CS 48 MIDTERM REVIEW
SPECIFICS

• COME EARLY TO GET A DECENT SEAT
• 60 MINUTES MAX
• TOPICS
  • SWE BASICS: ACTIVITIES, PROCESSES (WATERFALL, SPIRAL, ITERATIVE)
    • PRACTICE: EXTREME, SCRUM
  • REQUIREMENTS SPECIFICATION: FEATURES, FURPS+ (FURPS-REACT)
  • DOMAIN ANALYSIS (STATIC (UML) AND DYNAMIC VIEWS)
  • DESIGN: ARCHITECTURE (UML), LAYERS, STATE, SEQUENCES
  • IMPLEMENTATION AND TESTING: UNIT, TDD, INTEGRATION, SYSTEM
  • EXTRA CREDIT: ACM CODE OF ETHICS
TOPICS: SWE BASICS

• ACTIVITIES
  • REQUIREMENTS SPECIFICATION/PLANNING, DOMAIN ANALYSIS (PROBLEM AREA), SYSTEM DESIGN (COMPUTER SOLUTION), IMPLEMENTATION, TESTING/MAINTENANCE

• PROCESS: WATERFALL, SPIRAL, & ITERATIVE: UNIFIED PROCESS AND AGILE
  • 3–4 PROS AND CONS OF EACH

• PRACTICE (IMPLEMENTATION OF THE PROCESS): EXTREME & SCRUM FOR AGILE
  • SCRUM
    • PRODUCT BACKLOG, SPRINT BACKLOG
    • STANDUP
    • SCRUM MEETING IS AT SAME TIME, EVERY DAY; EVERYONE MUST PARTICIPATE
    • VERY RIGID STRUCTURE (OTHER OPTIONS LIKE KANBAN (WORKFLOW VISUALIZATION/REFLECTION) CAN BE OVERLAYERED ON EXISTING PROCESSES)
WATERFALL MODEL

• ERA (80'S)
  • SHRINK WRAPPED SOFTWARE
  • SOFTWARE BECOMING COMPLEX: US GOVT. DEFENSE
  • COMPUTERS BECOMING CHEAPER & IN WIDER USE (HOME DESKTOPS)
  • NO INTERNET
  • SOFTWARE SHIPPED VIA SNAIL MAIL

• PROS:
  • EASY TO PARTITION WORK, EMPHASIZE IMPORTANCE OF REQUIREMENTS SPEC AND DESIGN, SPECIALIZED SKILLS/EXPERTISE

• CONS:
  • EVERYTHING ON PAPER, NEVER BACK UP
  • BUILD SW LIKE BUILDINGS
  • HIGH RISK ISSUES TACKLED LATE
  • NO TESTING/PROTOTYPING EARLY

• BAD IMPLEMENTATIONS HAVE LED TO DISASTROUS OUTCOMES: 70% OF PROJECTS NOT COMPLETED, 50% OF PROJECTS WITH 2X COST OVERRUNS, 1994=$80B IN CANCELLED PROJECTS
SPRAL MODEL

INCLUDES FREQUENT RISK ANALYSES
FREQUENT REEVALUATION DURING AN EXTENDED PLANNING STAGE

PROS
• RISKS ARE IDENTIFIED EARLY ON IN THE PROJECT
• RAPID PROTOTYPING SHOWS PROGRESS, GIVES USERS AN IDEA OF SYSTEM/PRODUCT
• FEEDBACK CAN BE OBTAINED EARLY FROM THESE PROTOTYPES

CONS
• TIME IS WASTED ON MAKING TOO MANY PROTOTYPES
• RISK ANALYSIS IS TOO TIME CONSUMING FOR SMALL PROJECTS
• IT’S A COMPLEX PROCESS
• IT CAN BE HARD TO SET MILESTONES AND SCHEDULE SW RELEASES
• IT CAN BE NEVER ENDING IF HURDLES ARE NOT OVERCOME
INCREMENTAL / ITERATIVE PROCESS

Requirements → Design → Implementation & Test & Integration & More Design → Final Integration & System Test

Time

Feedback from iteration N leads to refinement and adaptation of the requirements and design in iteration N+1.

Iterations are fixed in length, timeboxed.

The system grows incrementally.

