modularity & dependences

Daniel Jackson
what makes a system “modular”? 

in traditional engineering
› components can be built, tested & replaced independently

in software engineering
› components can also be reasoned about independently
› changes to components are “localized” or “contained”

containing failures?
› may not follow from modularity
› maybe the opposite (since modularity encourages sharing)

“The current configuration of electronics on the Dreamliner puts passenger electronic entertainment on the same computer network as the flight control system.”

when does modularity fail?

client-service binding
› when service changes, client must too
› eg: old apps fail on new release of OS

abstraction violation
› service doesn’t change, but client must anyway
› eg: representation of datatype is leaked

module-problem binding
› one piece of the problem in two modules
› eg: document is paragraph-structured, in Word
Parnas’s uses relation

Client

uses

Service
example: a browser
minimal subsets

a common problem
› suppose you want module M
› what other modules do you need?

solution
› you need all the modules M uses
› and the ones they use...

examples
› minimal subset for Renderer? for Parser?
other uses of uses

you change module M
  › which modules might break?

you want to test M
  › which modules must be complete?

you want to reason about M
  › which module’s specs do you need?
software subtlety

X may use Y without knowing about it
  › eg, because Y is configured dynamically
  › X only knows interface of Y

example: observer pattern
  › interface I interposed between subject S and observer O
  › now S depends on I, but not on O
layering: a common pattern

Diagram of Android's multi-layered operating system architecture (in "The Embedded Beat") removed due to copyright restrictions.

Android architecture from https://community.freescale.com/community/the-embedded-beat/blog/2010/05/24/android-makes-the-move-to-power-architecture-technology
back edges
Matrix of classes of Spring framework (in "Dependency Structure Matrix") removed due to copyright restrictions.
highlighting back edges

from: http://www.lattix.com/products/modules/java
how to avoid modularity failures

client-service binding
› control dependences, especially back edges

abstraction violation
› make sure clients only rely on specs
› use language abstraction constructs

module-problem binding
› encapsulate design decisions
› this is “information hiding”
a rule of thumb
› “Don’t Repeat Yourself”

can you explain this rule?
› how does it relate to *uses*? information hiding?
› what are its limitations?