Today

- One brief project #1 description
- Finish software design introduction
  - Open implementation
  - Layering/uses relation
- Some consequences of reality in design

Reality: some consequences

- One commonly stated objective of good design is the ability to reason about the software system
  - It is not always clear if this means reasoning about the structure, or reasoning about the behavior, or (most likely) both
- Top-down design, ADT-based design, information hiding, layering all — at least in principle — help to some degree with reasoning
- One reason is that there is, or there can be, a clear specification of what the system is intended to do

Claim

- I claim that the basis for reasoning is in large part based on the fact that in these approaches the names relation and the invokes relation are closely related
- That is, to invoke a part of a program a second part of the program must know the first part's name
- With a specification (formal or otherwise) of the second part's interface, the first part can invoke it with confidence
- This has much in common with the strong relationship between static structure and dynamic behavior that Dijkstra advocated
A look at event-based programming

- One approach that is widely used and difficult to reason about is event-based programming
  - Roughly equivalent to interrupts at the architectural and operating systems levels
  - The key: names and invokes are decoupled (to varying degrees)

The broadcast analogy

- Has a flaw: people listen to the radio or watch the TV but (for now, at least) don’t fundamentally change anything going on at the source of the broadcast
- But when a programming event is raised, the computation that is invoked may well change the behavior of the component that invoked the event
- But that component doesn’t know what components are invoked, or what they do

A whiteboard example

- A set of vertices and a set of edges
  - A desired constraint between vertices and edges – together they form a graph
    - That is, no edge is included the edge set that does not have the corresponding vertices in the vertex set
    - Lots of policies to achieve this constraint
    - Direct access to the vertex and edge sets complicate maintenance of the constraint
- Possible extensions include
  - A lazy bit that allows the constraint to be violated
  - A count of the number of vertices
Trade-off between flexibility and reasoning

- At least it seems to be, not only for event-based programming, but also for exceptions, etc.
- We'll look at a broader approach – with some similar tradeoffs – next time when we talk about aspect-oriented programming