Conceptual Design

- Tells the customer what the system will do
- Answers:
  - Where will the data come from?
  - What will happen to the data in the system?
  - What will the system look like to users?
  - What choices will be offered to users?
  - What is the timing of events?
  - What will the reports and screens look like?

Characteristics of Good Conceptual Design

- in customer language with no technical jargon
- describes system functions
- independent of implementation
- linked to requirements
Technical Design

• Tells the programmers what the system will do
• Includes:
  – major hardware components and their function
  – hierarchy and function of software components
  – data structures
  – data flow

<table>
<thead>
<tr>
<th>WHAT</th>
<th>HOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONCEPTUAL DESIGN</td>
<td>TECHNICAL DESIGN</td>
</tr>
<tr>
<td>function</td>
<td>form</td>
</tr>
</tbody>
</table>

Customers → System designers

System builders
'The user will be able to route messages to any other user on any other network computer.'

Five Ways to Create Designs

- Modular decomposition
- Data-oriented decomposition
- Event-oriented decomposition
- Outside-in design
- Object-oriented design
Three Design Levels

- Architecture: associates system components with capabilities
- Code design: specifies algorithms and data structures for each component
- Executable design: lowest level of design, including memory allocation, data formats, bit patterns
Design Styles

- Pipes and filters
- Object-oriented design
- Implicit invocation
- Layering
- Repositories
- Interpreters
- Process control
- Client-server
Example of Implicit Invocation

DEBUG VALUE <system><file><line><var><value>
DEBUG ENTER <system><file><func><line><value>
DEBUG EXIT <system><file><func><line><value>
EVENT ADD <system><id#><event_type><file><line><text>
EVENT REMOVE <system><id#><event_type><file><line><text>
STOP-ERROR <signal><file><line>
DEBUG AT <system><file><func><line>
DEBUG FOCUS <system><file><func><line>
DEBUG CLEAR <system>
DEBUG RESET <system>
WHERE <system><level><file><func><line><addr><args>
WHERE_DUMP <system><level><name><value>
WHERE_BEGIN <system>
WHERE_END <system><level>
DEBUG SYSTEM <system>
DEBUG NOSYSTEM <system>
UPDATE <system><file><line>

Cryptography
File interface
Key management
Authentication
Users
FEEDBACK LOOP:

Input variables → Controller → Process → Controlled variable
Set point → Changes to manipulated variables

FEEDFORWARD LOOP:

Input variables → Controller → Process → Controlled variable
Set point → Changes to manipulated variables

Ring topology
Star topology
Bus Topology

Client-server

- **Client**
  - Requests action or service
  - Knows the identity of the server
  - Sends a message requesting service and waits for response
- **Server**
  - Does not know number of clients
  - Does not know client’s identity
  - Needs sophisticated security, systems management, and applications development.
Important Design Issues

- Modularity and levels of abstraction
- Collaborative design
- Designing the user interface
  - metaphors, mental model, navigation rules, look and feel
  - cultural issues
  - user preferences
- Concurrency
- Design patterns and reuse

Example of Abstraction

Level 1:
Rearrange L in nondecreasing order.

Level 2:
while (I is between 0 and (length of L)-2)
  set LOW to index of smallest value in L(I) .. L((length of L)-1)
  exchange L(I) and L(LOW)

Level 3:
while (I is between 0 and (length of L)-2)
  LOW = I
  while (J is between I+1 and (length of L)-1)
    if (I(LOW) > I(J))
      LOW = J
  TEMP = L(LOW)
  L(LOW) = L(I)
  L(I) = TEMP
Collaborative Design

- On most projects the design is not created by one person.
- Different parts are assigned to different people.
- Issues:
  - Who is best qualified to design which component
  - How do document the design
  - How to coordinate the design

Causes of Design Breakdown

- Lack of specialized design schemes
- Lack of meta-scheme about design process leading to poor allocation of resources
- Poor prioritization of issues leading to poor selection of alternative solutions
- Difficulty in considering all the stated or inferred constraints
- Difficulty in performing mental simulations with many steps or test cases
- Difficulty in keeping track and returning to subproblems whose solution had been postponed
- Difficulty in merging solutions from individual subproblems to form a complete solution
Stages in Distributed Development

1. A project is performed at a single site with on-site developers from foreign countries.
2. On-site analysts determine the system’s requirement. Then the requirements are provided to off-site groups of designers and programmers.
3. Off-site developers build generic products and components that are used world wide.
4. Off-site developers build products that take advantage of their individual expertise.

