

CSE 5324: Software Engineering I

(Analysis, Design, Creation)

Design - Methods of Design

What is design?

Software Design - is a process through which requirements are translated into a representation of software.

- Focus is on HOW the program will work.
- Try to avoid programming language and hardware specific details that affect HOW.
- Develop a program architecture and map requirements to portions of the architecture.

The Designer's goal -
produce a model or representation of an entity that will later be built.

Why design?

Makes Implementation Mechanical
Different from Requirement or Code

How is design done:

Many methods:

Structured

OO

Etc.

NEED FOR DESIGN

Design is the only way we can accurately translate a customer's requirements into a finished software product

Without a design we risk building an unstable system:

- one that will fail when small changes are made (maintainability)
- one that may be difficult to test (testability)
- one whose quality can not be determined until late in the development process

Goal:

Define process (or system) in enough detail to
Implement. (Create, realize)

What is designed:

Data

Architecture

Interface

Procedural

Data design:

Wasserman:

“Select logical representations of data objects”

Should apply same analysis principles as applied to function

Identify data structures and operations performed on them

Data dictionary

Defer low-level data decisions

Hide data representation in modules

Develop library of data structures and operations
(reuse)

Use appropriate programming language

Architecture design:

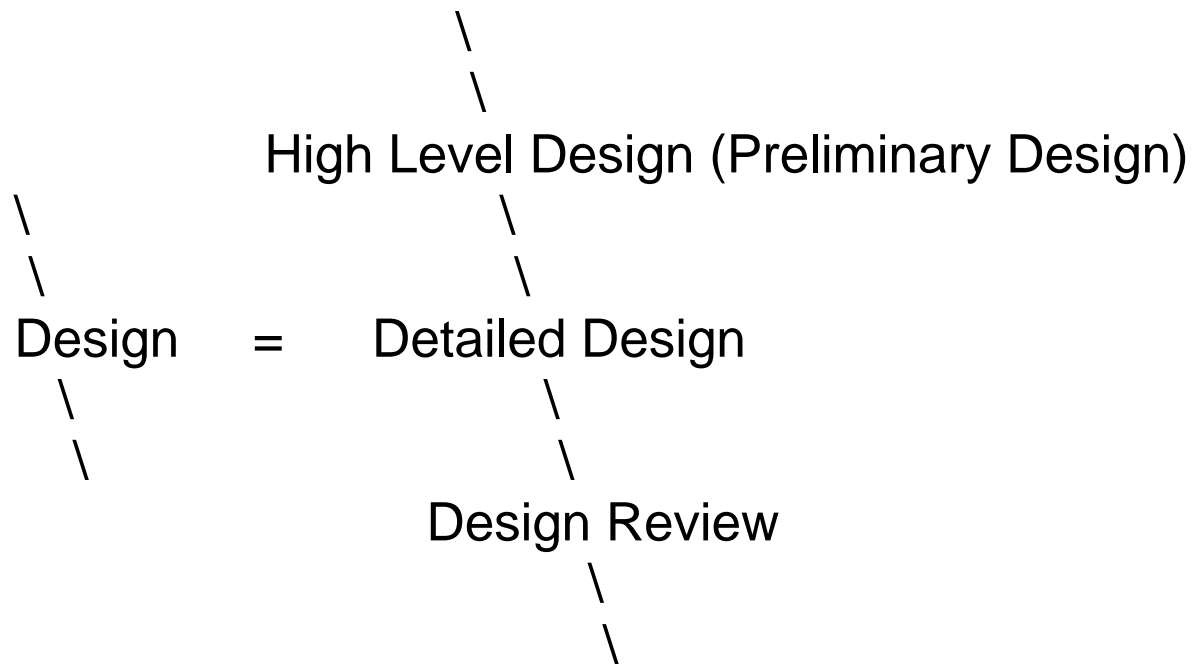
Develop modular program structure and show control relationships

This is very important: this shows the “outline” of the entire system.

DESIGN PROCESS:

The design for a system is created in stages.

From a project management point of view
(refinement of lifecycle)



(This started from “top-down” and “structured-programming” and “modular” ideas of late 1960’s and early 1970’s)

Data-flow oriented design:

Natural flow from analysis

Find type of information flow

Find flow boundaries

Map DFD to program flow

Factor control

Refine structure

Find type of information flow

Transform Flow
Transaction Flow

Transform Flow

Incoming flow -> Transform center ->
Outgoing flow

Overall flow is sequential, one of few paths
Look at DFD for few (one) straight line paths

Transaction Flow

Data Item triggers other flows

Select from many paths

Transaction->Transaction Center->

->Many

->Paths

In most systems BOTH flows are present.
(Find flow boundaries)

Map DFD to program flow

Transform Mapping – DFD with Transform Flow mapped to template for program structure

Design Steps:

Review Level 0 DFD (system spec and SRS)

Refine Data Flow (Show Details)

Is DFD transform or transaction?

Find and isolate transform center (find in and out flow boundaries)

Do a first level factoring
(top down distribution of control)

Do a second level factoring
(individual DFD bubbles to modules)

Refine using heuristics for design

Transaction Mapping:

Similar to above, BUT

Find transaction center

Factoring is on path basis

Afterwards:

- Need to describe processing for each module

- Describe Interfaces

- Local and Global Data structures

- Note Limits, Restrictions

- Review

- Optimize

Design Optimization:

- Knuths Law: “Don’t”

- Use CASE tools

- Analyze hot spots (time hogs), use appropriate

 - Algorithms

- Use appropriate programming Language

- Use instrumentation to find heavy loaded

 - modules: SW or HW

- Redo those

Interface Design:

Internal
External
Human

Some guidelines:

- Simplify information passing

- No globals

- Validate data

- Error handling and propagation: fix it,
or pass it up.

User Interface

- KISS

- WIMP (?)

Who are users?

- User Model

 - Experiences, other background

 - Novices, intermittent, frequent users

Design Issues:

HELP:

How can User get “Help”?

Is help available for all functions?

Which?

How to show help?

How to get out of help?

ERRORS:

How to show error messages

Provide some advice

Indicate any unexpected: (File lost, etc)

Don't blame User

COMMANDS:

Short cuts

Menus

Customization

Offer feedback (when slow, etc.)

Hints:

Iterate

Prototype

Use tool kits

USE IT!

User Interface Guidelines:

General:

- Be consistent
- Undo
- Make destruction difficult

Display:

- Show (only) important info
- Be consistent, predictable
- Compartmentalize
- Show analog displays (guage)

Data Input:

- Minimize
- Consistent
- Let user control flow
- Customize

Procedural Design:

Structured:

Sequence, condition, repetition

Graphical:

Flow Chart

Nassi-Shneiderman

Tables:

Decision Tables: (Like PLA's)

PDL (Program Design Language)

Pseudocode, structured English

Combined:

CSD (Control Structure Diagram)

(Mostly with CASE tools)

Why is design Important?

Is Structured Design the way to go?

Pro:

Popular

Fits with SA

Lots of Experience

Tools

The way you think

Flexible

Cons:

Like SA

Is OO more “natural”

Easy to omit, forget, etc. functions

Maintenance

Why Design at all?

High Level Design

- Also called Preliminary Design
- Concerned with the transformation of requirements into data and software architecture
- PRIMARY INPUT is the SRS
- PRIMARY OUTPUT is an architectural design and data design

Detailed Design

- Focuses on refinements to the architectural representation that lead to detailed data structure and algorithmic representations for software
- PRIMARY INPUTS are the outputs of the High Level Design step
- PRIMARY OUTPUT is the SDD (Software Design Description)
 - + contains architectural design (structure charts),
 - detailed design (module specifications)
 - data design (Design data dictionary)

and a Software Design Walkthrough document.
(We don't use here anymore)

STRIVING FOR QUALITY

Quality is an important objective of software design.

- Quality in the system really begins at the design stage
- Designs can be assessed for quality
 - One proven metric that can predict quality is $(\text{Fan-Out})^2$
i.e For a given module, as the number of other modules called by it increases, the complexity of the calling module is likely to increase, making that module more likely to contain mistakes.

