
- **Also used by experts when libraries of reusable components are available (or) when a product line is being developed.**

Strategy-Centred approaches

Forward Vs Backward Development

Forward Development:

- Solution is developed in direction of execution of the procedure.
- Use by beginners reflects their mental execution of the solution.
  - Solution relies not on computing knowledge, but on knowledge of the problem domain.
  - They recall procedures that they develop in a forward sense.
- Experts use it to retrieve a known solution schema from memory and implement it.

Backward Development :

- Direction of development maybe backward when no known schema procedure is available.

Strategy-Centred approaches

Breadth-first Vs Depth-first

Breadth-first

- Developing all the elements of the solution at one level of abstraction before proceeding to the next.
- The term ‘balanced development’ has been used to describe situation, when the solution is developed completely at level n, then at level n+1 and so on.
  - Experts are observed to use this strategy to solve problems that are relatively simple and familiar.

Depth-first

- One element of the system is developed to all levels of abstraction before any other element is developed.

Strategy-Centred approaches

Procedural Vs Declarative

Procedural

- Structure of the procedure controls the solution.
- Solution is based on steps of execution or procedures.
- Analogous to procedure-driven software development.
- Eg. Methods that emphasize on functional decomposition.

Declarative

- Static properties such as objects and roles, control the situation.
- Analogous to data-driven software development.
- Eg. Methods that concentrate on data analysis and database design.

Strategy-Centred approaches

Mental Simulation

Simulation can be used to evaluate a solution.

- Designers use mental simulation on partial or complete solution at higher or lower levels of abstraction.
- It provides them a way to verify that a solution meets desired objectives.
- Also a way of integrating partial solutions by controlling their interactions.

Topics Covered

Organization-centered Approach

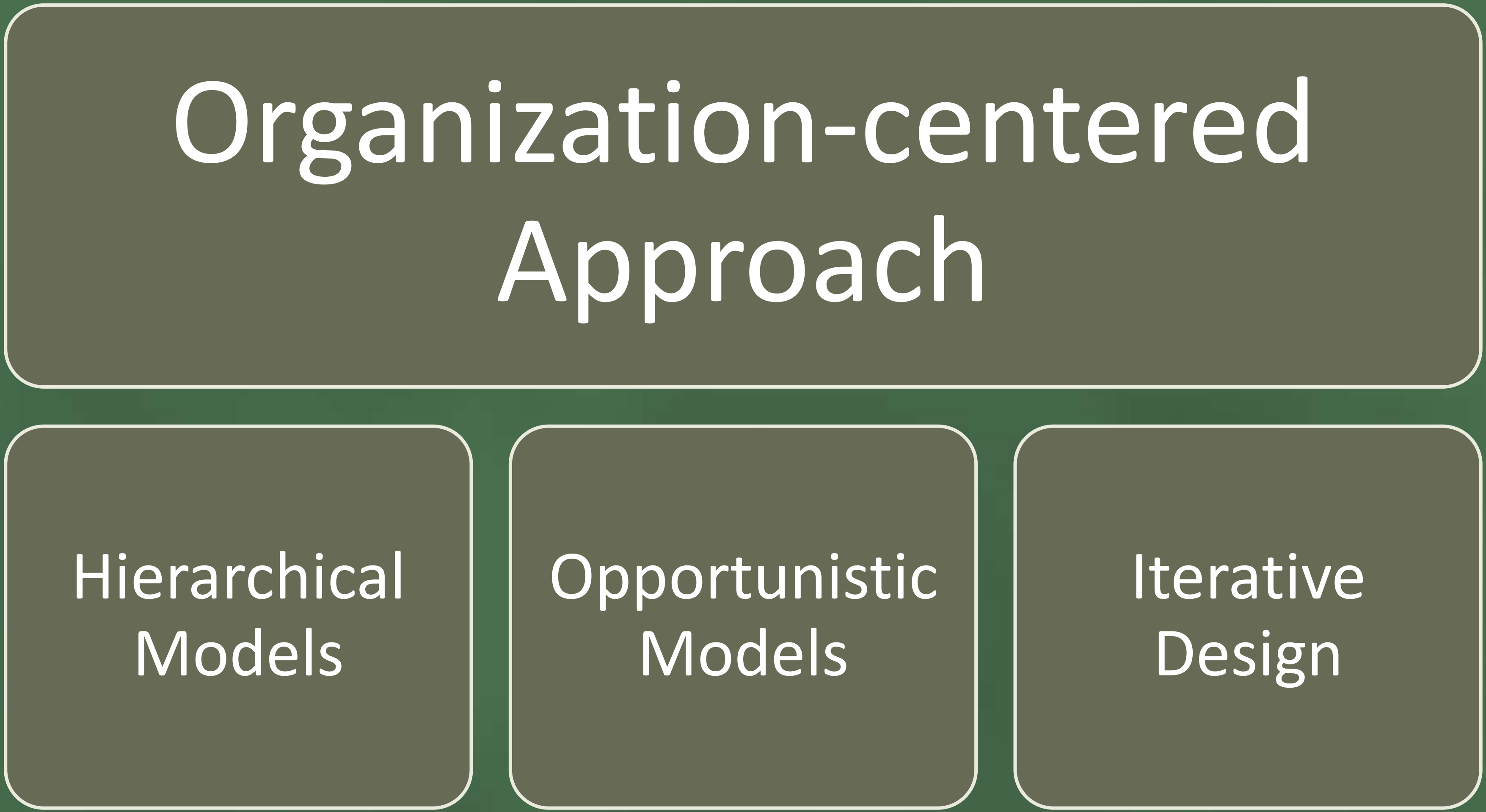
- Hierarchical v/s Opportunistic Model
- Iterative Nature of Design