4 weeks (for example)
ITERATING REDUCES RISK OVERALL

- Pros: Risk reduction, identify/deal with problems early when they are small, everyone contributes to and becomes expert in all SWE activities, build complex products incrementally

- Era (1995+): Software begins to change
- Programming languages change (more abstractions, tools, higher-level languages)
- Advent of the internet
- Browser-based software
- Remote services
- Software-as-a-service

In an iterative lifecycle, high-risk issues are tackled early, to drive down the riskiest project elements.
UNIFIED PROCESS (UP) 1999, 2000

- Iterative and Incremental through 4 phases/workflows/disciplines
  - Release at end of each timeboxed iteration
  - Inception – Explore feasibility, estimate costs
  - Elaboration – Architecture evaluation and its application
  - Construction – SW is designed, written, tested
  - Transition – Released, feedback collected

- "Like an online mentor that provides guidelines, templates, and examples for program development"
- Risk-driven
- Use case focused development
- Architecture-centric design
- UML for dynamic/static views
- Cons: too rigid, lots of planning, requires deep "object-oriented" skills
- "Owned" by IBM/Rational (RUP=Rational Unified Process) – to "fix" waterfall
“WE ARE UNCOVERING BETTER WAYS OF DEVELOPING SOFTWARE BY DOING IT AND HELPING OTHERS DO IT. THROUGH THIS WORK WE HAVE COME TO VALUE:

INDIVIDUALS AND INTERACTIONS OVER PROCESSES AND TOOLS
WORKING SOFTWARE OVER COMPREHENSIVE DOCUMENTATION
CUSTOMER COLLABORATION OVER CONTRACT NEGOTIATION
RESPONDING TO CHANGE OVER FOLLOWING A PLAN

THAT IS, WHILE THERE IS VALUE IN THE ITEMS ON THE RIGHT, WE VALUE THE ITEMS ON THE LEFT MORE”
AGILE DEVELOPMENT

• EMBRACE CHANGE; GOOD IF END GOAL IS UNKNOWN (VERY MODERN CONCEPT 201X); CONTINUOUS IMPROVEMENT VIA CUSTOMER FEEDBACK, COLLECTIVE OWNERSHIP, FAVORS THE SIMPLE

• PROS:
  • CAN LEAD TO FASTER, HIGHER QUALITY DELIVERY OF SOFTWARE (SW)
  • GOOD FOR SAAS, ENGAGES DEVELOPERS, CONTINUOUS DELIVERY, CUSTOMER FOCUSED, TEAM FIRST

• CONS
  • HARD TO PIN DOWN A DELIVERY DATE (HARD TO PLAN) B/C OF CONSTANT REPRIORITIZING; HARD TO ESTIMATE TIMINGS (SPRINTS ARE FIXED IN TIME)
  • ALL MEMBERS OF TEAM MUST BE HIGHLY SKILLED IN ALL ASPECTS ($$$)
  • REQUIRES THAT MEMBERS BE FULLY COMMITTED AND DEDICATED AND WILLING TO MAKE UP TIME WHEN TASKS GET DRAGGED OUT
  • FINAL PRODUCT CAN BE DIFFERENT
EXTREME PROGRAMMING (XP)

• METHODOLOGY USED TO IMPLEMENT AN AGILE PROCESS

1. SOFTWARE IS BUILT **ITERATIVELY**, WITH **FREQUENT RELEASES**

2. EACH RELEASE IMPLEMENTS THE SET OF **MOST VALUABLE FEATURES/USE-CASES/STORIES** THAT ARE CHOSEN BY THE CUSTOMER

3. EACH RELEASE IS IMPLEMENTED IN A **SERIES OF ITERATIONS**, EACH ITERATION ADDS MORE FEATURES/USE-CASES/STORIES

4. PROGRAMMERS TURN THE STORIES INTO **SMALLER-GRAINED TASKS**, WHICH THEY INDIVIDUALLY ACCEPT RESPONSIBILITY FOR

5. THE PROGRAMMER TURNS A TASK INTO A SET OF **TEST CASES** THAT WILL DEMONSTRATE THAT THE TASK IS FINISHED

6. **WORKING AS PAIRS**, THE PROGRAMMERS MAKE THE TEST CASES RUN, EVOLVING THE DESIGN IN THE MEANTIME TO MAINTAIN THE SIMPLEST POSSIBLE DESIGN FOR THE SYSTEM AS A WHOLE