Guidelines for Design Quality (Pressman)

- 1) A design should exhibit a hierarchical organization that makes intelligent use of control among components of software.
** Abstraction
- 2) A design should be modular; that is, the software should be logically partitioned into components that perform specific functions and subfunctions.
** Cohesion
- 3) A design should contain distinct and separable representations of data and procedure.
- 4) A design should lead to modules (e.g functions or procedures) that exhibit independent functional characteristics.
** Coupling, Cohesion
- 5) A design should lead to interfaces that reduce the complexity of connections between modules and with

the external environment.

** Coupling

- 6) A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis.

** methods, techniques

Design - Methods of Design

What is design?

Software Design - is a process through which requirements are translated into a representation of software.

- Focus is on HOW the program will work.
- Try to avoid programming language and hardware specific details that affect HOW.
- Develop a program architecture and map requirements to portions of the architecture.

The Designer's goal -
produce a model or representation of an entity that will later be built.

Object Oriented Methodolgy: Terms, ideas, techniques
OO Concepts, OO Analysis, OO Design
Design Concepts, Design Methods (Structured)
Real Time Analysis and Design

What is designed:

Data

Architecture

Interface

Procedural

Object Oriented Design:

Find objects, factor into good classes, define class hierarchies and interfaces, make it reusable, modifiable.

OO Design architecture stresses object relations rather than flow of control

What is different: (Fichman, Kemerer)
(From SA)

Module Hierarchy representation
Data Definition Spec
Procedural Spec
End-to-end processing sequences
Class and Hierarchy definition
Operations assigned to classes
...

Bertrand Meyer suggests:

Decomposable – design method makes easier subproblems

Composable – reusable

Understandable – by module

Continuity – coupling (local)

Protection – coupling (isolation)

To achieve:

Few, small, explicit interfaces. Information hiding.

OOD:

Booch

Coad – Yourdon

Rumbaugh

Jackson – design a part of analysis!

Coad suggest (for any method): Define:

Problem Domain

User I/F

Task Management

Data Management

Design Patterns:

(This is another of the SE Buzz words!)

Recurring and reusing of classes and objects
(Gamma)

Name of pattern

Problem to which applied

Characteristics

Consequences of applying

Should one use inheritance or composition?

(When both possible)

Use what is general, or make it specific?

Compare SA, SD versus OO:

Hospital system: Patient record system.

Spreadsheet

Is OO Important?

OO Prog languages: JAVA, et al

OO Interfaces:
 Windowing systems

CORBA
 Distributed objects

CSE 5324: Software Engineering I

(Analysis, Design, Creation)

Apologies:

The instructor is currently being held as a political prisoner in Austin (UT system capitol) due to economic and political instability. He is planning a secret breakout (please don't tell anyone) and will be back Thursday.

Today's guest speaker will be the eminent:

Professor A. Reyes

Who will inspire you, illuminate you, answer questions (and tell better jokes)

Review

Form teams for project

Design! Design, design

Preview

Why you should take CSE 5328

New stuff

What is important

Last class(es):

Basics of Design:

Concepts

Fundamentals

Structured design

Team Project:

If there was anyone that wanted to be a team leader and:

- Did not present last class OR**
- Wants to add anything to last weeks presentation**

Please do so now (1 minute per person)

**Teams should be formed and final by Thursday, October 7
if you are not on a team, I'll find a good team for you.**

CSE 5328 Software Development Studio

Class is about...

Instructor

What you need to know to get in

How class is done

Why you should take it

(Insert appropriate joke here.)

Ho ho ho...

News (you can use)

"Last week NASA announced that the "human" error stemming from space engineers using two sets of measurements -- one utilizing miles and the other kilometers -- caused the loss of the Mars Climate Orbiter spacecraft last week, NASA said Thursday."

The teams, located at the National Aeronautics and Space Administration's Jet Propulsion Laboratory in Pasadena and at Lockheed Martin Astronautics in Colorado, complicated matters further by failing to realize the error, the agency said in a statement.

The \$125 million orbiter, intended to serve as the first interplanetary weather satellite, is believed to have broken up when it hit the Martian atmosphere last week after an approach that was too near the surface.

Today's question is:

What kind of error was this?

How could/(should) it have been prevented?

Review, and new stuff (use what ya' want)

Design - Methods of Design

What is design?

Software Design - is a process through which requirements are translated into a representation of software.

- Focus is on HOW the program will work.
- Try to avoid programming language and hardware specific details that affect HOW.
- Develop a program architecture and map requirements to portions of the architecture.

The Designer's goal -
produce a model or representation of an entity that will later be built.

Why design?

Makes Implementation Mechanical
Different from Requirement or Code

How is design done:

Many methods:

Structured

OO

Etc.

NEED FOR DESIGN

Design is the only way we can accurately translate a customer's requirements into a finished software product

Without a design we risk building an unstable system:

- one that will fail when small changes are made (maintainability)
- one that may be difficult to test (testability)
- one whose quality can not be determined until late in the development process

Goal:

Define process (or system) in enough detail to
Implement. (Create, realize)

What is designed:

Data

Architecture

Interface

Procedural

Data design:

Wasserman:

“Select logical representations of data objects”

Should apply same analysis principles as applied to function

Identify data structures and operations performed on them

Data dictionary

Defer low-level data decisions

Hide data representation in modules

Develop library of data structures and operations
(reuse)

Use appropriate programming language

Architecture design:

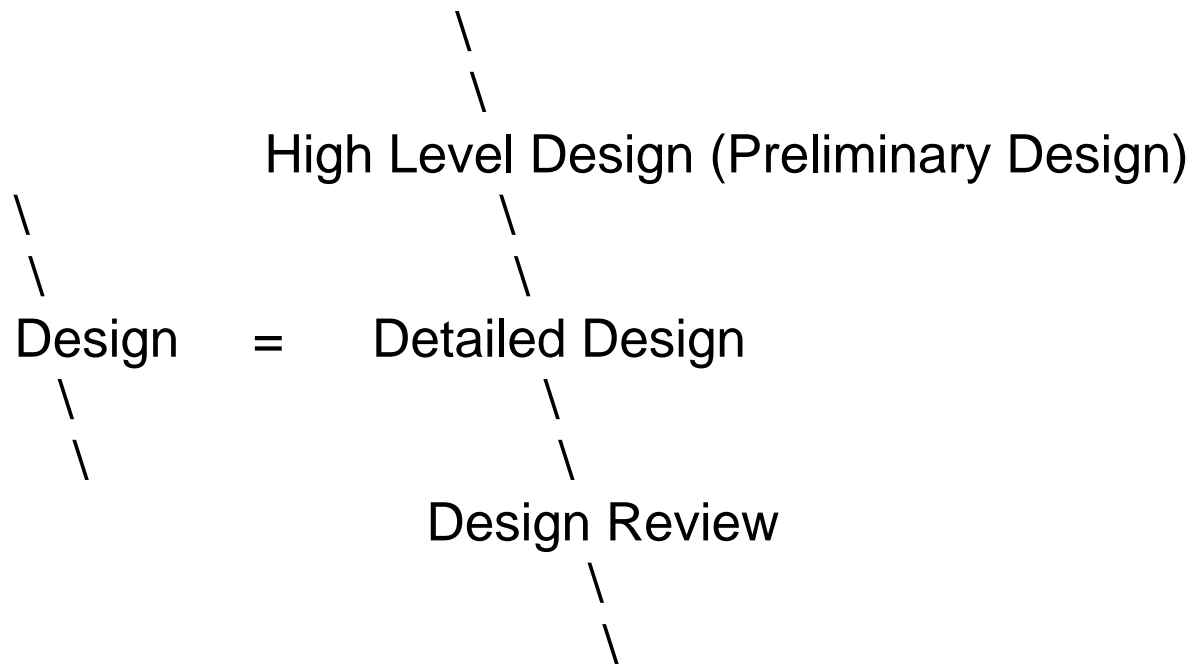
Develop modular program structure and show control relationships

This is very important: this shows the “outline” of the entire system.

