CSS 370 (1/7/2007)

Review of Software projects

* Shrink-wrap
  + Consumer (retail)
  + Vertical market (business sectors)
* Web and Internet Services
* Custom (bespoke) software
  + IT Support
  + Internal and Contracted Development
* Open Source

Business Focus

* Enterprise Support/Resource Planning Systems (ERP)
  + Management of financials, manufacturing, inventory, distribution, and order processing.
  + Integrated around a Commercial Off-The-Shelf (COTS) RDBMS
  + Vendors: SAP, PeopleSoft, Oracle, JD Edwards
  + Usually customized to the business
* Transaction Processing System
  + Day-To-Day transactions such as billing, order entry, warrantees, purchase orders
  + Commonly specialized per-business, e.g., airline reservations or banking
  + Involves COTS or proprietary RDBMS
  + Frequently business-critical systems
* Business Support systems
  + Support business operations such as payroll, benefits, etc
  + COTS or customized
* Knowledge Management Systems
  + Cataloging, encoding, classifying, and mining corporate data such as helpdesk solutions, data warehouses, and data mining
  + Originally associated with expert systems but moving evolving toward systems with looser connections between data
* User productivity systems
  + Software to support a specific job function (Point-of-sale systems, computer-aided drafting, software IDEs, etc.)
  + Either vertical-market COTS or custom
  + The least integrated with other business systems
* Information Systems Support
  + Systems for managing IT resources including networks, hardware, and software
* Web and E-Commerce Integration
  + Integration of system siwht customers, partners, and suppliers
  + Uses internal infrastructure
* Business Solutions
  + IT infrastructure
    - OS and related software
    - RDBMS
    - Network, Internet, web services
  + ERP
  + General and Vertical Market Management and Productivity Software
  + Internal Development Applications

Aspects of good software

* Efficiency (CPU, storage, bandwidth)
* Usability (friendliness, efficiency)
* Maintainability
* Dependability
* Reliability
* Availability
* Security

SDLCs

* Waterfall
* Iterative/incremental/repetitive/cyclical
  + Spiral
* Agile
  + Short cycles
  + Evolutionary requirements
  + Automated and regression testing
  + High performance and collaborative teams
  + XP/Scrum/Crystal Methodologies

Systems analyst:

At the center of the Venn digram made up of Business, Management, and IT (developers), integrating the 3 domains.

Communication x 3

Listening, Reading, Writing, Semantics, Education, Generalization and Abstraction, Problem-solving

Design Processes

* Iterative Partitioning and Detailing of the Solution, including
  + Logical systems and subsystems
  + Database structures
  + UI
  + Infrastructure
* Technology Selections
* Design Modeling
* Review and Approval

Role of Software Architect

* Balance needs, technologies, and constraints to outline a solution
* Balance and partitioning to meet tactical objectives and strategic goals
* Logical partitioning along business and technical needs
* Understand difficulties in development, technical support, and operations
* Plan for the unplanned

Projects from Business Process Reengineering

* Often include major overhauls or integration of applications across the organization
* May be related to mergers, acquisitions, and divestures of businesses requiring merging, migration, or partition of the information technology infrastructure (including applications)
* Highly visible projects – executive team as the customer
  + Wide latitude for sweeping changes
  + Relatively little cost-benefit analysis; executive team has already bought the product

ASSIGNMENT: Individual project

One-paragraph proposal with form, due Monday, Jan 14, 2008

Scale of problem

* At least 7 different activities, processes, use cases
  + data entry
  + approvals
  + reports
  + fulfillment
* You should be familiar with almost all the activities

1/23/20008

Requirements (high, business, user) vs. Specifications (low, system specs, developer). Homework part 1 is a requirements document, not a specifications document.

* Requirements are unambiguous (key terms, system aspects, and player roles are defined)
* Requirements are hierarchical
* Consistent system for indicating requirements level (mandatory/optional/etc), either keywords or formatting
* Format:
  + Intro
  + definitions
  + general
  + details
  + changes/amendments/disclaimers
* Due Monday, 1/28: group work Requirements gathering techniques for New Century Healthcare
* Due Wednesday, 1/30: group work Solution to requirements

2/13/2008

1. Structured Analysis
   1. Process
      1. Context Diagram (Level 1 DFD)
         1. Expression of system scope
         2. Define interactions with external world
         3. Boxes for external actors and entities, circle for the system (specific name of project), labeled directional arrows for interactions (data flow)
         4. Not interested in extra-system activities
      2. Data-Flow Diagram
         1. Hierarchical system processes
         2. Models black-box inputs and outputs
         3. Decompose problem until each process is described by a p-spec (process specification)
         4. Level 0: entire system in one diagram. Level 1: models one process from level 0, etc. hierarchical detail expansion. Numbering syntax indexes into each diagram level. Each level 0 diagram should decompose at the same level.
         5. Directional arrows labeled with data name. Must appear at level 0 unless data is not persistent. Data flow, entity, and data store names must remain consistent through diagram levels.
         6. Names should be meaningful and add value on top of the shape
   2. Data
   3. Behavior
2. Object-oriented (UML)
   1. Static Structure Diagrams
      1. “Packages” (subsystems)
      2. Domain Models
      3. Class Models
         1. One per interface
         2. One for some controller objects
         3. One per entity
         4. One per “code” class
         5. Relationships
            1. Inheritance (generalization), supertype/subtype
            2. Associations (collaboration/communication)