• ALTERNATIVES: **KANBAN** (VISUAL WORKFLOW REFLECTION), **LEAN** (REDUCE WASTE), **FEATURE DRIVEN**, **SCRUM** (ALL ARE MINOR VARIATIONS)

See www.extremeprogramming.org
TOPICS: REQUIREMENTS

• FEATURES
  • USE CASES (KNOW THE FORMAT): NAME, ACTORS, PRECONDITION, FLOW, ALTERNATIVE PATHS, POSTCONDITIONS (TESTABLE)
• USER STORIES
  • AS A [ROLE], I CAN [FEATURE] SO THAT [REASON]
  • TITLE: GIVEN [CONTEXT], WHEN [EVENT], THEN [OUTCOME] //ACCEPTANCE CRITERIA (TESTABLE)
• IMPORTANCE OF PRIORITIZATION
• HOW THESE FIT INTO SCRUM & AGILE DEVELOPMENT PROCESSES

• FURPS+ (FURPS+ OR FURPS-REACT) – NONFUNCTIONAL REQUIREMENTS
EXAMPLE USE CASE: ONLINE HR SYSTEM

USE CASE: UPDATE BENEFITS
ACTORS: EMPLOYEE, EMPLOYEE ACCOUNT DATABASE, HEALTHCARE PLAN SYSTEM, INSURANCE PLAN SYSTEM
PRECONDITION: EMPLOYEE HAS LOGGED ON TO THE SYSTEM AND SELECTED “UPDATE BENEFITS” OPTION
FLOW OF EVENTS:
BASIC PATH:
1. SYSTEM RETRIEVES EMPLOYEE ACCOUNT FROM EMPLOYEE ACCOUNT DATABASE
2. SYSTEM ASKS EMPLOYEE TO SELECT MEDICAL PLAN TYPE; USES UPDATE MEDICAL PLAN
3. SYSTEM ASKS EMPLOYEE TO SELECT DENTAL PLAN TYPE; USES UPDATE DENTAL PLAN
   ...

ALTERNATIVE PATHS:
   IF HEALTH PLAN IS NOT AVAILABLE IN THE EMPLOYEE’S AREA THE EMPLOYEE IS INFORMED AND ASKED TO SELECT ANOTHER PLAN (EXCEPTIONAL CASES THAT MUST BE HANDLED)
   EMPLOYEE SELECTS CANCEL, LOGS OUT, OR LEAVES PAGE AT ANY POINT PRIOR TO CONFIRMING THE UPDATE (AN END-EARLY PATH)

POSTCONDITION: EMPLOYEE ACCOUNT PLAN TYPE HAS BEEN UPDATED IN THE EMPLOYEE ACCOUNT DATABASE OR NOTHING HAS CHANGED (END-EARLY PATHS)
THE FURPS+ MODEL: NONFUNCTIONAL REQUIREMENTS

- **FUNCTIONAL**
  - FEATURES, CAPABILITIES, SECURITY
- **USABILITY**
  - HUMAN FACTORS, HELP, DOCUMENTATION
- **RELIABILITY**
  - FAILURE FREQUENCY, RECOVERABILITY, PREDICTABILITY
- **PERFORMANCE**
  - RESPONSE TIME, THROUGHPUT, ACCURACY,...
- **SUPPORTABILITY**
  - ADAPTABILITY, MAINTAINABILITY, CONFIGURABILITY,...
- **+ - REACT: REUSE, ECONOMICS (COST), AESTHETICS, COMPREHENSIBILITY, TECHNOLOGY CONSTRAINTS**
TOPICS: DOMAIN ANALYSIS

- STATIC VIEWS: STATIC CLASS DIAGRAM
  - UML
    - CONCEPTS (CLASSES: CONCRETE AND ABSTRACT)
    - ATTRIBUTES (PRIMITIVE DATATYPES THAT DESCRIBE CONCEPT STATE)
  - ASSOCIATIONS: DEPENDENCY, GENERALIZATION, AGGREGATION
UML: DEPENDENCY

- WHEN A CLASS “USES” OR OTHERWISE DEPENDS ON ANOTHER CLASS TO FULFILL A RESPONSIBILITY
  - DASHED LINE WITH ARROW IN UML
UML: GENERALIZATION

