Goals of the Lecture

- Discuss a variety of implementation issues
  - Iterative Implementation Techniques
  - Translating Associations
  - Scalability Considerations
  - Managing Rapid Prototypes and Beta software

Goal: Begin Implementation

- Situation
  - Constructed
    - Use cases, class diagrams, activity diagrams, state diagrams, etc.
    - Baseline architecture
    - Implementation Plans

- Problem
  - Where to start?

From UML Distilled

- Plan Iterations around Use Cases
  - Each iteration is a mini-project
    - Perform analysis, design, coding, testing, and integration
    - End with user demo and system testing
  - Repeat

- Each iteration is incremental, building on previously constructed functionality
Why more analysis and design?

- Why would A&D occur in each iteration?
  - Moving down levels of abstraction
    - The design should be complete at a logical level, what’s left are more practical issues
    - For instance, the details of a user-interface class will eventually need to be specified
      - screen layouts, interaction paradigms, etc.
  - Interactions between use cases may reveal “holes” in the logical design

First Steps

- Identify the collaborations that realize your system’s use cases
  - Specify the structural and behavioral aspects of each collaboration
- Create stubs for each class identified in a collaboration
  - Each class is defined in full but with empty bodies for operations

Example

```java
package chimera.concepts;
import java.util.*;
public class link extends concept {
  public link() {}  
  public Enumeration getAnchors() { return null;  }
  public void addAnchor(anchor a) {}  
  public void removeAnchor(anchor a) {}  
  public int numberOfAnchors() { return 0;  }
  private Vector anchors;
}
```

Next

- Construct a “start system” class
  - All it does is create the objects needed for your system, start up any required threads, and place the system in its start state
- Compile, link, and execute
  - Nothing should happen, but you’ll be surprised at how many problems crop up the first time you do this!
Finally...

- Pick an operation in one of the classes
  - Implement it
  - Compile, link, execute...
- Repeat until Use Case is done
- Repeat until all Use Cases are done
- Pause for redesign and risk management as needed

Don’t forget

- Comments
  - take advantage of tools like JavaDoc
- Testing
  - Kent Beck’s rule of thumb
    • A developer should write at least as much test code as production code
  - Archive the tests and use them for regression testing

Don’t forget, continued

- Metrics and Planning
  - Record how long it took you to do a task
  - Update the master plan with the team’s accomplishments
  - See <http://psp.colorado.edu/Visitors.html> for links to information on the importance of metrics at an individual level

Mapping Class Diagrams and Associations

- An association implies something about the interface of its classes when implemented
  - In particular, an interface is required that allows the association to be navigated
  - A designer can indicate the navigation responsibilities of a class with arrowheads on associations in UML diagrams
Example

Object

navigability

View

 Viewer

Link

Anchor

public class view extends concept {
    public view() {}
    public viewer getViewer() {
        return myViewer;
    }
    ...
    private viewer myViewer;
}

Discussion

- A multiplicity of one
  - private attribute
- A multiplicity of more than one
  - vector, array, binary tree, etc.
  - As dictated by the system’s functional reqs.
- Association Classes point to their members; not the other way around
  - This implies the need of a collection class that points to instances of association classes

Scalability Considerations

- A straightforward OO design process will not necessarily lead to a scalable system
- Most likely lead to primitive operations
  - Scalability requires compound operations
    - especially in the presence of client-server systems
Let me tell you a story...

- Feature request for Chimera:
  - Define an import format that allows anchors and links to be defined outside of Chimera and then imported en masse

- Implemented Feature
  - Defined an XML Document Type Definition that allowed Chimera hyperwebs to be imported and exported

Story, continued...

- Tested Feature
  - Created 100KB XML file defining a hyperweb of 1000s of anchors and links

- Released Feature
  - Primary user created six 26MB XML files defining 60,000 anchors and links (roughly half-a-million anchors and links) in total!

- Back to the drawing board!

Should have known...

