cs121 - software development
logical design

alexandre r.j. françois

visiting associate professor of computer science

outline

software design

object-oriented design

design heuristics

uml class and interaction diagrams

design practice

desirable design properties

sai: from conceptual to logical specification

design

what is design?

verb: planning

noun: blueprint

“Plans are worthless. Planning is priceless.”
- Eisenhower

software design

logical design addresses architectural issues

- what are the system's “components”
- what are their external (logical) interfaces
- how do they interact with each other

- architectural style:
  - components + connectors + constraints

physical design addresses organizational issues

- *components* (smallest units of physical design)
- internal structure (data, methods, etc.)

object-oriented design

everything is an object

an *object* is any specific entity that exists in the program at run time

- an object is a dynamic entity with specific values and attributes

a *class* is a static description of a category of objects

design heuristics

find real world objects

- identify the objects and their attributes
- determine what can be done to each object
- determine what each object is allowed to do to other objects
- determine the parts of each object that will be visible to other objects
- define each object's interface

design heuristics

form consistent abstractions

- abstraction: ignore some details
- aggregates (e.g. “game state”)

encapsulate implementation details

- prevent access to details

inherit – when inheritance simplifies the design

hide secrets

- hide complexity
- hide details that are subject to change

design heuristics

hide secrets

- hide complexity
- hide details that are subject to change

identify and isolate areas likely to change

- hardware dependencies
- inputs and outputs
- etc.

keep coupling between components loose

design heuristics

keep coupling between components loose

look for common design patterns

- reduce complexity by providing ready-made abstractions
- reduce errors by institutionalizing details of common solutions
- provide heuristic value by suggesting design alternatives
- streamline communications by moving the design dialog to a higher level

additional design heuristics

formalize class contracts

keep your design modular

draw a diagram

consider using brute force

uml class diagrams

static modeling

- class name, attributes (state), operations (behavior)
- class relationships: generalization, association, etc.

generalization (inheritance): **is-a**

association (composition, aggregation): **has-a**

an association has a direction and a role

- e.g. attribute

uml interaction diagram

dynamic: how objects interact in a running system

- instances (name + class/type)
- relationships (messages)

message types:

- simple, synchronous, asynchronous, return

design practice

design is about balance

- top-down vs. bottom up
- for now vs. for later

design decision captured in the

- **software design description document**
- discussions and decisions in the Wiki
- create diagrams
- document design decisions in the code (doxygen)

desirable design properties

minimize complexity

maintainable

loose coupling

extensibility

reusability

high fan in

low to medium fan out

portability

leanness

stratification

standard techniques

sai: from conceptual to logical specification

sai aims to facilitate system design

- architectural style: components-connectors-constraints
- architectural patterns

conceptual level description:

- cells, repositories, data (vague terms)

logical level description:

- data: types and instance names
- cells: input and output

see: mfsm.sourceforge.net/UserGuide.html

sai data model

hierarchy of “nodes”

logical characterization of a node instance:

- a type (type ID string)
- an instance name (string)
- a list of subnodes (optional)

a node's type and instance name form its *signature*

sai processing model

a cell's input is logically expressed in its active and passive filters

a cell's output is logically expressed as a hierarchy of signatures

a filter is a hierarchy of signatures

- names can be pattern strings (with wildcard characters)