Cultural and Language Issues

- Individuals from a culture where harmony is important, may not express individual opinions.
- Using electronic communication, nuances expressed by gestures and facial expressions are lost.
- Translation from one natural language to another is not precise.
  - There are over 500 words to describe pasta in Italian
  - There are over 160 words for “camel” in Bedoin
Designing the User Interface

- Metaphors
  - The fundamental terms, images, and concepts that can be recognized and learned.
- Mental Model
  - The organization and representation of data, functions, tasks, and roles
- Navigation Rules
  - How to move among data, functions, tasks, and roles
- Look and Feel
  - The system’s appearance and interaction techniques

Cultural Issues

- Must consider beliefs, values, norms, traditions, mores, and myths of those who will use the system.
- Different numerical formats.
- Text descriptions in various languages have varying lengths.
- Meanings of colors.
  - England: purple represents royalty
  - Japan: purple represents dignity and nobility
  - Greece: death and evil
Table 5.1. Issues to consider in trade-off analysis. (Lane, in Shaw and Garlan 1996)

<table>
<thead>
<tr>
<th>Functional dimensions</th>
<th>Structural dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>External event handling</td>
<td>Application interface abstraction level</td>
</tr>
<tr>
<td>• No external events</td>
<td>• Monolithic program</td>
</tr>
<tr>
<td>• Process events while waiting for input</td>
<td>• Abstract device</td>
</tr>
<tr>
<td>• External events prompt user commands</td>
<td>• Toolkit</td>
</tr>
<tr>
<td>User customizability</td>
<td>• Interaction manager with fixed data types</td>
</tr>
<tr>
<td>• High</td>
<td>• Interaction manager with extensible data types</td>
</tr>
<tr>
<td>• Medium</td>
<td>• Extensible interaction manager</td>
</tr>
<tr>
<td>• Low</td>
<td>Abstract device variability</td>
</tr>
<tr>
<td>User interface adaptability across devices</td>
<td>Ideal device</td>
</tr>
<tr>
<td>• None</td>
<td>• Parameterized device</td>
</tr>
<tr>
<td>• Local behavior changes</td>
<td>• Device with variable operations</td>
</tr>
<tr>
<td>• Global behavior change</td>
<td>• Ad hoc device</td>
</tr>
<tr>
<td>• Application semantics change</td>
<td>Notation for user interface definition</td>
</tr>
<tr>
<td>Computer system organization</td>
<td>• Implicit in shared user interface code</td>
</tr>
<tr>
<td>• Uniprocessing</td>
<td>• Implicit in application code</td>
</tr>
<tr>
<td>• Multiprocessing</td>
<td>• External declarative notation</td>
</tr>
<tr>
<td>• Distributed processing</td>
<td>• External procedural notation</td>
</tr>
<tr>
<td>Basic interface class</td>
<td>• Internal declarative notation</td>
</tr>
<tr>
<td>• Menu selection</td>
<td>• Internal procedural notation</td>
</tr>
<tr>
<td>• Form filling</td>
<td>Basis of communication</td>
</tr>
<tr>
<td>• Command language</td>
<td>• Events</td>
</tr>
<tr>
<td>• Natural language</td>
<td>• Pure state</td>
</tr>
<tr>
<td>• Direct manipulation</td>
<td>• State with hints</td>
</tr>
<tr>
<td>Application portability across user styles</td>
<td>• State plus events</td>
</tr>
<tr>
<td>• High</td>
<td>Control thread mechanisms</td>
</tr>
<tr>
<td>• Medium</td>
<td>• None</td>
</tr>
<tr>
<td>• Low</td>
<td>• Standard processes</td>
</tr>
<tr>
<td></td>
<td>• Lightweight processes</td>
</tr>
<tr>
<td></td>
<td>• Non-preemptive processes</td>
</tr>
<tr>
<td></td>
<td>• Event handlers</td>
</tr>
<tr>
<td></td>
<td>• Interrupt service routines</td>
</tr>
</tbody>
</table>
Concurrency

- **Mutual exclusion**
  - A component desiring to change the state of an object:
    - Locks the object.
    - Changes the state.
    - Unlocks the object.

- **Monitors**
  - An abstract object (in the OO sense of the word)
    - Mutator methods perform the locking and unlocking.

- **Guardians**
  - A task that is always running to control access to a resource.
    - Uses a rendezvous to coordinate multiple requests.