DESIGN PROCESS:

The design for a system is created in stages.

From a project management point of view
(refinement of lifecycle)



(This started from “top-down” and “structured-programming” and “modular” ideas of late 1960’s and early 1970’s)

Data-flow oriented design:

Natural flow from analysis

Find type of information flow

Find flow boundaries

Map DFD to program flow

Factor control

Refine structure

Find type of information flow

Transform Flow
Transaction Flow

Transform Flow

Incoming flow -> Transform center ->
Outgoing flow

Overall flow is sequential, one of few paths
Look at DFD for few (one) straight line paths

Transaction Flow

Data Item triggers other flows

Select from many paths

Transaction->Transaction Center->

->Many

->Paths

In most systems BOTH flows are present.
(Find flow boundaries)

Map DFD to program flow

Transform Mapping – DFD with Transform Flow mapped to template for program structure

Design Steps:

Review Level 0 DFD (system spec and SRS)

Refine Data Flow (Show Details)

Is DFD transform or transaction?

Find and isolate transform center (find in and out flow boundaries)

Do a first level factoring
(top down distribution of control)

Do a second level factoring
(individual DFD bubbles to modules)

Refine using heuristics for design

Transaction Mapping:

Similar to above, BUT

Find transaction center

Factoring is on path basis

Afterwards:

- Need to describe processing for each module

- Describe Interfaces

- Local and Global Data structures

- Note Limits, Restrictions

- Review

- Optimize

Design Optimization:

- Knuths Law: “Don’t”

- Use CASE tools

- Analyze hot spots (time hogs), use appropriate

 - Algorithms

- Use appropriate programming Language

- Use instrumentation to find heavy loaded

 - modules: SW or HW

- Redo those

Interface Design:

Internal
External
Human

Some guidelines:

- Simplify information passing

- No globals

- Validate data

- Error handling and propagation: fix it,
or pass it up.

User Interface

- KISS

- WIMP (?)

Who are users?

- User Model

 - Experiences, other background

 - Novices, intermittent, frequent users

Design Issues:

HELP:

How can User get “Help”?

Is help available for all functions?

Which?

How to show help?

How to get out of help?

ERRORS:

How to show error messages

Provide some advice

Indicate any unexpected: (File lost, etc)

Don't blame User

COMMANDS:

Short cuts

Menus

Customization

Offer feedback (when slow, etc.)

Hints:

Iterate

Prototype

Use tool kits

USE IT!

User Interface Guidelines:

General:

- Be consistent
- Undo
- Make destruction difficult

Display:

- Show (only) important info
- Be consistent, predictable
- Compartmentalize
- Show analog displays (guage)

Data Input:

- Minimize
- Consistent
- Let user control flow
- Customize

Procedural Design:

Structured:

Sequence, condition, repetition

Graphical:

Flow Chart

Nassi-Shneiderman

Tables:

Decision Tables: (Like PLA's)

PDL (Program Design Language)

Pseudocode, structured English

Combined:

CSD (Control Structure Diagram)

(Mostly with CASE tools)

Why is design Important?

Is Structured Design the way to go?

Pro:

- Popular
- Fits with SA
- Lots of Experience
- Tools
- The way you think
- Flexible

Cons:

- Like SA
- Is OO more “natural”
- Easy to omit, forget, etc. functions
- Maintenance
- Why Design at all?

High Level Design

- Also called Preliminary Design
- Concerned with the transformation of requirements into data and software architecture
- PRIMARY INPUT is the SRS
- PRIMARY OUTPUT is an architectural design and data design

Detailed Design

- Focuses on refinements to the architectural representation that lead to detailed data structure and algorithmic representations for software
- PRIMARY INPUTS are the outputs of the High Level Design step
- PRIMARY OUTPUT is the SDD (Software Design Description)
 - + contains architectural design (structure charts),
 - detailed design (module specifications)
 - data design (Design data dictionary)

and a Software Design Walkthrough document.
(We don't use here anymore)

STRIVING FOR QUALITY

Quality is an important objective of software design.

- Quality in the system really begins at the design stage
- Designs can be assessed for quality
 - One proven metric that can predict quality is $(\text{Fan-Out})^2$
i.e For a given module, as the number of other modules called by it increases, the complexity of the calling module is likely to increase, making that module more likely to contains mistakes.

Guidelines for Design Quality (Pressman)

- 7) A design should exhibit a hierarchical organization that makes intelligent use of control among components of software.
** Abstraction
- 8) A design should be modular; that is, the software should be logically partitioned into components that perform specific functions and subfunctions.
** Cohesion
- 9) A design should contain distinct and separable representations of data and procedure.
- 10) A design should lead to modules (e.g functions or procedures) that exhibit independent functional characteristics.
** Coupling, Cohesion

11) A design should lead to interfaces that reduce the complexity of connections between modules and with the external environment.

** Coupling

12) A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis.

** methods, techniques

Design Review

- **Examines and evaluates design documents for completeness, correctness, quality, agreement with requirements.**
- **PRIMARY INPUTS are the outputs of the detailed design stage.**
- **PRIMARY OUTPUTS is an acceptance or rejection of the SDD, plus a list of issues or modifications to be handled.**

Running through the High Level and Detailed Design Phases are:

Data Design - transforms the information domain model created during the requirements analysis into the data structures that will be required to implement the software. (We cover this more later when we talk about Databases.)

Architectural Design - Defines the relationships among major structural components of the system.

Procedural Design - transforms structural components into a procedural description of the software.

Interface Design (sometimes) - establishes layout and interaction mechanisms for human-machine interaction.

Design - Methods of Design

What is design?

Software Design - is a process through which requirements are translated into a representation of software.

- Focus is on HOW the program will work.
- Try to avoid programming language and hardware specific details that affect HOW.
- Develop a program architecture and map requirements to portions of the architecture.

The Designer's goal -
produce a model or representation of an entity that will later be built.

Object Oriented Methodolgy: Terms, ideas, techniques
OO Concepts, OO Analysis, OO Design
Design Concepts, Design Methods (Structured)
Real Time Analysis and Design

What is designed:

Data

Architecture

Interface

Procedural

Object Oriented Design:

Find objects, factor into good classes, define class hierarchies and interfaces, make it reusable, modifiable.

OO Design architecture stresses object relations rather than flow of control

What is different: (Fichman, Kemerer)
(From SA)

Module Hierarchy representation
Data Definition Spec
Procedural Spec
End-to-end processing sequences
Class and Hierarchy definition
Operations assigned to classes
...

Bertrand Meyer suggests:

Decomposable – design method makes easier
subproblems

Composable – reusable

Understandable – by module

Continuity – coupling (local)

Protection – coupling (isolation)

To achieve:

Few, small, explicit interfaces. Information
hiding.

OOD:

Booch

Coad – Yourdon

Rumbaugh

Jackson – design a part of analysis!

Coad suggest (for any method): Define:

Problem Domain

User I/F

Task Management

Data Management

Design Patterns:

(This is another of the SE Buzz words!)

Recurring and reusing of classes and objects
(Gamma)

Name of pattern

Problem to which applied

Characteristics

Consequences of applying

Should one use inheritance or composition?

(When both possible)

Use what is general, or make it specific?

Compare SA, SD versus OO:

Hospital system: Patient record system.

Spreadsheet

Is OO Important?

OO Prog languages: JAVA, et al

OO Interfaces:
 Windowing systems

CORBA
 Distributed objects

CSE 5324: Software Engineering I

(Analysis, Design, Creation)

Apologies:

I'm back!

Review

Form teams for project

Design! Design, design

Preview

Why you should take CSE 5328

Exams

New stuff

What is important

Last class(es):

Basics of Design:

Concepts

Fundamentals

Structured design

Team Project:

If there was anyone that wanted to be a team leader and:

- Did not present last class OR**
- Wants to add anything to last weeks presentation**

Please do so now (1 minute per person)

**Teams should be formed and final by Thursday, October 7
if you are not on a team, I'll find a good team for you.**

CSE 5328 Software Development Studio

Class is about...

