Developing Software

A Disciplined Approach

Use-Case Diagrams

Example: Use-Case Diagrams

Relationship Lines in Use Case Diagrams

An ER Diagram
The Student Class

<table>
<thead>
<tr>
<th>student</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
</tr>
<tr>
<td>sname</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>register()</td>
</tr>
</tbody>
</table>

Generalization

<table>
<thead>
<tr>
<th>student</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>register()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ta</th>
</tr>
</thead>
<tbody>
<tr>
<td>salary</td>
</tr>
<tr>
<td>time()</td>
</tr>
</tbody>
</table>

Associations

<table>
<thead>
<tr>
<th>student</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>course_off</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>0..* one 0..*</td>
</tr>
</tbody>
</table>

Aggregations

<table>
<thead>
<tr>
<th>course_off</th>
</tr>
</thead>
<tbody>
<tr>
<td>instructor</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>faculty</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Object-Oriented Analysis

- Interview users, collecting requirements.
- Record objects described by users; collect like ones into classes.
- Discover class attributes and record in UML diagram (part of Requirements Document).
- Identify relationships among classes and cardinalities of relationships.

Object-Oriented Analysis, cont’d

- Discover class services and record in UML diagram.
- Specify use and behavior of services using in Requirements Document. Use pseudocode, preconditions/postconditions, English, etc.
Identify Classes

- course
- student
- faculty
- course_off
- err
- ta

Identify Attributes

- course
- course.description
- student
- student.soc
- student.grade
- faculty
- faculty.name
- ta
- salary

Identify Relationships

- Student
- Course
- Faculty
- TA

Identify Services

- Enroll
- Withdraw
- GetGrade
- GetMajor

Refine

Object-Oriented Design

- Analyze the relationships. How will they be implemented?
- Refine the services into methods.
- Analyze the methods. What new classes, attributes, methods, and relationships are needed to support the implementation of the methods?
Design Patterns

- Idea: to capture common features of architectural designs
- Example: Composite object (hierarchies of components and subcomponents)
- Example: Chain of Responsibility for handling events
- Attempt to capture wisdom of expert designers/developers: reuse the design and knowledge about its mapping to code

Makefiles: a GCC Example

```plaintext
myprog : myprog.o support.o
  gcc -o myprog myprog.o support.o

myprog.o : myprog.c defs.h
  gcc -c myprog.c

support.o : support.c defs.h
  gcc -c support.o

clean :
  rm myprog myprog.o \ support.o
```

Windows Software Dependencies

```
CPP
CPP
CPP

CPP
CPP
CPP

CPP
CPP
CPP
```

Windows Makefile, Example 1

```
# Makefile to compile three source files a.cpp, b.cpp, and c.cpp
a.exe : a.obj b.obj c.obj
  link.exe a.obj b.obj c.obj # Microsoft linker

a.obj : a.cpp
  cl.exe /c a.cpp # MS Visual C++ compiler

b.obj : b.cpp
  cl.exe /c b.cpp

C.obj : c.cpp
  cl.exe /c c.cpp
```
Windows Makefile, Example 2

CPP = cl.exe
# MS Visual C++ compiler
CPP_FLAGS = /c
# command-line flags for compiler
LINK = link.exe
# MS Linker
OBJECTS = a.obj b.obj c.obj

a.exe : $(OBJECTS)
	$(LINK) $(OBJECTS)

a.obj : a.cpp
	$(CPP) $(CPP_FLAGS) a.cpp

b.obj : b.cpp
	$(CPP) $(CPP_FLAGS) b.cpp

c.obj : c.cpp
	$(CPP) $(CPP_FLAGS) c.cpp

clean :
	del $(OBJECTS)

Versioning

Version Management Software

- SCCS, RCS, CVS, SourceSafe are examples.
- Support check-in, check-out (with or without locks), and merge operators.
- Often paired with configuration management software.
- Typical implementation: backwards deltas.

CVS (Concurrent Versions System)

- Maintains multiple versions of projects organized into directories
- Supports merging of updates made concurrently by multiple developers
- Versions stored in central repository in RCS format (backward deltas)
- Free download available at www.cvs.home.org

Checking In and Out

- To check project files out of repository:
  cvs checkout bigproject
- To check project files into repository:
  cvs commit
- To update local copy (merge repository copy into local copy):
  cvs update

Testing

- Coverage: Assuring that all lines of code or paths or interfaces are tested.
- Use a tool such as GNATS or Bugzilla to maintain sharable bug database.
- Only tester is allowed to designate bugs as “removed”.
Assertions

```c
#include <cassert>
... void CQueue::add(CNode *node) {
    assert(node != NULL);
    ...
    assert(rear == node);
}
```

Assertions, cont.

- Assert is conditionally compiled; does not get compiled if symbol NDEBUG is defined.
- Use assert only to detect errors in code, not to handle exceptions.
- Asserts can verify preconditions and postconditions: assuring that code lives up to its contract.

Software Development is Engineering.

- The developer is most productive when he/she can combine existing components. (Software reuse has eliminated the Software Crisis of the 1980s.)
- Organizing existing software modules for effective retrieval is still an open problem. (Do XML and XQuery provide a solution?)

Software Development is Science/Mathematics.

- Every new Requirements Document, every new Design Document, every new software module is a new model of (part of) the world. The business of the computer scientist is building models of reality.
- The computer scientist also is an experimentalist: now and then he/she “runs” programs.

Conclusions

- Disciplined use of tools such as CVS, make, and Bugzilla has been a major win.
- OO has been a major win in the coding phase; it has encouraged massive software reuse, especially when supported by application frameworks.
- OOA and OOD have supported marginal productivity improvements over older methodologies.