CS 2340 Objects and Design

Software Design

Christopher Simpkins
chris.simpkins@gatech.edu

Design

Design (noun)

> *A plan or protocol for carrying out or accomplishing something. – Webster's Dictionary*

Engineering design (verb):

> *A systematic, intelligent process in which designers generate, evaluate and specify designs for devices, systems, or processes whose form(s) and function(s) acheive clients' objectives and users' needs while satisfying a specified set of constraints. – Dym and Little, quoted in Carlos Otero, Software Engineering Design*

Fundamental Design Principles – Otero

- Modularization
- Abstraction
- Encapsulation
- Coupling and Cohesion
- Separation of interface and implementation
- Suficiency and completeness

Object Design – Wirfs-Brock

- Design driven by roles and responsibilities
- Write story about system, identify themes and abstractions, identify candidate roles/classes that support themes
- When candidates identified, model reponsibilities and collaborations
- Exploratory design with CRC cards (candidates, responsibilities, collaborations)
- Object roles often fit into these steriotypes:
  - Informatoin holder – knows and provides information
  - Structurer – maintains relationships between objects and information about those relationships
  - Service provider – performs work and, in general, offers computing services
  - Coordinator – reacts to events by delegating tasks to others
  - Controller – makes decisions and closely directs others' actions
  - Interfacer – transofrms information and requests between idstinct parts of the system

Software Development is not Engineering – Reeves

Software development is not engineering (at least not in the traditional sense). In traditional engineering:

- Engineers (design team) create detailed designs
- Designs are given to manufacturing teams to build
- If design is complete, no further input from design team is necessary
- Building can be very expensive
- Bugs in hardware design even more expensive: e.g., if design error in a car is not caught early, thousands of cars can be sold with defects resulting in deaths and recalls, bridges can collapse, etc.

Feedback cycle very long and expensive, leading to traditional engineering's fetish with detailed documentation

The Code is the Design – Reeves

Fundamental principle of software design:

> *The code is the design. – Jack Reeves, 1992*

In software development:

- Source code is given to a compiler and linker, which builds the runnable software very quickly and inexpensively
- Tight feedback loop
- Easy to generate complex designs
- Testing and debugging is part of the design process

Software design is about managing complexity.

Agile Design – Martin

Fundamental principle of agile design:

> *The code is the design. – Jack Reeves, 1992*

Design Smells

- Rigidity – system is too hard to change becuase change in one place forces changes in many other places
- Fragility – changes break things that are conceptually unrelated
- Immobility – too hard to resuse components in other systems
- Viscosity – hard to do it right, easy to do it wrong
- Needless Complexity – infrastructure with no direct benefit
- Needless Repetition – repeated structures that could be unified under a single abstraction
- Opacity – hard to read and understand

Design smells avoided or fixed by applying design principles like SRP, OCP ...

Levels of Design

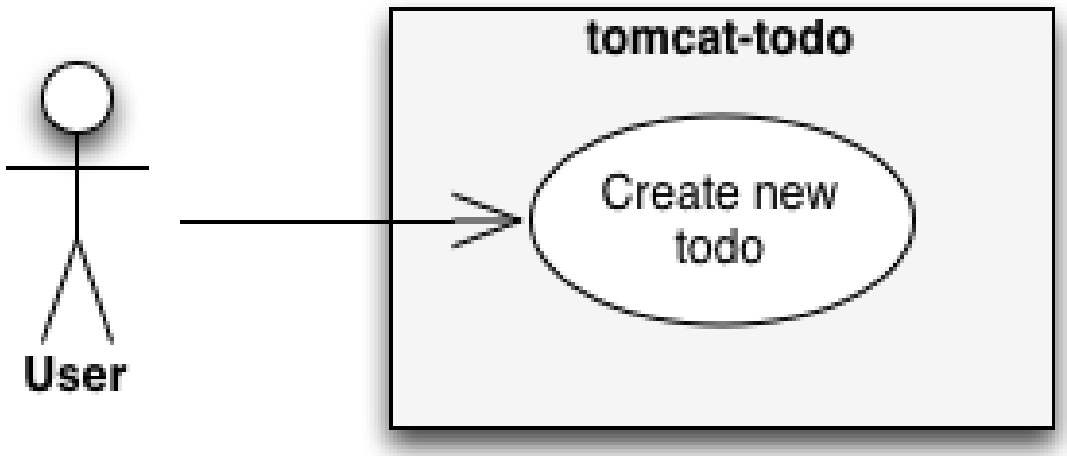
- High-level: system architecture
  - Stand-alone
  - Client-server
  - N-tier
  - Service-oriented architecture
- Detailed Design
  - Classes and methods
  - Data interchange formats (XML schema, JSON)
  - Entity-relationship models, database schema

Unified Modeling Langauge (UML)

A standardized diagrammatic language for communicating OO designs in a language-independent way. Very rich, but for now focus on:

- use cases,
- domain model (classes and associations),
- packages, and
- sequence digrams.

