Basics of Software Design

Seeing the Big Picture

— One Slice At A Time

Greg Butler

Summary — Low Level & Concrete

Design decomposes a system into simpler parts.

A design must describe
- Components (and their properties).
- Connections between components.
- How the parts work together as a whole.

A design can be checked by tracing scenarios of use.

Major Problem: Describing a part without being too detailed.
Solution: Specifications.

Designers use models for communication and feedback, and to "trace" parts to the whole.

Design must address all concerns, not just function.

Separation of concerns: address one concern at a time
- Different models for different concerns.
- All models must form a consistent whole.
- Issue-based design makes design rationale explicit.

For large-scale systems, architecture is important.

Summary — High Level

Problem for Design: Multiple, conflicting concerns.

Solution: separation of concerns
- New problem: keeping big picture of separate views

Solution: trade-offs
- New problem: basis for trade-offs
- New problem: consistency of trade-offs

Problem for Design: Complexity of system.

Solution: abstractions and structure
- New problem: managing web of dependencies
- New problem: relating models from separate views

Problem for Design: Evolution of software systems.

Solution: design for change
- New problem: crystal ball to anticipate changes

Example: Design a Toaster

Design creates a plan for manufacturing the product.

It describes the components, their dimensions, their layout, their attachment to each other, the materials, etc

Components:

- Chassis or base:
- Slots for bread: big enough for "toast" bread
- Heating elements: 120 Volt, 60Hz
- Power cord:
- Cover:
- Control switch:
- Automatic "pop" mechanism and switch (?)

Note that Chassis may consist of even smaller parts.

Connectors:

- Rivets: holding Chassis subparts together
- Screws or catches: holding Cover to Chassis
- Power cord: connector for outlet to heating elements

Toaster design described by
engineering drawings of components
specifications for materials
"exploded" drawing shows components fit together

Lesson: *Design decomposes a system into components and connectors*

Software components are subsystem, module, unit (or object).

Unit is an individual routine, procedure, function

Module is a provider of computational resources or services

Subsystem is subset of the modules making up a system.
also a provider of services.

Connectors are procedure calls, shared data, middle-ware, ...

the *interfaces*

Lesson: *For software design, components have function*

Function is computed by algorithms.

Algorithms manipulate data structures.

Lesson: *Design is described by models*

Function can be modeled by specifications
(eg, pre- & post-conditions)

Algorithm internal details modelled by pseudocode.

Relationship of parts to whole can be modeled by *structure charts* (for example)

Example — Division

Informal description of function:

given M and N, find the quotient and remainder when M is divided by N.

Specification of function:

{ Pre: $N \geq 0$ and M are integers }
{ Post: $(M = QN + R) \wedge (0 \leq R < N)$ }