Instructor

What you need to know to get in

How class is done

Why you should take it

Exam 1 Statistics

Average **86**

Mean **84**

Median **82**

High **mid 90's**

Low **about 50**

1st Q **88**

3rd Q **75**

News (you can use)

"Last week NASA announced that the "human" error stemming from space engineers using two sets of measurements -- one utilizing miles and the other kilometers -- caused the loss of the Mars Climate Orbiter spacecraft last week, NASA said Thursday."

The teams, located at the National Aeronautics and Space Administration's Jet Propulsion Laboratory in Pasadena and at Lockheed Martin Astronautics in Colorado, complicated matters further by failing to realize the error, the agency said in a statement.

The \$125 million orbiter, intended to serve as the first interplanetary weather satellite, is believed to have broken up when it hit the Martian atmosphere last week after an approach that was too near the surface.

Today's question is:

What kind of error was this?

How could/(should) it have been prevented?

Review, and new stuff (use what ya' want)

Design - Methods of Design

What is design?

Software Design - is a process through which requirements are translated into a representation of software.

- Focus is on HOW the program will work.
- Try to avoid programming language and hardware specific details that affect HOW.
- Develop a program architecture and map requirements to portions of the architecture.

The Designer's goal -
produce a model or representation of an entity
that will later be built.

Why design?

Makes Implementation Mechanical
Different from Requirement or Code

How is design done:

Many methods:

Structured

OO

Etc.

NEED FOR DESIGN

Design is the only way we can accurately translate a customer's requirements into a finished software product

Without a design we risk building an unstable system:

- one that will fail when small changes are made (maintainability)
- one that may be difficult to test (testability)
- one whose quality can not be determined until late in the development process

Goal:

Define process (or system) in enough detail to
Implement. (Create, realize)

What is designed:

Data

Architecture

Interface

Procedural

Data design:

Wasserman:

“Select logical representations of data objects”

Should apply same analysis principles as applied to function

Identify data structures and operations performed on them

Data dictionary

Defer low-level data decisions

Hide data representation in modules

Develop library of data structures and operations
(reuse)

Use appropriate programming language

Architecture design:

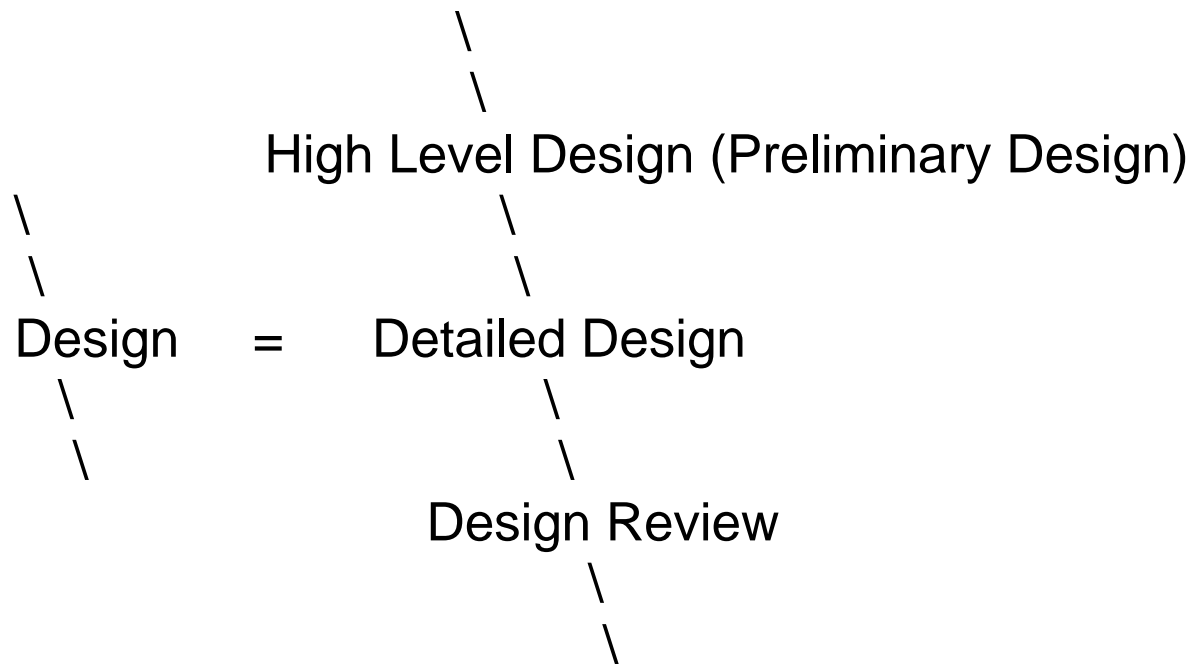
Develop modular program structure and show control relationships

This is very important: this shows the “outline” of the entire system.

DESIGN PROCESS:

The design for a system is created in stages.

From a project management point of view
(refinement of lifecycle)



(This started from “top-down” and “structured-programming” and “modular” ideas of late 1960’s and early 1970’s)

Data-flow oriented design:

Natural flow from analysis

Find type of information flow

Find flow boundaries

Map DFD to program flow

Factor control

Refine structure

Find type of information flow

Transform Flow
Transaction Flow

Transform Flow

Incoming flow -> Transform center ->
Outgoing flow

Overall flow is sequential, one of few paths
Look at DFD for few (one) straight line paths

Transaction Flow

Data Item triggers other flows

Select from many paths

Transaction->Transaction Center->

->Many

->Paths

In most systems BOTH flows are present.
(Find flow boundaries)

Map DFD to program flow

Transform Mapping – DFD with Transform Flow mapped to template for program structure

Design Steps:

Review Level 0 DFD (system spec and SRS)

Refine Data Flow (Show Details)

Is DFD transform or transaction?

Find and isolate transform center (find in and out flow boundaries)

Do a first level factoring
(top down distribution of control)

Do a second level factoring
(individual DFD bubbles to modules)

Refine using heuristics for design

Transaction Mapping:

Similar to above, BUT

Find transaction center

Factoring is on path basis

Afterwards:

- Need to describe processing for each module

- Describe Interfaces

- Local and Global Data structures

- Note Limits, Restrictions

- Review

- Optimize

Design Optimization:

- Knuths Law: “Don’t”

- Use CASE tools

- Analyze hot spots (time hogs), use appropriate

 - Algorithms

- Use appropriate programming Language

- Use instrumentation to find heavy loaded

 - modules: SW or HW

- Redo those

Interface Design:

Internal
External
Human

Some guidelines:

- Simplify information passing

- No globals

- Validate data

- Error handling and propagation: fix it,
or pass it up.

User Interface

- KISS

- WIMP (?)

Who are users?

- User Model

 - Experiences, other background

 - Novices, intermittent, frequent users

Design Issues:

HELP:

How can User get “Help”?

Is help available for all functions?

Which?

How to show help?

How to get out of help?

ERRORS:

How to show error messages

Provide some advice

Indicate any unexpected: (File lost, etc)

Don't blame User

COMMANDS:

Short cuts

Menus

Customization

Offer feedback (when slow, etc.)

Hints:

Iterate

Prototype

Use tool kits

USE IT!

User Interface Guidelines:

General:

- Be consistent
- Undo
- Make destruction difficult

Display:

- Show (only) important info
- Be consistent, predictable
- Compartmentalize
- Show analog displays (guage)

Data Input:

- Minimize
- Consistent
- Let user control flow
- Customize

Procedural Design:

Structured:

Sequence, condition, repetition

Graphical:

Flow Chart

Nassi-Shneiderman

Tables:

Decision Tables: (Like PLA's)

PDL (Program Design Language)

Pseudocode, structured English

Combined:

CSD (Control Structure Diagram)

(Mostly with CASE tools)

Why is design Important?

