Non-object-oriented design methods

Software Engineering Design
Lecture 15

(reminder) Software Design is
- a creative process
  - no cook book solutions
- goal driven
  - we create a design for solving some problem
- constraint driven
  - by the function to be served and the constructions which are possible
- good designs can be recognised
  - simple, coherent, adequately meets requirements, adaptable

Some Other Design Methods
- So far we have focussed on object-oriented design
- However OO design is NOT suitable for all types of systems
- All design methods involve a hierarchy of decompositions which partition the design into subsystems or components
- This lecture outlines some non-OO methods for doing this decomposition

Data-Oriented Design (DOD)
- A school of thought: “identification of the inherent data structure can be used to derive the structure of a program”
- Logical construction of programs (LCP)
  - Developed by Warner (’74)
  - Draw upon relationship between data structure and procedure structure
- DOD may be successfully applied in applications that have well-defined, hierarchical structure of information

Event-Oriented Design
- Each subsystem consists of components that handle similar type of events
- Examples: editors; rule-based systems in AI; most real-time systems are event driven
- Disadvantage:
  - Subsystems don’t know if or when events will be handled

In summary...
- Data-oriented design
  - start with external data structures and continue by adding more detailed data structures
- Event-oriented design
  - what events are possible for this system?
  - what response is required for each event?
  - how does each event change the system state?
"Structured" design

- Try to decompose each subsystem into modules
- Two main strategies:
  1. **Object-oriented decomposition** – subsystem decomposed into a set of communicating objects
  2. **Function-oriented pipelining** – subsystem decomposed into functional modules

In summary...

Structured design

- modular / functional design
  - each subsystem captures one of the functions of the system
- outside-in / top-down design
  - start with black boxes and their inputs and outputs
  - then divide each box into internal input - output boxes

Outside-In Design Example

**Brick Sorter Design**

- Input via the light sensor, sends message to the controller
- Output via the kick arm, receives eject warning from the sensor process and then kicks when the controller is ready
- Controller manages the timely co-ordination of the inputs and outputs
- Example: presented using the Uppaal system
- see www.uppaal.com for more information

Verification

- Eventually, the controller is ready to eject a red brick
  - E<> Controller.c3
- Eventually bricks reach the end of the conveyer
  - E<> Box.k5
- The observer never sees a red brick (because they have all been correctly removed) and so he is never sad
  - A[]not Observer.sad
Formal Methods and Refinement

- A final (extreme) alternative to the design process is the use of formal methods.
- Software specifications are meticulously transformed into mathematical statements.
- Then a process of refinement is used to derive provably working code from the mathematical specification.

Formal Methods cont.

- Formal methods are used for safety critical applications.
- Zed is a specification language that can map specifications in first-order logic into executable pseudo-code.
- The requirements stage of development is very expensive, but the design, implementation and testing can be a lot cheaper than other methodologies.

Test Driven Design

- Test driven design is an Agile method
  - Individuals and interactions over processes and tools
  - Working software over comprehensive documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan
  - TDD maps the requirements directly into testing code. The source code is then written specifically to pass these tests (and only to pass these tests)
  - This process is repeated incrementally until the product passes all the tests, and thus meets the requirements.

Test Driven Design

- "TDD completely turns traditional development around. When you first go to implement a new feature, the first question that you ask is whether the existing design is the best design possible that enables you to implement that functionality. If so, you proceed via a TDD approach. If not, you refactor it locally to change the portion of the design affected by the new feature, enabling you to add that feature as easily as possible."
  
  Scott Ambler

SED Road Map

- This is the final lecture in the series on software design
- Next week we will study methods for testing software