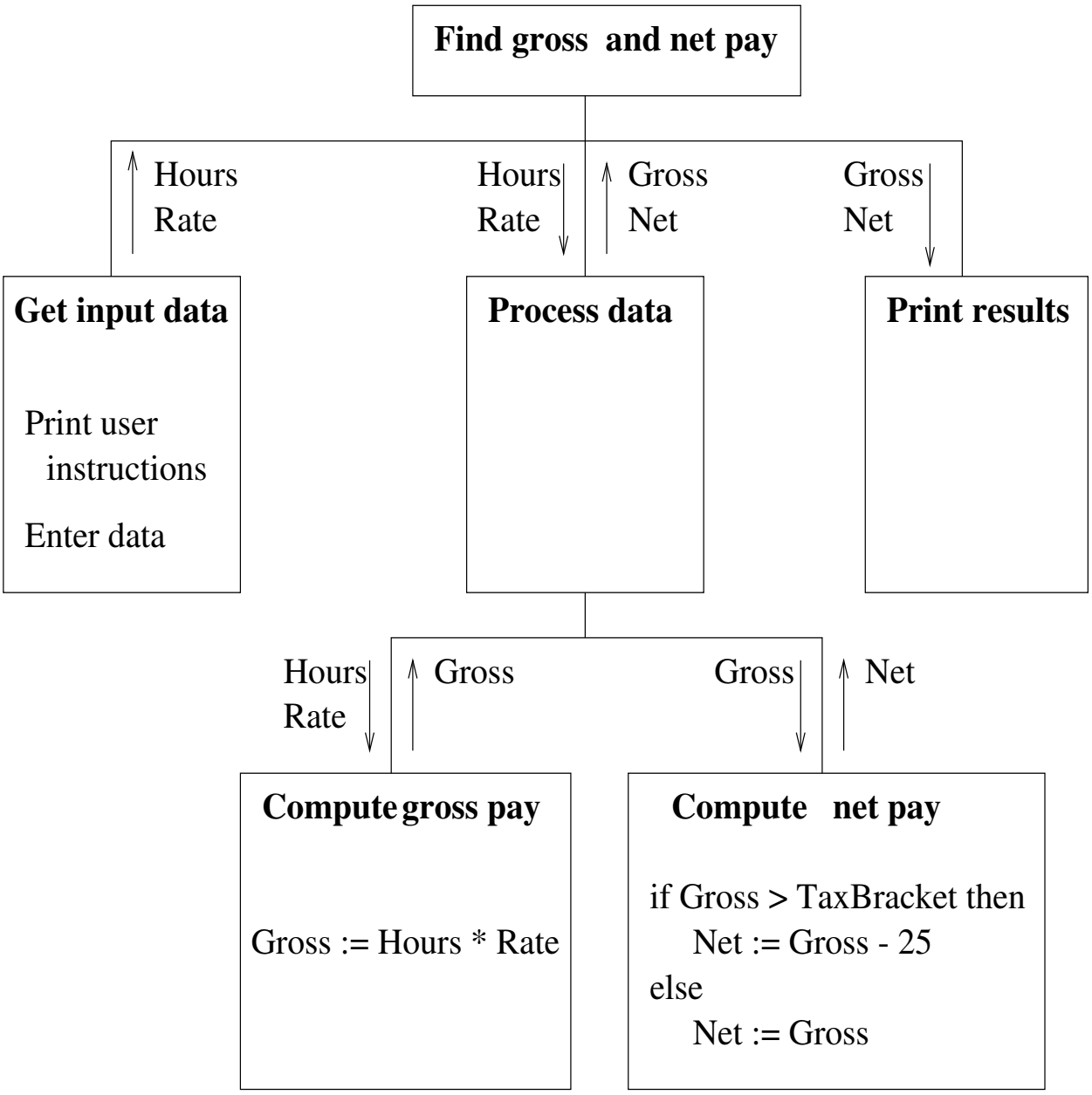
Algorithm informally:

subtract N from M repeatedly until the subtraction would yield a negative result. Then Q will be the number of subtractions performed and R is what is left over when we can no longer subtract N.

Pseudocode description:

```
1.  R := M;
2.  Q := 0;
3.  if R < N then go to step 7;
4.  R := R − N;
5.  Q := Q + 1;
6.  go to step 3;
7.  stop
```

Structure Chart and Pseudocode for Payroll Problem



Validating a Design — Function

You have a design of a software system, how do you check it?