Judging Expertise

- Defining Experts and Novices
- Levels of Expertise
- Stages of Acquiring Expertise

Better Programming Tools

- What makes a tool more suitable for programmers?



Organization-centered Approach

Hierarchical Models

Opportunistic Models

Iterative Design

Organization-centered Approach

Hierarchical Models

Opportunistic Models

Iterative Design

Hierarchical Model

- Process of breaking down a problem into a top-down or bottom-up structure.
- All goals/functions are identified at a certain level of abstraction before being refined successively into more levels.
- Encourages Breadth-first design as opposed to depth-first.
- Strongly influenced by structured programming.

Organization-centered Approach

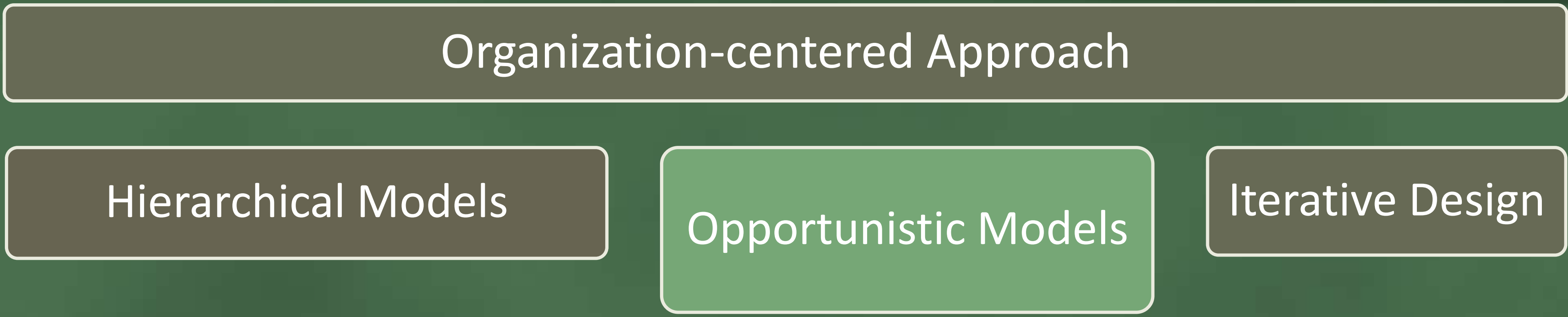
Hierarchical Models

Opportunistic Models

Iterative Design

Hierarchical Model is not ideal

- Real design is organized opportunistically.
- Designers focus on different aspects of the solution during design process.



Organization-centered Approach

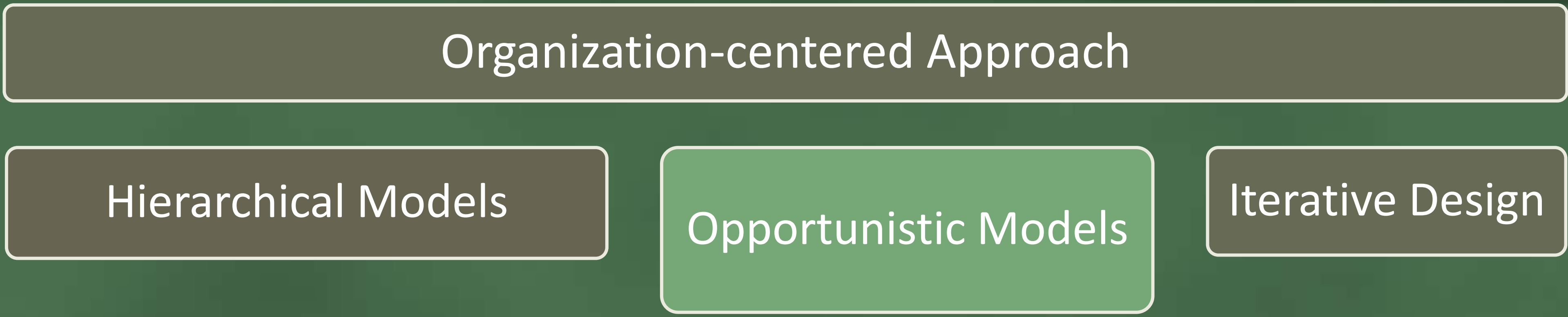
Hierarchical Models

Opportunistic Models

Iterative Design

Opportunistic Model

- Real design is organized opportunistically.
- Designers focus on different aspects of the solution during design process.