Use Cases



A use case describes some user's interaction with the system. Most use cases contain:

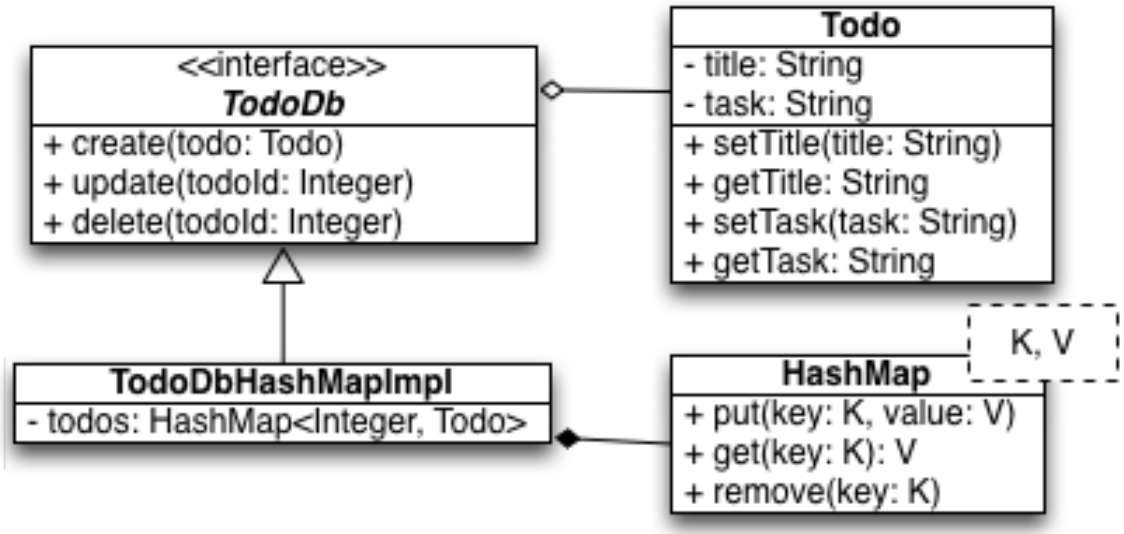
- an *actor*, here simply "User,"
- a *use case*, here "Create new todo," and
- (optionally) a *system boundary*, here "tomcat-todo."

Class Diagrams

Class diagrams contain

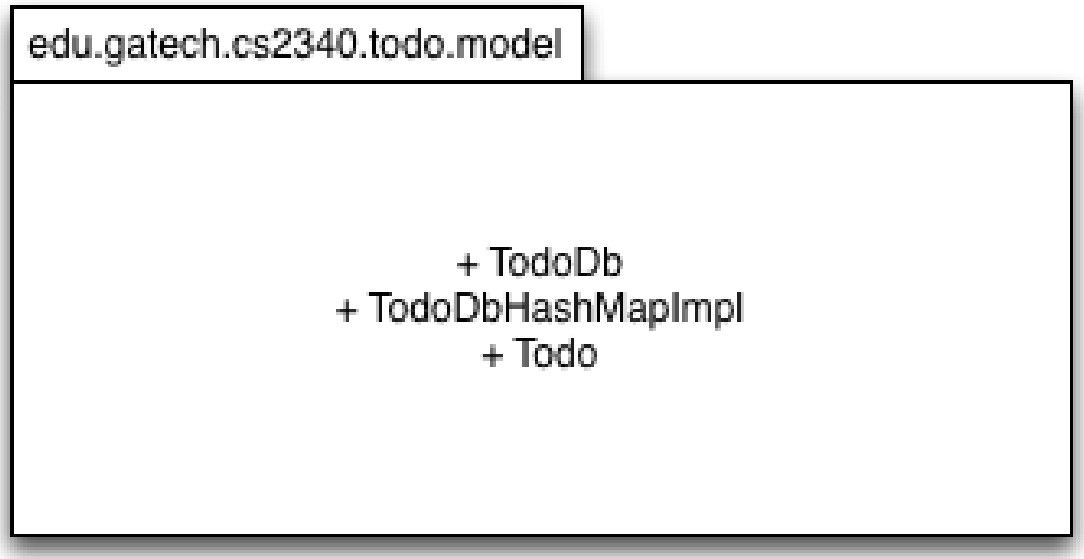
- a *class name*, here “Todo,”
- *instance variables*, here “title” and “task”. Note that types are given after names, as in “: String”. The “-” means private.
- *methods*. The “+” means public. (“#” means protected, but we have no protected members in this example.)