Characteristics of Good Design

- **Component independence**
  - coupling
  - cohesion

- **Exception identification and handling**

- **Fault prevention and tolerance**
  - active
  - passive
Uncoupled - no dependencies
Loosely coupled - some dependencies
Highly coupled - many dependencies

HIGH COUPLING

Content coupling
Common coupling
Control coupling
Stamp coupling
Data coupling
Uncoupled

LOW
Component B
Go to D1

Component D
Go to D1
D1:

Global:
A1
A2
A3
Variables:
V1
V2

Common data area and variable names

Component X
Change V1 to zero

Component Y
Increment V1

Component Z
V = V2 + A1
Sequential
Communication
Procedural
Temporal
Logical
Coincidental

HIGH COHESION

FUNCTIONAL
Sequential with complete, related functions

COMMUNICATIONAL
Access same data

COINCIDENTAL
Parts unrelated

FUNCTION A
FUNCTION B
FUNCTION C
FUNCTION D
FUNCTION E

FUNCTION A
FUNCTION A'
FUNCTION A''

TIME T0
TIME T0 + X
TIME T0 + 2X

FUNCTION A
FUNCTION B
FUNCTION C

FUNCTION A - part 1
FUNCTION A - part 2
FUNCTION A - part 3

LOGICAL
Similar functions

FUNCTIONAL
Sequential with complete, related functions

TEMPORAL
Related by time

FUNCTIONAL
Sequential with complete, related functions

PROCEDURAL
Related by order of functions
Exception Identification and Handling

- Types of exceptions
  - Failure to provide a service
  - Providing the wrong service or data
  - Corrupting data
- Handling exceptions
  - Retrying
  - Correct
  - Report

Exception Handling in Java

```java
try {
    code in which an exception may occur
}
catch (exception e) {
    code to process exception
}
finally {
    code to be executed whether or not exception occurred
}
```
Fault Prevention and Fault Tolerance

- **Fault:**
  - Results when a human makes a mistake in the requirements, design, or code.

- **Failure**
  - Departure of a system from its required behavior.

- **Not every fault may result in a failure**
  - Fault containing code never executed
  - Data may not exceed bounds

Active Fault Detection

- **Mutual suspicion**
  - Each component assumes that the components feeding it contain faults.
    - Check input data for validity.

- **Use redundancy**
  - Check sum of sum of rows against sum of sum of columns.
  - Use checksums and parity.
  - Multiple versions of the same software by different design teams, different programming languages, different computer architectures.
    - Shown to be less than reliable because designers tend to use common design patterns and techniques.
Techniques for Improving Design

- Reducing complexity
- Design by contract
- Prototyping design
- Fault-tree analysis

Reducing Complexity

- Redraw diagrams to reduce/eliminate crossovers.
- Simplifying logic.
Conditions

w  Student has a grade-point average of B or better
x  Student has combined SAT > 1200
y  Student has outstanding recommendations
z  Student has extracurricular activities

Actions

A1  Invite for early admission
A2  Admit for regular semester
A3  Invite for honors program
A4  Invite for regular program
A5  Conditional admission
A6  Reject

<table>
<thead>
<tr>
<th>Variables</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A1  A2  A3  A4  A5  A6</td>
</tr>
<tr>
<td>w 0 0 0 0</td>
<td>X</td>
</tr>
<tr>
<td>x 0 0 0 1</td>
<td>X</td>
</tr>
<tr>
<td>y 0 0 1 0</td>
<td>X</td>
</tr>
<tr>
<td>z 0 0 1 1</td>
<td>X</td>
</tr>
<tr>
<td>w 0 1 0 0</td>
<td>X</td>
</tr>
<tr>
<td>x 0 1 0 1</td>
<td>X</td>
</tr>
<tr>
<td>y 0 1 1 0</td>
<td>X</td>
</tr>
<tr>
<td>z 0 1 1 1</td>
<td>X</td>
</tr>
<tr>
<td>w 1 0 0 0</td>
<td>X</td>
</tr>
<tr>
<td>x 1 0 0 1</td>
<td>X</td>
</tr>
<tr>
<td>y 1 0 1 0</td>
<td>X</td>
</tr>
<tr>
<td>z 1 0 1 1</td>
<td>X</td>
</tr>
<tr>
<td>w 1 1 0 0</td>
<td>X</td>
</tr>
<tr>
<td>x 1 1 0 1</td>
<td>X</td>
</tr>
<tr>
<td>y 1 1 1 0</td>
<td>X</td>
</tr>
<tr>
<td>z 1 1 1 1</td>
<td>X</td>
</tr>
</tbody>
</table>
Simplifying the Logic

\[ A1 = wxyz + wxyz' \]
\[ A1 = wxy(z + z') \]
\[ A1 = wxy(1) \]
\[ A1 = wxy \]

Similarly we can conclude that:
\[ A2 = xy' + w'xy + wx' \]
\[ A3 = wx \]
\[ A4 = x'y + w'x + wx'y' \]
\[ A5 = w'x'y \]
\[ A6 = w'x'y' \]

Design by Contract

- Software components client and supplier agree on
  - Precondition: client ensures is true
  - Invariant: supplier maintains truth
  - Postcondition: supplier ensures is true, if precondition is true.
Example of Contract

1. The client ensures that the dictionary is not full and the key is not empty.
2. The supplier records the element in the table.
3. The client can access the updated table.
4. If the table is full, or the key is empty, no action is taken.