Is Structured Design the way to go?

Pro:

Popular

Fits with SA

Lots of Experience

Tools

The way you think

Flexible

Cons:

Like SA

Is OO more “natural”

Easy to omit, forget, etc. functions

Maintenance

Why Design at all?

High Level Design

- Also called Preliminary Design
- Concerned with the transformation of requirements into data and software architecture
- PRIMARY INPUT is the SRS
- PRIMARY OUTPUT is an architectural design and data design

Detailed Design

- Focuses on refinements to the architectural representation that lead to detailed data structure and algorithmic representations for software
- PRIMARY INPUTS are the outputs of the High Level Design step
- PRIMARY OUTPUT is the SDD (Software Design Description)
 - + contains architectural design (structure charts),
 - detailed design (module specifications)
 - data design (Design data dictionary)

and a Software Design Walkthrough document.
(We don't use here anymore)

STRIVING FOR QUALITY

Quality is an important objective of software design.

- Quality in the system really begins at the design stage
- Designs can be assessed for quality
 - One proven metric that can predict quality is $(\text{Fan-Out})^2$
i.e For a given module, as the number of other modules called by it increases, the complexity of the calling module is likely to increase, making that module more likely to contain mistakes.

Guidelines for Design Quality (Pressman)

13) A design should exhibit a hierarchical organization that makes intelligent use of control among components of software.

** Abstraction

14) A design should be modular; that is, the software should be logically partitioned into components that perform specific functions and subfunctions.

** Cohesion

15) A design should contain distinct and separable representations of data and procedure.

16) A design should lead to modules (e.g. functions or procedures) that exhibit independent functional characteristics.

** Coupling, Cohesion

17) A design should lead to interfaces that reduce the complexity of connections between modules and with the external environment.

** Coupling

18) A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis.

** methods, techniques

Design Review

- **Examines and evaluates design documents for completeness, correctness, quality, agreement with requirements.**
- **PRIMARY INPUTS are the outputs of the detailed design stage.**
- **PRIMARY OUTPUTS is an acceptance or rejection of the SDD, plus a list of issues or modifications to be handled.**

Running through the High Level and Detailed Design Phases are:

Data Design - transforms the information domain model created during the requirements analysis into the data structures that will be required to implement the software. (We cover this more later when we talk about Databases.)

Architectural Design - Defines the relationships among major structural components of the system.

Procedural Design - transforms structural components into a procedural description of the software.

Interface Design (sometimes) - establishes layout and interaction mechanisms for human-machine interaction.

Design - Methods of Design

What is design?

Software Design - is a process through which requirements are translated into a representation of software.

- Focus is on HOW the program will work.
- Try to avoid programming language and hardware specific details that affect HOW.
- Develop a program architecture and map requirements to portions of the architecture.

The Designer's goal -
produce a model or representation of an entity that will later be built.

Object Oriented Methodolgy: Terms, ideas, techniques
OO Concepts, OO Analysis, OO Design
Design Concepts, Design Methods (Structured)
Real Time Analysis and Design

What is designed:

Data

Architecture

Interface

Procedural

Object Oriented Design:

Find objects, factor into good classes, define class hierarchies and interfaces, make it reusable, modifiable.

OO Design architecture stresses object relations rather than flow of control

What is different: (Fichman, Kemerer)
(From SA)

Module Hierarchy representation
Data Definition Spec
Procedural Spec
End-to-end processing sequences
Class and Hierarchy definition
Operations assigned to classes
...

Bertrand Meyer suggests:

Decomposable – design method makes easier
subproblems

Composable – reusable

Understandable – by module

Continuity – coupling (local)

Protection – coupling (isolation)

To achieve:

Few, small, explicit interfaces. Information
hiding.

OOD:

Booch

Coad – Yourdon

Rumbaugh

Jackson – design a part of analysis!

Coad suggest (for any method): Define:

Problem Domain

User I/F

Task Management

Data Management

Design Patterns:

(This is another of the SE Buzz words!)

Recurring and reusing of classes and objects
(Gamma)

Name of pattern

Problem to which applied

Characteristics

Consequences of applying

Should one use inheritance or composition?

(When both possible)

Use what is general, or make it specific?

Compare SA, SD versus OO:

Hospital system: Patient record system.

Spreadsheet

Is OO Important?

OO Prog languages: JAVA, et al

OO Interfaces:
 Windowing systems

CORBA
 Distributed objects

CSE 5324: Software Engineering I

(Analysis, Design, Creation)

Exam 2

October 26 (Tuesday)

Covering:

Object Oriented Methods, particularly Analysis

Design

QA and Testing

Brooks: Chapters: 7 through 13

(Material covered since last exam)

Reading: Chap 4 (OO), 5, 6.2, 6.3 (Design),
7 ("Program" Testing)
NOT Chap 8 (will be in Exam 3)

What is the process?

"Our" lifecycle may be:

Waterfall (linear-sequential), Spiral, Incremental,
Formal, or other

But all include: Code Generation and Testing

What are our "goals" ? (What do we want from the
software development process?)

We want:

To be able to develop software quickly (calendar
months)

To minimize resource needs: (few(er) people,
for a shorter time - effort)

To create reuseable components (modules, designs,
even requirements)

To reduce costs of fixing defects in delivered software

???? Can we do this:

(For a 200 KLOC DP product)

CMM level 1 organization:

Duration 30 months (6.7 KLOC/Month)

Effort 594 Person Months

(340 LOC/Person/Month)

Total Cost \$5,440,000

CMM level 3 organization:

Duration 15 months (13 KLOC/Month)

Effort 79.5 Person Months

(2500 LOC/Person/Month)

Total Cost \$728,000

CMM level 5 organization:

Duration 9 months (22 KLOC/Month)

Effort 16 Person Months

(14,000 LOC/Person/Month)

Total Cost \$146,000

So we know how to do what we want (above)

(CMM)

Institute Basic Project Management (repeatable)

We define and document processes

Measure and manage quality and productivity

Optimize and continually improve processes

We have seen the costs of fixing defects increase later in the development process

To minimize the costs, fix defects early. How?

Even if the costs are minimized, if the product is not high "quality" it won't meet requirements
(Or won't be used, or won't sell)

What is quality?

Which is better:

Cars:

Yugo, (Trabi, Maruti, etc) OR

Cadillac (Saturn, M.Benz, Volvo, etc.)

Why?

Who makes good computers?

Who makes good electronics, cameras?

Who makes good software?

Why? How do you determine?

(Who makes bad software?)

How can we determine "quality"?

The (old?) American Car Companies method:

"We getting better all the time"
(Don't change anything...)

The Saturn (Japan, from older US) method:

Measure now (metrics)
Improve the metrics

Quality can be:

Speed
Capacity (sizes, etc.)
Number of defects
Kinds of defects ("cosmetic", or lose data)

V and V

Validation: Are we building the right product?

Verification: Are we building the product right?
(Boehm)

Quality control is: inspections, reviews, and tests
with feedback.

Quality Assurance

Auditing and reporting aspects of management.

Is quality free?

(There was a book by this name) NO.

What are the costs of quality?

Building in quality:

(Prevention)

Planning

Formal Technical Reviews

Training

Special software, equipment

Appraisal:

Inspection

Testing

Costs of failure (defect):

(In house, before shipping)

Rework

Repair

Analysis of failure

Missed deadlines

Low morale

Dismissing employees (firing, killing)

(In the field, after shipping)

Product Return

Canceled sales

Penalties

Help desk support

Warranties

SQA

Based on requirements
(What else?)

Standards and development steps measured

"Implicit requirements" - maintainability, useability

In other industries:

Bell Labs in 1916, formal QA
Many other "test" groups

In software:

Was the responsibility of programmer
(small systems)

Now?

Independent test groups

SQA (needs to:)

Prepare SQA plan

Participates in development of software process description for a project

Reviews SE for compliance

Audits work (for compliance)

Documents deviations

Records and Reports noncompliance

Software reviews:

Use other people (diverse group) to:

Suggest improvements

Eliminate unnecessary work

Make product: more manageable, predictable,
uniform

What are defects?