Associations



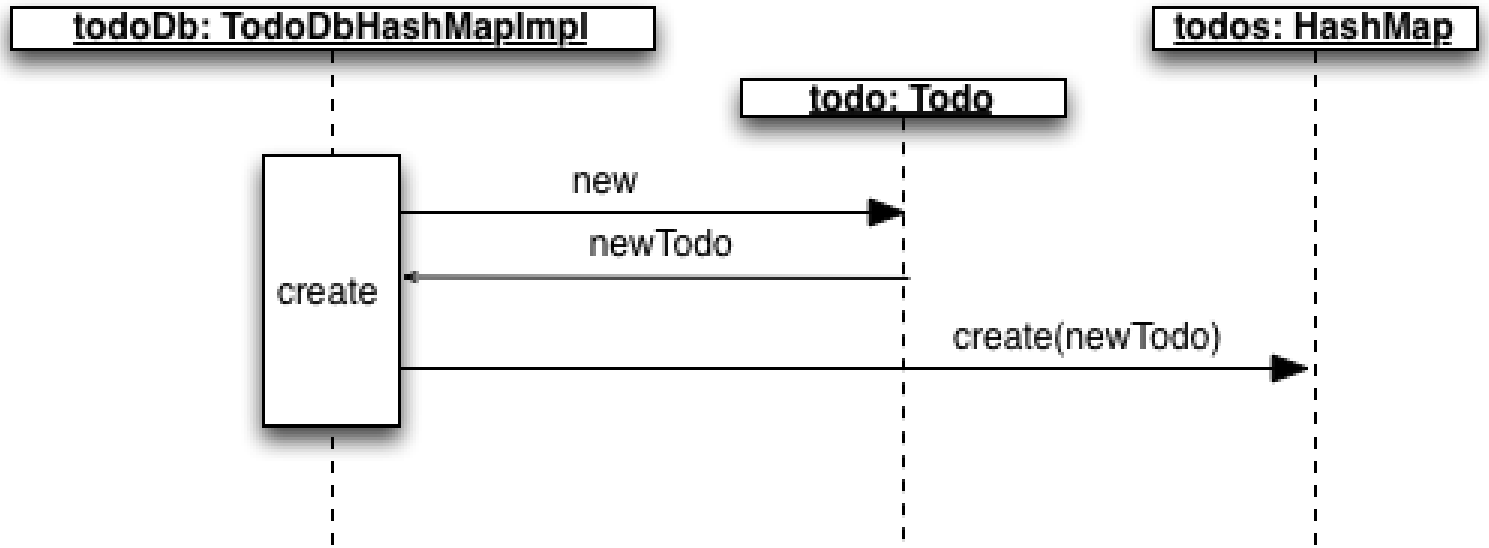
- *TodoDb* is italicized, meaning it is abstract. The "«interface»" further means that it is an interface.
- `TodoDbHashMapImpl` is a subtype of `TodoDb`.
- `TodoDbHashMapImpl` is *composed* of a `HashMap<K,V>`.
- `HashMap` is a paramterized type with type parameters K and V.
- `TodoDb` aggregates `Todo` objects.

Packages



- The tab shows the package name.
- The main box lists classes and interfaces using the same +, -, and # visibility modifiers used for members in class diagrams.
- An alternative form is to simply put the package name in the main box and not list the members of the package.

Sequence Diagrams



- The top rectangles represent objects (instances of types/classes).
- Time progresses vertically downward.
- The dashed lines represent object lifetimes.
- The narrow vertical boxes represent operations of the objects (here, only create on the todoDb object).
- Arrows are “messages” or method calls and returns.

Closing Thoughts

- Design is an art born of empirical science and intuitive experience
- Practical experience and formal studies of software systems have produced design principles and guidelines
- Design involves tradeoffs – balancing constraints means prioritizing

And a final word about design documentation: if Jack Reeves is right (and I think he is), then

- The code is the design, produced by the design team,
- The compiler and linker are the manufacturing team, and
- Compilers and linkers don't use design documents.
- Design documents are for other "designers" to help them understand the code

Punch line: design documents are like comments – they make up for lack of expressivity in the code. They're a necessary evil at best, not the "point" of design.