Imply pointers or references

* + - * 1. Intersection (association)
        2. Aggregates (composition). Filled or unfilled diamonds indicate access to items within the collection (filled is private, unfilled is public)
        3. Functional (depends-on) relationships
  1. Dynamic
     1. Analysis: **Use-case scenarios**
        1. Specific user-centric sequences that accomplish a useful unit of work using the system
        2. Must provide user expectations
        3. Can usually be completed in one “session”
        4. Encompass both requirements and specifications
        5. My revisit/reiterate/decompose into more detail/more use cases over time
        6. No standard formal style but general expectations about content
        7. General Approaches
           1. Personaes – creative story about user role in accomplishing work in basic prose
           2. Standard scenario/cookbook approach (1. 2. 3., etc)
           3. Conversational approach: two columns showing the conversation between the user and the system
        8. Content
           1. Name/title
           2. Unique ID
           3. Actors (roles, groups, external systems)

Primary

Secondary

* + - * 1. Preconditions – assumed to be true
        2. Post-conditions – ending state, NOT an action. Probably eventually tied to a pre-condition in some other use-case.
        3. Triggers – what makes the scenario happen
        4. Scenario
        5. Variations
        6. Unhandled Exceptions (results in early exit from use-case if triggered)
        7. See Also
        8. Notes
      1. Example:
         1. UC1: Register for a class
         2. Actors: Student
         3. Preconditions: Registered student, authenticated
         4. Trigger: Selected “add class”
         5. Scenario:

Student provides dept code and course number

System checks student prerequisites

System shows list of available sections with class times, locations, and other information

Student selects the section to add

System checks for overlap in student’s schedule and display

System checks for class size opening and reserves spot if available

System checks for required entry codes and requests them from user

Request entry code

Validate entry code

Add student to class list

Show confirmation to student

Submit update to billing

* + 1. Analysis: Use-case diagrams
       1. UML high-level “context” view of system, used as a visual table-of-contents for use-cases (each circle is a use-case)
       2. Visually shows all actors and expected system functionality
       3. Analogous to Context Diagram
       4. Used to communicate system with all stakeholders
       5. Notion of scope (which external entities will interact with the system)
       6. May be refined/refactored over time
       7. Syntax:
          1. Stick-figures: Named actors (users/external systems)
          2. Ovals: named use-cases
          3. solid lines: interaction lines; may or may not have arrows, which indicate interaction; connects actors to use-cases
          4. hollow-headed arrow lines: <uses> (composition) lines and <extends> (inheritance) lines: only connects use-cases to other use-cases
    2. Analysis/Design: Robustiness Diagrams:
       1. Visually document the key objects required to implement one use-case
       2. Bridge between analysis (use-case) and design (sequence diagram)
       3. Based on Application Layer Model of Design
       4. Data/UI/Business Logic, AKA Model/View/Controller
       5. Allows parallel development and developer specialization with multiple classes potentially running on multiple machines
       6. Uses stick figures to represent actors, sideways ‘T’s connected to circles to represent interfaces (either user or external system), circles with inline arrows to represent processes (controller), and underlined circles to represent data stores.
       7. Actors can only talk to UI, and UI should flow through processes, instead of talking directly to data stores.
    3. Design: Activity diagrams (flow charts)
       1. UML flowchart
       2. Expand on the State Diagram syntax
       3. Document the flow of business processes (not data)
       4. Adds conditionals, parallelism, and synchronization
    4. Design: Sequence diagrams (1:1 mapping between to static structure diagram elements)
       1. Breakdown of a SINGLE use-case scenario into (1:1 correspondence) key individual methods/messages required to accomplish the use-case
       2. Identifies all required objects
       3. Timeline/sequence-driven
       4. Objects at top with vertical lifelines
       5. Messages as named directional arrows to and from the object; activation is vertical rectangle on lifeline representing call time.
       6. All calls have implied returns; optionally you may specify explicit returns
       7. Each method/message has parameters, returns, and expected functionality
       8. Should be able to create “header” for classes based on function calls
    5. Design: Collaboration diagrams (1:1 mapping between to static structure diagram elements)
       1. Reorganization of sequence in class-centric view
       2. Boxes are classes, named solid lines are methods, arrows are the direction of the call.
    6. Design: State diagrams (1:1 mapping between to static structure diagram elements)
       1. AKA state transition diagram
       2. Lifecycle of an object
       3. Interested in events that change an object’s state (“status”)
       4. Set business rules that allow certain events to occur and certain methods to be called
       5. Persistent objects may have longer lifecycles than dynamic objects
       6. All classes in a model potentially have a state diagram
       7. Syntax
          1. Box: named state. Data at rest, current status of an object
          2. One-way directional arrow: state transition. Method that causes a change in the state.
          3. [Guard/conditiona]l/method()
          4. Black dot with a downward-pointing vertical arrow away from the : constructor/initiator
          5. Circled black dot with an arrow pointing at it: destructor
    7. Implementation: Component diagrams (not covered in this class)