6.005 elements of software construction

dependences

Daniel Jackson

November 19, 2007
topics for today

dependences and decoupling
  ◆ sad stories of coupling
  ◆ where dependences come from

dependence diagrams
  ◆ notation and examples

photo organizer study
  ◆ analysis of existing dependences and their liabilities

strategies for decoupling
  ◆ factoring out; interposing spec; callbacks
background
The Navy's Smart Ship technology may not be as smart as the service contends. Although PCs have reduced workloads for sailors aboard the Aegis missile cruiser USS Yorktown, software glitches resulted in system failures and crippled ship operations, according to Navy officials. Navy brass have called the Yorktown Smart Ship pilot a success in reducing manpower, maintenance and costs...

But the Navy last fall learned a difficult lesson about automation: The very information technology on which the ships depend also makes them vulnerable. The Yorktown last September suffered a systems failure when bad data was fed into its computers during maneuvers off the coast of Cape Charles, Va. The ship had to be towed into the Naval base at Norfolk, Va., because a database overflow caused its propulsion system to fail, according to Anthony DiGiorgio, a civilian engineer with the Atlantic Fleet Technical Support Center in Norfolk. “We are putting equipment in the engine room that we cannot maintain and, when it fails, results in a critical failure,” DiGiorgio said. It took two days of pierside maintenance to fix the problem. The Yorktown has been towed into port after other systems failures, he said.

Atlantic Fleet officials acknowledged that the Yorktown last September experienced what they termed “an engineering local area network casualty,” but denied that the ship’s systems failure lasted as long as DiGiorgio said. The Yorktown was dead in the water for about two hours and 45 minutes, fleet officials said, and did not have to be towed in. ... The Yorktown lost control of its propulsion system because its computers were unable to divide by the number zero, the memo said. The Yorktown’s Standard Monitoring Control System administrator entered zero into the data field for the Remote Data Base Manager program. That caused the database to overflow and crash all LAN consoles and remote terminal units, the memo said... Despite the USS Yorktown’s setbacks, the Navy plans to use Smart Ship technology on other classes of ships...

... [A]ccording to DiGiorgio, who in an interview said he has serviced automated control systems on Navy ships for the past 26 years, “There is very little segregation of error when software shares bad data,” DiGiorgio said. “Instead of one computer knocking off on the Yorktown, they all did, one after the other. What if this happened in actual combat?”
two modules are “coupled” when you can’t
  ‣ understand one without understanding the other
  ‣ modify one without modifying the other

decoupling: the essence of software design
  ‣ how to have the sharing and communication needed
  ‣ with minimal coupling

as with most ideas in software engineering
  ‣ you can’t address it until you can describe it
  ‣ so need a simple, lightweight notation
dependence sources

original concept

"I have identified some simple concepts that can help programmers design software so that subsets and extensions are more easily obtained. These concepts are simple if you think about software in the way suggested by this paper. Programmers do not commonly do so."

Available at: http://portal.acm.org/citation.cfm?id=800099.803218

the notation we use


emergency stop study using dependences

what causes dependences?

in unsafe languages, eg. C

• out-of-bound access to array writes somewhere else
• where depends on compiler, generally unpredictable
• so one module can clobber another arbitrarily!

in safe languages, eg. Java, Python

• array accesses checked at runtime; no pointer arithmetic; etc
• modules can only communicate explicitly
• all dependences can be seen in naming
  if A doesn’t name B, or something that B names, they can’t interfere

but determining dependences still tricky

• some objects have many names
• communication through external resources (eg, files)
dependence diagrams
elements

**squared box denotes**

- class, abstract class or interface
- double bar at top for class
  suggests code with spec

**rounded box**

- for grouping classes or interfaces
- show grosser dependences
relationships

A uses B: A relies on B to meet its spec

A implements I: result is that a dependence on I leads to an indirect dependence on A

A extends B: result is that a dependence on B leads to an indirect dependence on A, and vice versa
how dependences arise

A uses B when

- of static method or object method calls from A to B
- code in A reads or writes field in B
- A reads a file or pipe that B writes

**a common tricky case**

- A puts B objects in container C
  - typically A uses B and C, C uses Object, B extends Object

**inheritance**

- when A extends B, methods in A can call methods in B and vice versa
- so A uses B and B uses A!
- this is why inheritance is often messy: A and B really form a single module
example: quoter

from lecture 5

notes

• ignored dependences on `java.util`
• grouped all dependences on `java.net`
• design idea #1: separate network code from formatting code
• design idea #2: make `QuoteDisplayer` independent of formatting language (i.e., `HTMLGenerator` and `RTFGenerator`) by introducing interface `Generator`
• `QuoteApp` depends on `HTMLGenerator` and `RTFGenerator` because it needs to instantiate them
example: address book

from lecture 19

notes
- for intra-**swing** dependences, only some samples shown
- key design idea of MVC: dependences of model on view are weakened by interposing **Listener** interfaces
- dependence on classes such as **ListDataListener**: model still coupled to decision to use **swing**, though not on details
questions

- why Main -> java.io?
- why Photo -> metadata?
- why Catalog -> metadata?
- why Main -> metadata?
- why Catalog -> AndFilter?
- why Catalog -> java.io?
making changes

how would you make these changes?

- switch to a different metadata extractor API
- add a metadata tag for geo-location and a corresponding filter
- only import files that match certain criteria
- allow filtering on negations (tag value is not x)
- instead of HTML output, show thumbnails in a GUI

how do the dependences make things harder?
decoupling strategies
strategy #1: factoring

idea

- to eliminate dependence of A on B
- break out Ab, part of A that encapsulates use of B
- result: Ab depends on B, but A does not

simplest strategy

- used in all programming idioms

example

- QuoteDisplayer/Quoter
strategy #2: interposing spec

idea

• to eliminate dependences of A on B1, B2, etc
• introduce spec B that generalizes over B1, B2, etc
• have A use just B

example

• QuoteDisplayer/Generator
strategy #3: reverse call

idea
• to weaken dependence of A on B
• have B call A instead
• usually A notifies B through interposed spec

called “Observer” pattern

diagram

example
• AddressBook/ListDataListener
exercise

for EXIF example

• list undesirable dependences
• for each one, find appropriate strategy to eliminate or weaken
• show resulting (fragment of) dependence diagram
aren’t the diagrams the same?

- no! different purposes
- dependence diagram: shows coupling
- object model: shows state invariants

differences

- not all dependences are relations
  - dependence due to method call doesn’t show in object model
- not all relations are dependences
  - object may contain reference to another but never use it
    - Liskov calls this “weak dependence”
- object model doesn’t aggregate links to a package
summary

decoupling
  · essence of software design

dependence diagram
  · lightweight and easy to use
  · but sheds much light on design
  · best way to expose design flaws

but note
  · not all dependences are equal!
  · eg, compare on dependence on metadata package in EXIF program to dependence on swing package in AddressBook program