1. Module or algorithm

Answer: Trace scenarios for sample inputs

Step through algorithm using input values

Confirm that result matches expectation

Answer: Formal analysis of algorithm

2. That parts work as a whole

Answer: Trace scenarios of user interaction with the system

This also includes sample input values

Tracing propagates to checks of individual modules

Individual modules produce output values

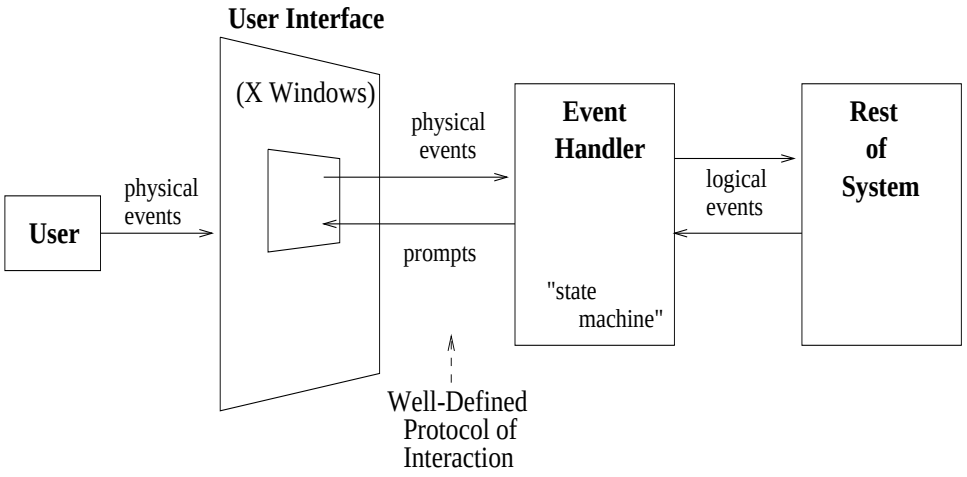
Output values used by other modules

But how to create a design?!

Simplest approach: use other people's experience

Toaster: Adapt an existing toaster design

Software: Adapt an existing design, or standard architecture such as "interactive interface"



Software: Use libraries of existing algorithms, data structures, GUI widgets, etc

Example: Design a Toaster

What things should design take into account?

Function — what it is meant to do

- egonomics & usability
- efficiency

Maintenance — how to clean it, service it, repair it

Aesthetics — is it "attractive", marketable

Construction process — how is it to be built

Cost

Lesson: *Design Addresses Multiple Goals*

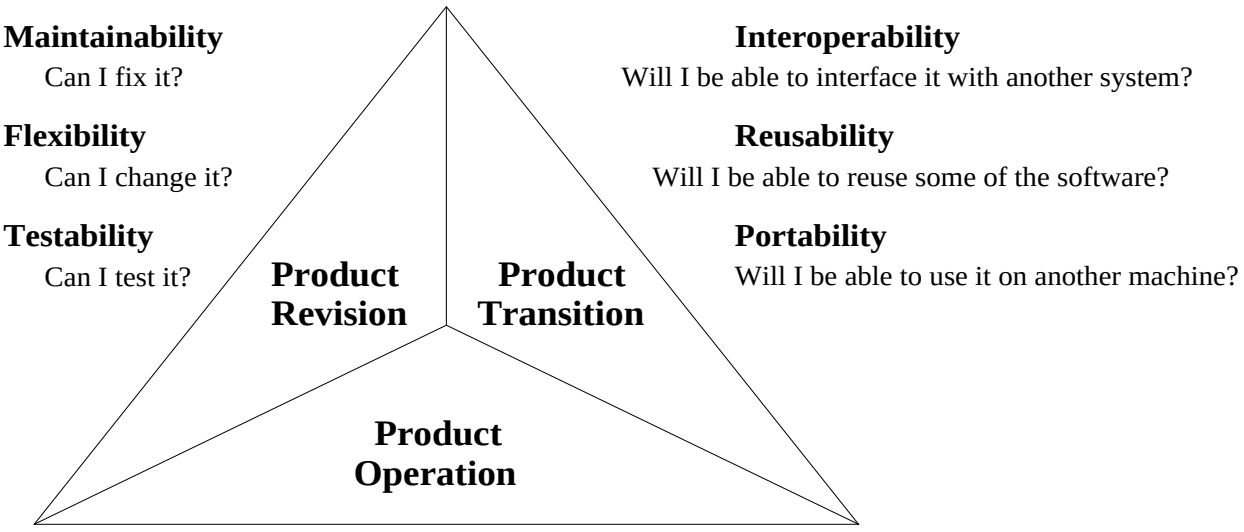
Usually *more* concerns than a person can juggle!

$\Rightarrow$ *separate concerns* and focus on **one** at a time

Lesson: *Consider Product & its Life-cycle*

The concerns (for software) are often classified into three categories:

Life-Cycle View



Correctness	Does it do what I want?
Reliability	Does it do it accurately all the time?
Efficiency	Will it run on my hardware as well as it can?
Integrity	Is it secure?
Usability	Can I run it?