Errors, faults, bugs, bugettes

As defects occur in a phase, if they are not removed
in that phase (step), they are "amplified" (effort, cost)
in the next and latter phases (steps)

Formal Technical Reviews:

(walkthroughs, inspections, reviews)

Goal: Find errors, verify requirements, follow standards, training

FTR:

3 to 5 people
peers, no managers

Advance preparation required, no more than
2 hours

Meeting less than 2 hours

FTR guidelines:

review product, not person

Set agenda, follow it

Limit debate and response

Don't solve problems

Take written notes

Limit participants and prepare in advance

Schedule resources, etc

Training for reviewers

SQA

(Use statistics)

Collect and categorize defect information

Trace underlying cause

Pareto principle (80 % defects in 20% causes -
modules) identify which

Correct those

$MTBF = MTTF + MTTR$

Availability (per cent of time)

Safety and hazards: how dangerous

SQA plan: what documents?

ISO 9000 (ISO 9001, ISO 9000-3)
what must be done, not HOW

Software Testing

You can not prove that no defects are present, only that there are defects.

What is the objective of testing?

Finding errors. (defects, bugs)

Guarantee quality

There are many methods for testing:

What do exams test?

Statistical sampling testing

One can test for "related" artifacts (markers)

Some chemical testing is done by testing for by-products

Mining tests for other products usually found with what is being searched for.

How to find (test) suitability of novia?
(girlfriend, boyfriend, significant other, etc.)

Go to a restaurant.
Watch how waiter is treated.
That's you, in a few months.

Testing is not so easy:

Specification for Sort:

Input specification:

p: array of n integers, $n > 0$

Output specification:

q: array of n integers such that:

$q[0] \leq q[1] \leq \dots \leq q[n-1]$

Implementation:

```
void Sort ( int p[ ], int q[ ] )  
{  
    int l;  
    for ( l = 0, l < n, l++ )  
        q[ l ] = 0;  
}
```

Specification is wrong!

The elements of the array q are a permutation of the elements of p , which are unchanged

Even when the specification is correct

Ask user for two numbers, add them together, print results

How big are numbers? What base? Negative? Etc.

An example implementation:

```
Int A, B, Sum
```

```
Read A, B
```

```
Sum = A + B
```

```
Print Sum
```

Exhaustive Testing

All possible values for A and B

assuming a 64 bit CPU

A and B are 0 to $+2^{63}-1$ (-1 to -2^{63})

even at 1000000 tests per second,
this takes years.

What are Objectives of testing

Glen Myers

Testing is process of executing program with intent of finding error

Good test case high probability of finding as yet undiscovered error

Success is finding as yet undiscovered error

WE WANT TO FIND ERRORS (not avoid them)

Principles by Davis

Testing traceable to requirements

Planned long before testing starts

Start "in the small" then go larger

Not possible to exhaustive test

Effective - Third party tests

How testable is a program (system)?

Operability

- The better it works, easier to test

- Defects block testing

- Defects slow process: reporting, analysis

Observability

- Test what you see

- For an input, distinct output

- Can query state

- Logging

Controllability

- All outputs can be generated by inputs

- All code can be tested by inputs

Simplicity

- Small systems more testable than large

- Functional simple

- Modularity of code

Stability

Change disrupts testing

Change invalidates previous tests

What is a good test?

Kaner, et al

High probability of finding an error

Not redundant

Best of its type

Not too simple or complex

Why not just "prove" correctness?

The sad case of "correctness proofs"

The programmer should let the proof and the program grow hand in hand.

Dijkstra

Naur (1969) published a paper on constructing and proving a product correct.

The line editing problem:

Given text separated by blanks or newline,
convert, line - by - line to:

line breaks only where blank or new line

line filled as far as possible

no line more than maxpos characters

Data:

This is an example
of some
lines of text that can be very very very long

Program was 25 lines of Algol

Leavenworth in 1970 found first line is
preceded by blank unless exactly maxpos
char long.

Test data would have shone this

London in 1971 found 3 more errors

London published corrected proof

Goodenough and Gerhart found 3 more
errors (testing would find)

Software Testing

(Part 2)

What is the objective of testing?

Finding errors. (defects, bugs)

Guarantee quality

How can one test?

Designing test cases can be as difficult (or more) than implementation.

Many (most) products can be tested in two ways:

Black Box

when we know the function of the product

we can test each function

when this is software - this is testing at the interface level.

(don't need to know internals)

White Box

when the internal structure is known
we can test that all parts are exercised
for software testing paths through logic
(based on internals)

White Box testing can be impractical:
Exhaustive testing

Both methods are important (but not equally used)- different types of errors are found

White Box (or clear, transparent or glass box)

what type of errors are common?

logic errors occur more frequently in less used paths (special cases are "special" and more difficult)

we often miss-predict which paths are commonly followed, program flow is often counterintuitive

typographical errors are random
(`l = j; ++k;`) syntax checks won't find semantic errors (some languages better

than others - "C" vs ADA)

A Program to count the words in "standard input"

```
procedure CountWords;
```

```
var
```

```
    nw: integer;
```

```
    c: character;
```

```
    inword: boolean;
```

```
begin
```

```
    nw := 0;
```

```
    inword := false;
```

```
    while (getc( c ) <> ENDFILE ) do
```

```
        if ( c = BLANK ) or ( c = NEWLINE ) or ( c = TAB )
```

```
            then inword := false
```

```
            else if ( not inword ) then begin
```

```
                inword := true;
```

```
                nw := nw + 1
```

```
            end;
```

```
    putdec( nw, 1 );
```

```
end;
```

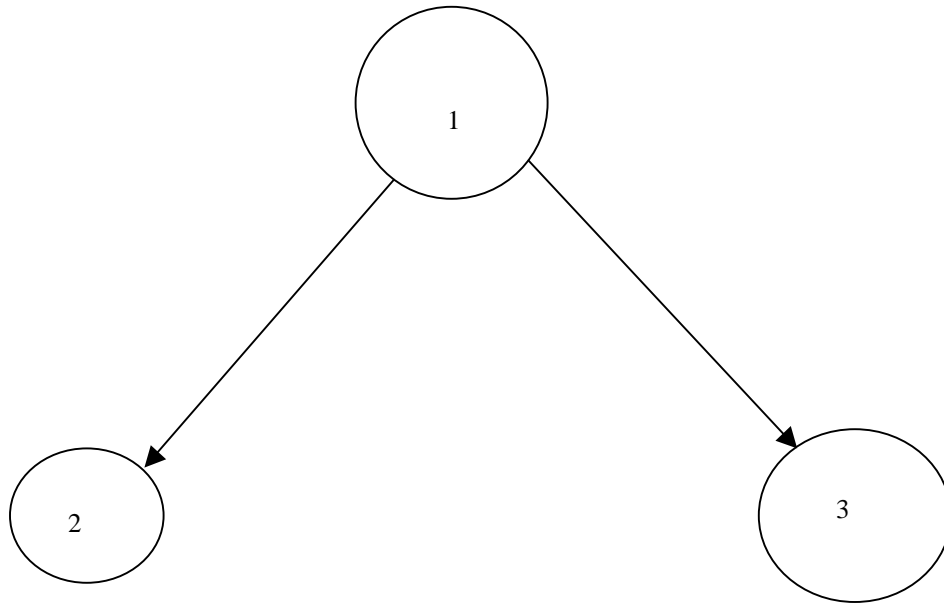


```
1. procedure CountWords;
2. var
3.   nw: integer;
4.   c: character;
5.   inword: boolean;
6. begin
7.   nw := 0;
8.   inword := false;
9.   while (getc( c ) <> ENDFILE ) do
10.     if ( c = BLANK ) or ( c = NEWLINE ) or (c = TAB )
11.       then inword := false
12.     else if ( not inword ) then begin
13.       inword := true;
14.       nw := nw + 1
15.     end;
16.   putdec( nw, 1 );
17. end;
```

Some White Box testing methods

Basis Path Testing

A flow graph:



Nodes - 1, 2, 3

Edges - 1 ->2, 1 -> 3

Regions - Bounded Nodes by edges

Predicate Node - 1

We want to find a basis set of paths, a test based on this basis set executes each statement at least once

Cyclomatic Complexity

A metric that quantitatively measures logical complexity

The number of independent paths in the basis set

Paths:

path 1: 7,8 -> 9 -> 9a -> 10a -> 10b -> 10c -> 12 -> 9 ->
15 -> 16

path 2: 7,8 -> 9 -> 15 -> 16

etc.

should not be combinations of other paths

Basis sets are not unique

If we force execution of each path in basis set

every program statement is executed at least once

How many paths are there?