- Not necessarily true
  - Relationship with user for 3+ years
  - Chimera suited hypermedia needs of their initial evaluation efforts with no problems

- Feature request was motivated by a desire to automate the creation of links for a single (!!) subsystem
  - Scalability reqs. were revealed only then!

Story continued...

- Chimera Server displays the names of all the links of a hyperweb
  - Vector linkIds = hyperweb.getLinks();
  - for i = 1 to linkIds.size() {
      - link L = linkIds.elementAt(i);
      - String name = L.getAttribute("name");
      - -- Add name to scrolling list
  }

- For 33,000 links, 1/3 complete in 8 hours!
Problem

- OO Design led to creation of
  - concept - manages attributes
  - link - manages anchors
  - hyperweb - collects links
- Natural Algorithm uses primitives
  - What’s needed however is an operation that gets all the links and names at once
  - This implies new operations and classes!

Results

- The new compound operation displayed 33,000 link names in under 5 minutes; 288 times faster!
- Implications on design
  - Identify the need for compound operations via use-cases
  - Add these operations and their associated support classes early!

Managing Prototypes

- During design, a rapid prototype might be needed
  - Purpose
    - Answer specific questions about the design
    - The fewer the better! This reduced focus allows the prototype to concentrate on one aspect only and makes it easier for you to discard it!
    - An end-user should use the prototype
      - Make sure they understand the prototype’s purpose!

Prototypes, continued

- Manager should
  - Assign at most two, preferably one, developer should construct the prototype
    - Design should continue in other areas during this phase
  - Impose a deadline on prototype construction
    - a week or two, no more!
  - Schedule the demonstration with the end-user
  - RAD Tools should be used
    - the tight deadline should encourage their use!
Prototypes, continued

- User’s feedback must be recorded
  - “Think-Aloud” while using the prototype
  - Interviews and surveys
  - For user-interfaces, draw alternatives with user still present
- Analyze data / record design decisions
  - Decide if prototype needs to be iterated
- Archive everything for later use!

Beta software, introduction

- Types of software
  - Developmental
    - Internal builds, development team only
  - Alpha
    - Internal builds, select external users allowed
  - Beta
    - External builds, large group of external users
  - Final Candidate, Golden Master, Release
- Beta ≠ Preview Releases
  - Often an excuse to ship buggy software

Purpose of Betas

- Most often all functionality is created during developmental and alpha builds of a system
- Beta software is used to test the software on the widest range of hardware/software configurations
  - functionality is often frozen
  - focus on finding and fixing bugs
    - e.g. system works flawlessly on PowerMac G3, but crashes on PowerMac 6100 with G3 upgrade card

Managing Betas

- Identify a time period for the beta
  - Can be open-ended
  - However better to set a time-limit or a number-of-open bugs threshold
    - e.g. beta is over when all category 1 and 2 bugs are fixed and there are less than 10 category 3 bugs
    - these criteria are very context dependent!
Managing Betas, continued

- Private FTP site to distribute betas to external users
- Mailing list or discussion forum for feedback
  - Development team and SQA must participate!
- Bug Database records **all** feedback
  - SQA classifies bugs, managers assign bugs based on priority, developers fix
  - Beta cycles should be short: 1-2 week granularity

Discussion

- A beta-cycle should not greatly impact design
  - Freezing functionality helps to ensure this
- Instead Focus on
  - stability, race-conditions, obscure configurations
- Record for the future
  - Unanticipated ways that users use the software
  - Requests for new functionality
- Note: Sequels are not necessarily good!
  - Word 3000, now records barometric pressure!

Fredrick Brooks, 1975

...conceptual integrity is *the* most important consideration in system design. It is better to have a system omit certain anomalous features [and] to reflect one set of design ideas, than to have one that contains many good but independent and uncoordinated ideas

Fredrick Brooks, 20 years later

A clean, elegant programming product must present... a coherent mental model... [Conceptual] integrity... is the most important factor in ease of use... **Today I am more convinced than ever**. Conceptual integrity is central to product quality.