Product Utility View

Conventional Wisdom for Software Designers

— of small software systems:

Focus on **function**

focus on *correctness* first

tune program to address *efficiency*

separate user interface from core of system

can address *usability* separately on UI

— of medium-scale software systems:

Design for **change**!

Validating a Design — Non-Functional Concerns

For function, we look at scenarios of use

For **product revision**, create a scenario of possible revision and then ask "what if"

Flexibility: Suppose I had to add tax calculations to my payroll system, what would be the impact on each module in my design?

Develop a scenario of proposed changes

Step through each proposed change

Determine the impact of the change on modules

For **product transition**, create a scenario of possible transition and then ask "what if"

Portability: Suppose I had to port my payroll system from IBM AS/400 to Windows, what would I have to do?

Develop a scenario of proposed steps to do the port.

Step through each proposed change

Determine the impact of the change on modules

If the results are not satisfactory, then the scenario has raised some **issue** to be resolved

⇒ leads into issue-driven design and design rationale
⇒ leads into design patterns

Example: Design a Knife

Function — to cut

Design is *simple*

- a *blade* with an edge sharpened to cut
- a *handle* ergonomically fitting the hand

Yet there are **many different** shapes and styles of knives!

Each has a slightly different **use**:

same function but different contexts

Lesson: *There are many good designs*

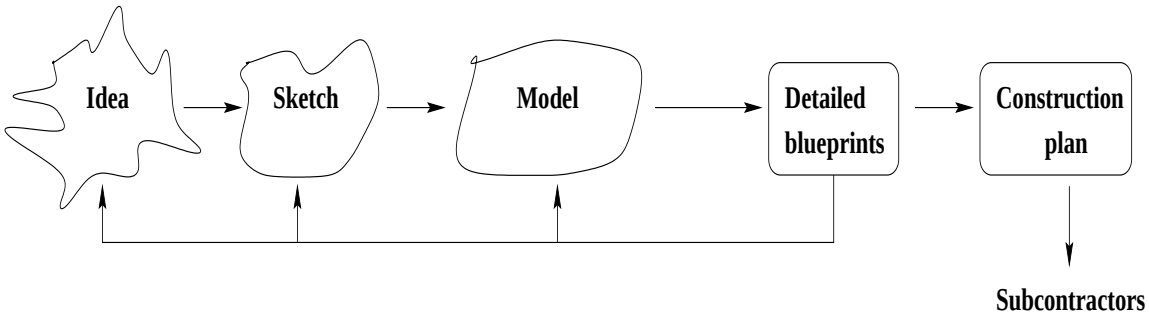
Design is a trade-off between conflicting goals, many different trade-offs are possible!

Lesson: *Know as much about use of product as possible*

The evaluation of a trade-off is based on the

- actual *context* of **use** of the product, and
- *priorities* or *preferences* of the **user** or **customer**.

Example: Design a Building



Design may be *complicated*

Design *process* is complicated

- much feedback to check with customer, other designers, engineers, etc
- models (both paper and non-paper) built to aid feedback and communication
- several kinds of models — different perspectives
- analysis of design by structural engineers, etc
- reviews that relevant regulations are met

Building architects have difficulty knowing **function** and **use** of a building, and designing appropriately for them.

Lesson: *Design needs models and multiple perspectives*

Lesson: *Design needs feedback and iteration*

Lesson: *Formal analysis of models is useful*

Lesson: *Design reviews can catch expensive mistakes*

Lesson: *Large-scale design demands an* **architecture**

Conventional Wisdom for Software Designers

— of large-scale software systems:

1. Get the **architecture** right!

The Bad News: it is essential that the architecture address **each** of the **concerns**.

2. Maintain the architectural vision!

So What's Different About Software Design

Function is a computation: virtual machine, algorithms, data structures

Complexity of inputs, function, ... demands

Variety of software systems — one size of design does not fit all!

Lack of standards

Rate of technological change

Lack of experience, experiment, results, ...

Sheer size and numbers, kinds of dependencies, ...