Cyclomatic complexity

$$V(G) = E - N + 2 \quad (\text{Edges, Nodes})$$

or

Number of Regions

or

$$V(G) = P + 1 \quad (\text{Predicate Nodes})$$

$V(G)$ is upper bound of independent paths
upper bound of number of tests to be designed
and executed

Deriving test cases:

Use design (or code), draw flow graph

Determine $V(G)$

Find basis set of linear independent paths

Prepare test cases to force execution of each

Can use "Graph Matrix"

	NODE				
	1	2	3	4	5
1		1			
2			1		1
3				1	1
4					
5					

If use 0, 1 the connection exists or not
may weight with probability
processing time, other resource use

Testing at control structure

Condition testing

Expression relational-operator Expression

Errors include: Rel-op error, arithmetic error, other

test each condition in program

by: branch testing

domain testing (domain of boolean,
var states)

Data Flow Testing

Select Paths according to definition and use of variables

Def - Use

DU chain:

Where variable is defined, first set, used,
span where in use ("live"), last used

Loop testing

Simple loop test must

- skip the loop

- make exactly one pass

- two passes

- m passes, $m < \text{limit}$

- limit, limit+1, limit-1 passes

Nested Loops

- Start at innermost loop

- test holding outside loops at minimum val.

- work outward

- until finished

Spaghetti loops (unstructured)

- Re-do them

Black Box Testing:

Sometimes called: Behavioral or partition testing

Find:

- Wrong or missing functionality

- Interface errors

- Data Structure Errors

- Initialization, Termination Errors

Meyer says:

- Test cases should reduce need for additional test cases, tell us about classes of errors, not just single errors

Graph based

Graph of objects and relationships

Nodes, node weights, links, link weights

Beizer describes

Transaction flow modeling

Finite State modeling

Data Flow

Timing modeling

May or may not be symmetric

Test cases cover: Nodes, Links

Equivalence Partition

Input domain is divided into classes

Input in ranges: valid parts of domain and invalid
domain: 5..10 is valid, 0..4, 11..100 is invalid

Input generated into each equivalent partition

Boundary Value

Very Common

Find range, test just above and below

Comparison Test

Create several different implementations

Test and compare

What are interesting to test:

GUI

Client/Server

Distributed

Documentation, Help

Real Time

What does the "real world" do?

Probably the wrong thing.

Software Testing

Testing Approaches (Strategies)

What is the goal of testing?

(Finding errors.)

Assess quality

Uncover errors

Ensure standards are met

"You can't test-in quality"

(Another sad story)

The story of The Tandy Computer Back Up

Tandy Company used to manufacture computer systems, and provided system software (They made an OS, utilities, and software to backup the disk)

The backup software would backup (copy) files to floppy disks and cost \$129.99

The box for the software clearly reminded users to back up their files often, to avoid any data loss...

...

Testing is an important part of Software Quality

But not all:

Also included are:

Formal Technical Reviews

Software Engineering Methods

Standards and Procedures

Configuration Management and SQA

Metrics

All testing follows:

Begins at the module (or object) level and
works upwards

Different techniques are appropriate at different times

Testing is done by developer and independent group

Testing is not debugging

Validation and Verification

Verify: Building product right

Validate: Building right product

Boehm

There is a conflict when producers test their own product

They don't want to find errors

"Independent" Test Group

Fallacies:

Developers do no testing

Software test is hidden from developers

Testers start testing when code complete
(or schedule tells them to)

Software testing follows lifecycle (backwards)

Unit Test

Focus on implementation - source code
Heavy use of white box testing

Integration Testing

Design Oriented
Black Box mostly, some White Box

Validation Testing

Requirements Oriented
Black Box
Does system meet: functional, behavior,
performance requirements

System Test

(Not really SE)
Does overall system work

When is testing finished?

The question that all managers, developers
and testers ask

"When the schedule says so"

When we run out of time, when there are no more
bugs

An SE answer

Musa and Ackerman:

Can not be certain software does not fail but based
on a model with 98 % confidence the probability of
one year of failure-free operation is probabilistically
0.95

The logarithmic Poisson execution time model:

$$f(t) = (1/p) \ln(l_0 pt + 1)$$

$f(t)$ is cumulative failures over time t

l_0 is failures per unit time (failure intensity)

p is exponential reduction in intensity as repairs are made

Instantaneous failure intensity, derivative of $f(t)$

$$l(t) = l_0 (l_0 pt + 1)$$

Can use this to predict error rate drop off

What is error rate increases

Gilb says success is:

Specify quantifiable product requirements before testing

State testing objectives explicitly

Understand users

Rapid Cycle Testing - fast feedback

Build robust software

FTR

Continuous improvement of testing process

Testing:

Units:

Interface:

Parameters match? Units match?
Number and order of parameters?

Files:

Open, close OK? Buffer sizes,
formats? EOF?

Data Structures:

Typing? Initialization or default?
Under/Over flow?

Drivers and stubs written

Integration Test:

Top Down

Depth first

Breadth first

Bottom Up

Atomic modules built to clusters

Regression testing

Use test suites

Validation testing:

What are "reasonable" expectations
Needs to be documented!

Configuration reviews (Audits)

Alpha test

Developers site by customer

Beta test

Customer site by end users

System tests:

Recover from faults

Security

Stress (over stressed)

Performance

Debugging

Brute force
printf

Backtrack
know where fault is, inspect code

Reason it out (cause elimination)
divide and conquer

IBM:

Mills found removing 60% of defects would lead to 3% reliability improvement

Terms:

Failure - when executing

Fault - static, built in

Probability of Failure on Demand

Prob. system fails when request is made

0.01 1 in 1000 service requests on avg. fail

When useful?

Rate of failure occurrence

frequency of unexpected behaviour

2/100, 2 times per 100 time units

MTTF - Mean time to Fail

Availability

0.96 in 100 hours available 96 hours

Failures:

- Transient (only some input)
- Permanent
- Recoverable (no operator needed)
- Corrupting (system state or data)

Reliability:

- Fault Avoidance

- Fault Tolerance

- Fault Detection

CSE 5324: Software Engineering I

(Analysis, Design, Creation)

SQA and Testing

Questions:

What do pipes and filters have to do with design?

Coupling and cohesion?

OOA?

Fan-in, fan-out, arch. design

SQA and testing:

Faults, failures, bugs, and errors

Classification: HP, IBM

What is being tested: steps

When do you stop?

CSE 5324: Software Engineering I

(Analysis, Design, Creation)

How much testing is needed?

When to stop?

Seeded faults (counting fish in a lake)

Confidence

"With 95% confidence level MTBF is 1000 hours"
(for this module)

When 100% coverage is done

Inspections are good, testing still necessary

Testing the system is:

System functional requirements - Function test
(Functioning system)

Speed, reliability, etc. - Performance test
(Verified and validated software)

Customer requirements - Acceptance test
(Accepted system)

Users Environment - Installation test
(Alpha, Beta testing)

Configuration management

(Tools, documents, etc.)

