On the Nature of Software Design

1. What is software design?
2. What is *good* software design?
3. Can good design be taught?

Related Questions

1. What is the appropriate role of structure in human endeavour?
2. Is marriage a bourgeois and outdated institution?
3. How should I raise my children?
4. What *rights* ought an individual living in society posses?
5. What *obligations* ought be required from an individual living in society?
6. Why is my `#%@^!&^@`-ing committee being so difficult?

Sidebar

Definition

A problem is said to be P-hard, or philosophically hard, if it is reducible to the question:

"What is the meaning of life?"

Conjecture

All interesting problems of software engineering are P-hard.

Conjecture

P-hard $\equiv$ BS-soft

What is Software Design?

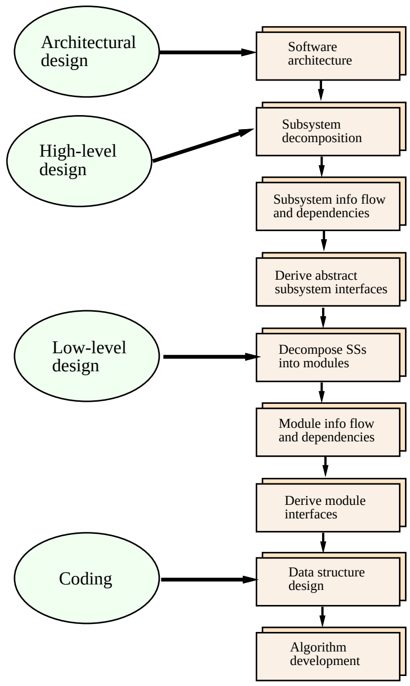
Some possible answers:

- the *structure* of a solution to a problem, *how* to implement the *what*:
  - → a *process* or methodology one follows to develop software
  - → an *artifact*, a high-level view of the code
- how the system *appears* to various people:
  - → system developer
  - → user interface look-and-feel
  - → re-user of system
  - → sys-admin

Attributes of Approaches to Software Design

- repeatability
- scrutability
- tolerance of creativity
- flexibility, modularity, adaptability, re-usability:
  - → when needs change
  - → when things go wrong
- dealing with scale

The Basic Phases of Software Design



How It *Appears* to be Done

- top-down <u>vs.</u> bottom-up
- functional decomposition <u>vs.</u> data-centric
- object-oriented
- role of documentation, paper trails, fakin' it:
  - → Parnas and Clements:

> *Designing a software system is a lot like proving a mathematical theorem.*

What is *Good* Software Design?

Related questions:

- What is a good person?
- What is a good book?
- What is a good car?
- What is *quality*?

  [Ref: *Zen and the Art of Motorcycle Maintenance*]

Design Quality: What are Your Values?

- end-system performance
- fastest/easiest to implement
- easiest to maintain, "evolve", port
- easiest to install, configure, connect
- reliable, robust, secure
- easiest to use
- flashy, fun, addictive
- easiest to advertise, sell

What does "quality" mean to:

→ Iomega, Microsoft, Netscape, Intel, IBM, FAA, DoD

The Big Software Engineering Texts Say . . .

Many design methodologies define various "quality metrics" for software design. Mostly, these are based on ideas of *coupling* and *cohesion*:

→ *high cohesion* means "related" parts are in the same container

→ *low coupling* means the big containers have fewer interdependencies

Other common design criteria:

- hierarchical, decomposable
- modular, information-hiding, small & simple interfaces
- separable, independent, "like things together"
- limited shared "special knowledge"

Conflicting Goals

- cohesion, coupling vs. object-oriented design
- efficiency vs. safety, portability . . .
- opaque types vs. understanding of grungy details (*e.g.,* `sizeof`, bit ordering)
- robust vs. fragile representation
- parameter passing vs. shared state
- explicit interfaces vs. implicit shared context
- overhead vs. efficiency in accessing data
- hierarchical abstraction vs. flat ontological plane
- defined protocols/access permissions vs. anarchy

Creative Design

- Well-defined methodologies stifle creativity, slow you down, can't see the forest through the trees
- Well-defined methodologies entail repeatable results

Creativity is not always a goal!

- profitable, well-understood cash cows lay golden eggs
- DoD projects, nuclear reactor software, traffic light controls:
  → main goals are reliability, robustness, security, scrutability, traceable system evolution

Can Good Software Design be Taught?

- well-defined methodologies
- creative design
- good engineering practice

How to Design Creatively

-
-
-
-
-
-
-
-

What People Say About Creative/Innovative Design

- It's a black art.
- It can't be taught.
- It's hard to measure objectively
  "I'll know it when I see it."
- Experience is necessary but not sufficient.
- Often, truly original design comes from a single person's vision.

User Interface Design

Schneiderman's Eight Golden Rules:

1. Strive for consistency
2. Enable frequent users to use shortcuts
3. Offer informative feedback
4. Design dialogues to yield closure
5. Offer simple error handling
6. Permit easy reversal of actions
7. Support internal locus of control
8. Reduce short-term memory load

Software "Engineering"

- term coined as wishful thinking
- still very young
- training/competence levels of practitioners varies wildly
- obsession with short-term achievement
- Technological Peter Principle

"Real" engineering:

→ attacks real, practical problems

→ reasonable solutions

→ based in science

→ codified knowledge

→ professional responsibility, "iron ring"

Software Design: Some Books

Some different views:

Design as high-level coding:

- *Large-Scale C++ Software Design* by John Lakos
- *Structured Design* by Constantine and Yourdon
- *Object-Oriented Analysis and Design With Applications* by Grady Booch

Design from the user's perspective:

- *The Design of Everyday Things* by Donald Norman
- *Bringing Design to Software* ed. by Terry Winograd

Design for special kinds of systems:

- *Safeware: System Safety and Computers* by Nancy Leveson
- *Real-Time Software Systems*

Software design as philosophy:

- *Beyond Programming To a New Era of Design* by Bruce Blum