Example Using Eiffel

put (x: Elemen; key: STRING) is
require
count <= capacity;
not key.empty
end
... Some insertion algorithm
ensure
has(x);
item(key)=x;
count = old count + 1
end
Prototyping Design

- Not sure that requirement can be satisfied.
- A feasibility prototype allows us to find out whether the proposed solution will meet the requirement.
- Prototype concentrates on the specific design issue, and is meant to be thrown away.
- Current research has encouraged the use of rapid prototyping to build and save parts of the prototype for use in the actual system.

Fault Tree Analysis

- Fault tree is a method that helps to decompose a design and look for situations that might lead to failure.
  - Identify possible failures
  - Construct a graph showing the relationship of the failures
- Identify
  - Single points of failure
  - Uncertainty
  - Ambiguity
  - Missing components
Design Evaluation and Validation

- Mathematical validation
- Measuring design quality
- Comparing designs
  - one specification, many designs
  - comparison table
- Design reviews
Mathematical Validation

- Break the system into a set of processes
- For each process
  - Set of inputs
  - Set of expected outputs
  - A set of assertions
  - Demonstrate
    - If the input is formulated correctly, it is transformed properly
      into the set of expected outputs.
    - The process terminates without failure.
- Time consuming and expensive process
  - Generally limited to most critical parts of a system.
    - Security (US)
    - Safety (UK)

Design Quality

![Graph showing the relationship between system design complexity and faults per thousand lines of code.](Diagram)
Comparing Designs

- **Key Word in Context**
  - Accepts an ordered set of lines; each line is an ordered set of words, each word is an ordered set of characters. Any line may be circularly shifted by repeatedly removing the first word and appending it at the end of the line.
  - A KWIC index system outputs a list of all circularly shifts of all lines in alphabetical order.
Table 5.5. Weighted comparison of Shaw and Garlan designs.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Priority</th>
<th>Shared data</th>
<th>Abstract data type</th>
<th>Implicit invocation</th>
<th>Pipe and filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy to change algorithm</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Easy to change data representation</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Easy to change function</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Good performance</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Easy to reuse</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>
Design Reviews

• Preliminary design review
  – examines conceptual design with customer and users
• Critical design review
  – presents technical design to developers
• Program design review
  – programmers get feedback on their designs before implementation

Questions for Any Design Review

Is it a solution to the problem?
Is it modular, well-structured, and easy to understand?
Can we improve the structure and understandability?
Is it portable to other platforms?
Is it reusable?
Is it easy to modify or expand?
Does it support ease of testing?
Does it maximize performance, where appropriate?
Does it reuse components from other projects, where appropriate?
Are the algorithms appropriate, or can they be improved?
If this system is to have a phased development, are the phases interfaced sufficiently so that there is an easy transition from one phase to the next?
Is it well-documented, including design choices and rationale?
Does it cross-reference the components and data with the requirements?
Does it use appropriate techniques for handling faults and preventing failures?
Documenting the Design

- design rationale
- menus and other display-screen formats
- human interfaces: function keys, touch screen descriptions, keyboard layouts, use of a mouse or joystick
- report formats
- input: where data come from, how they are formatted, on what media they are stored
- output: where data are sent, how they are formatted, on what media they are stored
- general functional characteristics
- performance constraints
- archival procedures
- fault-handling approach

Information System Example

Opposition schedule = * Data flow *
   Television company name
   +  {Opposition transmission date
       +  Opposition transmission time + Opposition program name
       +  (Opposition predicted rating)}

Input: Opposition schedule
For each Television company name, create Opposition company.
For each Opposition schedule,
   Locate the Episode where Episode schedule date = Opposition
   transmission date AND Episode start time = Opposition
   transmission time
Create instance of Opposition program
Create the relationships Planning and Competing
Output: List of Opposition programs
Real-time Example

- “Not all conversions [from Ariane-4] were protected because a maximum workload target of 80% had been set for the SRI computer.”
- Designers felt that in certain cases, an overflow could not occur.
- This conclusion was based upon trajectory parameters of the Ariane-4, not the Ariane-5.

Real-time Example (Cont.)

- The text suggests that design by contract might have caught the error during testing.
- The SRI was programmed using Ada, which has an exception mechanism.
- Unfortunately, the designers chose to treat exceptions as being caused by hardware failure, thus they shutdown the SRI when the exception occurred. The backup then took over, but it, too, hit the same exception, and it re-started.
- The OBC could also have validated its input.