Organization-centered Approach

Hierarchical Models

Opportunistic Models

Iterative Design

Opportunistic Model

- Can cope with situations where designers need to focus on areas which are more critical than the rest.
- If information needed to handle an aspect of the design is not present, it is put aside.
- Resource limitations might force designers to solely focus or ignore a component temporarily.

Organization-centered Approach

Hierarchical Models

Opportunistic Models

Iterative Design

Opportunistic Model isn’t ideal either

- Causes deviations due to failure of working memory.
- Not all components have the same level of maturity.
- Opportunistic design either leads or lags ideal design.

Iterative Design

- Design activity is inherently iterative. (Design, Code, Revise)
- This iterative cycle is usually accompanied by intensive note-taking.

Organization-centered Approach

Hierarchical Models

Opportunistic Models

Iterative Design

Note-Taking: Hayes and Flower Model

- Plan the structure of the text.
- Translate the plan of the text into linguistic representation.
- Review the Text.
  - Revision: Some notes aren’t thorough and/or lack organization.
  - Deletion: Some notes don’t need to be in final documentation.

Judging Expertise

Separating an Expert from a Novice

- Organization of knowledge
  - Experts possess hierarchical knowledge.
  - Experts have better processing capacity.
  - Experts have more Understanding and Recall.

- Strategies and use of Knowledge

Judging Expertise

Separating an Expert from a Novice

- Strategies and use of Knowledge
  - Experts construct a more complete problem representation before solving the problem.
  - Experts use design rules.
  - Possess meta-cognitive knowledge i.e. know a number of alternative strategies and can select the optimal one for the problem at hand.
  - Use more external memory.
  - Use Top-Down approach for familiar problems.
  - Retrieve schemas for known problems. Novices build their own schemas.
  - Some aspects of programming tasks are carried out automatically.
    - IDE macros and shortcuts???

Judging Expertise

“Super-Experts”

- More technical and computing knowledge. (duh!)
- Broader experience instead of longer experience.
- Ability to combine computing knowledge with application domain.
- Better Social Skills (Communication, Cooperation, etc.).

Judging Expertise

Stages of Acquiring Expertise

Construction of elementary schemas.

Construction of complex schemas.

- Schema becomes hierarchically superior to other schema.
- Effect of schema is recognized by experts, not users.

Converting structure of schemas into a hierarchy by abstraction from focal point.

More Suitable Tools for Programmers

Where can you make improvements?

- Implementation and Visualization of Schema.
- Implementation of various Design Strategies
- Teaching Tools

More Suitable Tools for Programmers

Implementation and Visualization of Schema

- Knowledge-base level
  - Programming Language Specific Features (Code Snippets and Templates)
- Structural level
  - Display of non-contiguous elements of code/components to be grouped together. (UML-based and Reverse Engineering tools, etc.)

More Suitable Tools for Programmers

Implementation of Design Strategies

- Helps designers make strategic design choices.
  - Top-Down or Bottom-Up?
  - Forward Design or Backward?
  - Procedural or Declarative approach?

Top-Down supported by many languages. Not so many for Bottom-Up.

Support for backward development available (MAIDAY) but not for forward.

Simulation support for debugging.

More Suitable Tools for Programmers

Opportunistic Design

- Very few environments available that support opportunistic design.
- Features of HOODNICE, ReuseNICE:
  - Several levels of the design tree can be displayed and modified.
  - Solution can be temporarily inconsistent with rules defined by the method.
  - Notes concerning design decisions can be stored in a workbook.
  - HOOD Editor supports top-down and bottom-up design.

What do designers want?

A tool that gives a representation of the solution plan.

Helps identify incomplete and ignored tasks.

More Suitable Tools for Programmers

Teaching Tools

- Learning models that monitors the process of acquiring knowledge schemas.
    - With the hope that teaching schemas to beginners will help them develop expertise.
- Major Limitation
    - Does not help build strategic knowledge.

Future Research

Unexplored areas of Software Design

- Analysis of Subjects’ understanding of problem.
  - Currently, it is assumed that given a problem, everyone creates the same representation of it.
- Learning models that monitor the process of acquiring knowledge schemas.
  - With the hope that teaching schemas to beginners will help them develop expertise.
- Understanding why various design strategies were adopted.
  - This would help integrate all design strategies in one tool.
- Acquisition of expertise and its analysis.