Software Development Life Cycles

台北科技大學
資訊工程系
鄭有進 副教授
http://www.ntut.edu.tw/~yccheng
Software development is hard
Software development process

- To build a software application, you need a **software development process** that addresses requirements, expectations and realities simultaneously.

- There are well-documented processes in the SE area.

- In this lecture, a process based on object orientation is highlighted.
Steps in a process

1. Define the problem
2. Understand the problem
3. Formulate a software solution
4. Construct the solution in a particular language
5. Spec-test the construction
6. Integrate modules
7. Integrate the modules

- Requirement Analysis
- Design
- Implementation
- Testing
- Integration
Modeling tools in a process

- Requirement
- Analysis
- Design
- Implementation
- Testing
- Integration

Integration station
- Functional testing, black-box testing
- Programming language, styles, idioms, unit testing
- Architectural patterns, design patterns

Domain models, analysis patterns
- Use cases, scenarios
Artifacts created in a process

- Requirement
- Analysis
- Design
- Implementation
- Testing
- Integration

- Domain models (static model, interaction model, and state model)
- Functional and nonfunctional requirements
- Program code and unit tests
- Architecture diagrams, class diagrams, interaction diagrams
- Build reports
- Spec conformance report, bug
Procedure-orientation

- We view work to be done mainly as a procedure
  - Yet another evidence that this is workable model: web services
- Implicit in the process are entities upon which the procedure is carries out.
  \[ \text{Data structure} + \text{algorithm} = \text{program} \]
- But procedure does not capture the entities well
Object-orientation

- We still view the work to be done as a procedure.
- But participants (entities) of the process are made explicit.
- The participants interact to get the procedure done.
  - Now the *de facto* programming paradigm for intra-service modeling
Object, component, web services, which one?

- Objects if problem is solved within one process-space
- Component if problem is solved in the same site but span several processes
- Web service if problem is solved span several sites
Form and Function (I)

Software has two aspects: form and function

```
UseCase 1..* 1 BaselineArchitecture
function form 1 1 SupportArchitecture
```

```
Architecture
```

```
Form and Function (II)

- In mainstream OO processes (e.g., RUP and XP), function and form grow together.
  - Architecture baseline takes shape early as primary use cases are built.
- However, in most cases of software development, function gets most of the attention.
  - Form is usually just whatever the outcome of the aggregation of functionalities.
Process weights

Agile methods: XP Crystal JAD …

RUP OMT Fusion …

Light weight
Few artifacts Hot communication Small team

Heavy weight
Many artifacts Cold communication Large team
Proven process elements

- Use cases (requirement capture)
- Increments (staging strategy)
- Iterations (rework scheduling strategy)
- Patterns
- Unit tests
- Continuous integration
- Review
A process and its workflow

1. Write use cases
2. For each use case
   3. Find domain model
   4. Find design model
   5. Implement and do unit tests
   6. Integration
   7. Test use case
   8. Review
Increments and iterations (1)

- A use case is roughly an increment: it is not the whole thing, but it delivers functionality by coherent chunks.

- Several iterations may be executed on an increment
  - To remove bugs, add functionality, and improve quality
Increments and iterations (II)

- Aim for a working system (at least covering the present use case) at the end of an increment.
  - Early feedback to users can improve subsequent development
- Iterations should be time-boxed: e.g., 2-4 weeks on small projects.
Increments and iterations (III)

- Incremental strategy ensures that transition from domain model to design model is feasible
  - Increment delivers systems in coherent chunks
  - First increment should shape the baseline architecture
  - Later increments built on top on early increments
Iteration vs. waterfall (I)
Iteration vs. waterfall (II)
Process by phases

Effort/result

inception

elaboration

construction

transition
1. Use case modeling

- Use case: a sequence of steps describing an user-system interaction in which a well-defined service is tendered.

- Scenario: a set of use cases tied together in order to complete a user’s task in the application domain.
Use case functions (1)

- Use case captures *functional requirement* for an actor
- Use cases are high-level, domain-oriented descriptions written in a way understood by project stakeholders
  - Client, end-users, project manager, developers
Use case functions (II)

- As a starting point in system modeling.
  - All the other artifacts are derived from the use cases.

- As a guide to testing the system built.
  - The system built must reflect the use case.
  - Useful in function/acceptance tests
Use case tips

- Write just enough detail to keep the use case interesting
  - Too much detail reverts the development to a structured process
- You don’t need to have all use cases in place to start building system.
  - However, the first use case should not be a trivial use case
2. Domain Modeling

- Extract the static conceptual domain structure.
- Use whatever you feel most comfortable (CRC, semantic nets, conceptual diagram, header files, etc.)
  1. concentrate on the concepts
  2. find out how concepts are related
- The concepts should have a consistent medium-to-coarse granularity level
  - Fine grained concepts bug you down (too much detail)
3. Design modeling (1)

- Models the domain as a set of software classes.
  - Therefore, a design model is also a valid domain model, but it may be not easy to obtain off-hand.
- Coarse-grained concepts are usually realized be a set of cooperating classes
  - Each coarse-grained concept becomes a micro-controller delegating work to its internal classes
  - Knowledge of design patterns and architecture styles are extremely helpful
Design modeling (II)

- Created by finding what a class must do to realize the function being required on the conceptual entities.
  - Responsibilities that are implied in the conceptual model must be made explicit in the design model
  - Discover the responsibilities in the order of their occurrence
4. Implementation and unit tests

- Can happen after the creation of design model
  - design models are drawn and recorded
- Can happen in parallel with design model creation
- Write tests for every member function of a class (more detail on unit tests later)
- Tests should be accumulative and automatic
5/6. Continual integration and functional tests

- Function tests are written for each use case.
- An increment is completed only if the functional tests pass.
- Functional tests can be written by extending unit tests.
7. Review

- User evaluation and feedback
- Process observation and improvement
- Learning “what you don’t know you don’t know”
- Artifacts accumulation and synchronization
  - Use cases, domain model, design model, key interaction, tests
Coverage by courses