Version (control) - releases

Production vs. development systems

Regression testing

"Deltas", change control, SCCS

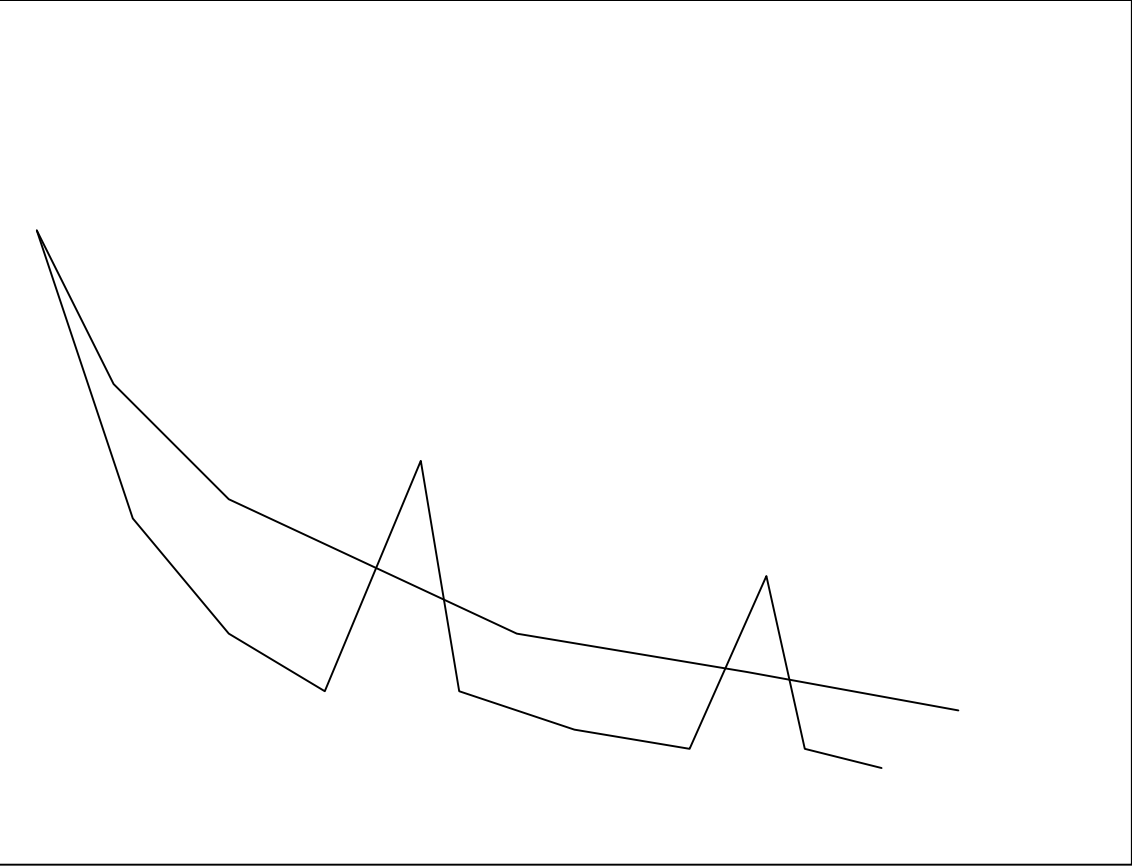
Who should test?

Professionals? You? When?

Stress testing

Reliability / Availability / Maintainability

Time / fault model(s)



Motorola zero-failure

Based on:

at time t number of failures

$ae^{-b(t)}$, where a and b are constants

test to a required reliability without finding failures,
then stop

OR - reset clock and keep testing

Test plans have:

Objectives

References

Tests

Schedules

Tools, etc. needed

Analysis

Report forms (bug reports)

Safety Critical

Cleanroom

CSE 5324: Software Engineering I

(Analysis, Design, Creation)

How much testing is needed?

When to stop?

Seeded faults (counting fish in a lake)

Confidence

"With 95% confidence level MTBF is 1000 hours"
(for this module)

When 100% coverage is done

Inspections are good, testing still necessary

Testing the system is:

System functional requirements - Function test
(Functioning system)

Speed, reliability, etc. - Performance test
(Verified and validated software)

Customer requirements - Acceptance test
(Accepted system)

Users Environment - Installation test
(Alpha, Beta testing)

Configuration management

(Tools, documents, etc.)

Version (control) - releases

Production vs. development systems

Regression testing

"Deltas", change control, SCCS

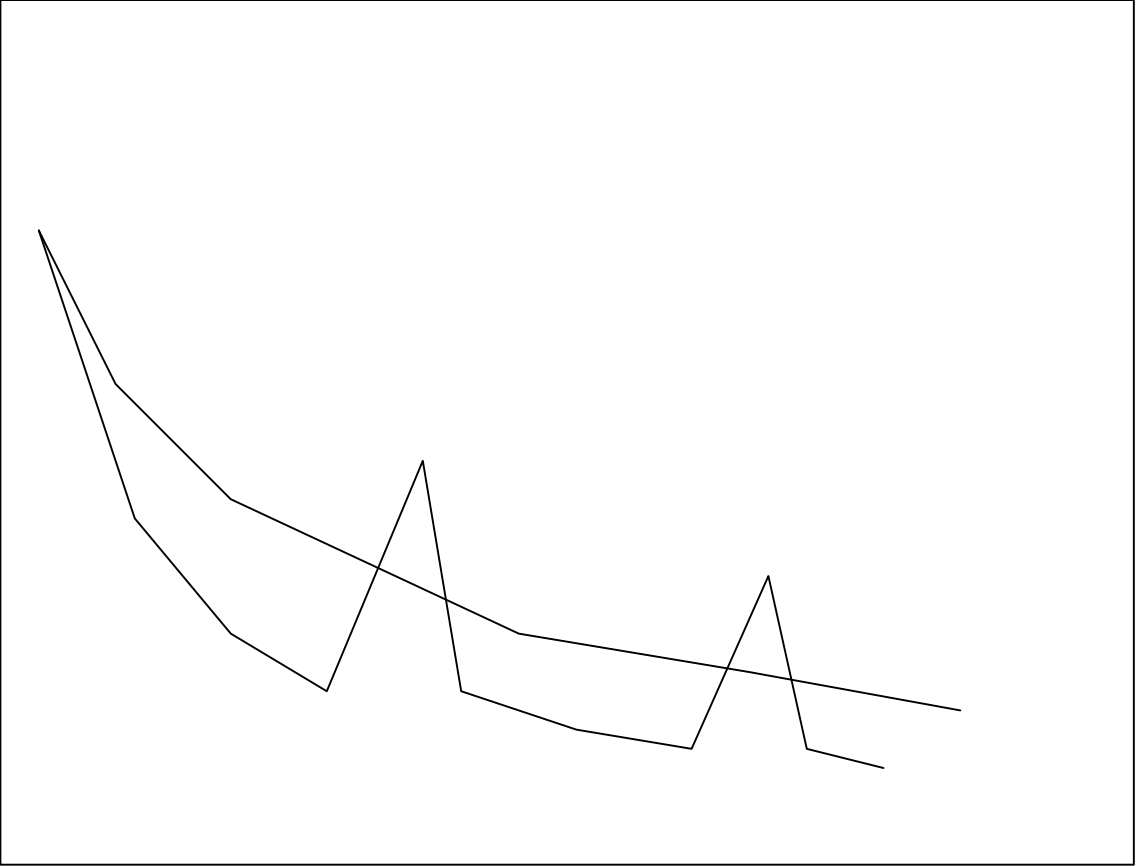
Who should test?

Professionals? You? When?

Stress testing

Reliability / Availability / Maintainability

Time / fault model(s)



Motorola zero-failure

Based on:

at time t number of failures

$ae^{-b(t)}$, where a and b are constants

test to a required reliability without finding failures,
then stop

OR - reset clock and keep testing

Test plans have:

Objectives

References

Tests

Schedules

Tools, etc. needed

Analysis

Report forms (bug reports)

Safety Critical

Cleanroom

CSE 5324: Software Engineering I
(Analysis, Design, Creation)