• A.K.A., INHERITANCE – ONE CLASS IS DERIVED FROM ANOTHER
  • IN UML, TRIANGLE AT END OF LINE “POINTS” AT PARENT CLASS
UML: AGGREGATION & MULTIPLICITY

• “WHOLE” IS IDENTIFIED BY THE DIAMOND SHAPE AT THAT END OF THE LINE
TOPICS: DESIGN

• STATIC VIEWS -> SYSTEM ARCHITECTURE (STATIC CLASS DIAGRAM VIA UML)

• LAYERING

• SPECIFYING BEHAVIOR/ACTIVITY (DYNAMICS)
  • STATE DIAGRAMS (FOR COMPLEX OBJECTS)
  • SEQUENCE DIAGRAMS (INCLUDING COMMUNICATION DIAGRAMS)
    • FOR COMMUNICATION BETWEEN OBJECTS
    • SEQUENCE DIAGRAMS (SYNCHRONOUS MESSAGES, ASYNCHRONOUS MESSAGES, RETURN VALUES, OBJECT CREATION AND DELETION; OBJECT TIMELINES)
      • TYPES OF LINE AND ARROWS, INCLUDING TIMELINE
      • COMMUNICATION DIAGRAMS: FOR COMPLEX SEQUENCES

• DESIGN PRINCIPLES: EXPERT, COUPLING, COHESION

• DESIGN PATTERNS: STRUCTURAL, CREATIONAL, BEHAVIORAL
LAYERED ARCHITECTURES

• Concept – Each layer is a base for implementing layers above it
  • Ideally, knowledge and contact is ONE-WAY: DOWN ↓
  • Lower layers don’t even know upper layers exist
  • Interaction only with adjacent layer

• What are some good reasons to use layers?
  • Reduce complexity – Separate the domain from the implementation as much as possible
  • Increase modifiability, and REUSE potential
  • Easy to plug in off-the-shelf and 3rd party code
  • Encapsulation and data hiding
STATE DIAGRAMS

- PURPOSE: TO MODEL THE CHANGING STATES OF **COMPLEX OBJECTS**

```
Telephone

Idle  ->  Active
  
  off hook

  on hook

transition  event

initial state

state
```
SEQUENCE DIAGRAMS

- USE FOR SIMPLER INTERACTIONS – SEQUENCE EASILY SHOWN AS TOP-TO-BOTTOM INTERACTIONS
COMMUNICATION DIAGRAMS

```
c : Client

1: «create»
2: setActions(a, d, o)
3: «destroy»

«local»

link

Transaction

{transient}

object

message

«global»

p : ODBDProxy

2.1: setValues(d, 3.4)
2.2: setValues(a, "CO")
```
DESIGN PATTERNS: KNOW 3+ FROM EACH TYPE, DRAW THEIR UML

• 7 ARE STRUCTURAL PATTERNS – COMPOSITION OF CLASSES/OBJECTS
  • ADAPTOR, BRIDGE, COMPOSITE, DECORATOR, FAÇADE, FLYWEIGHT AND PROXY

• 5 ARE CREATIONAL PATTERNS – FOR CREATING OBJECTS
  • ABSTRACT FACTORY, BUILDER, FACTORY METHOD, PROTOTYPE, SINGLETON

• 11 ARE BEHAVIORAL PATTERNS – WAYS CLASSES/OBJECTS INTERACT
  • E.G., CHAIN OF RESPONSIBILITY, COMMAND, AND ... 9 MORE

• SEE HTTP://WWW.CS.UCSB.EDU/~CS48/SLIDES/DESIGNDIAGRAMS.PDF
  • HTTPS://SOURCEMAKING.COM/DESIGN_PATTERNS

• GREAT REFERENCE WITH EXAMPLES:
  HTTPS://GITHUB.COM/KAMRANAHMEDSE/DESIGN-PATTERNS-FOR-HUMANS
TOPICS: IMPLEMENTATION & TESTING

• UNIT TESTING
• INTEGRATION TESTING
• SYSTEM TESTING
• WHY THIS ORDER?
  • TEST PARTS BEFORE THE WHOLE

• TEST DRIVEN DEVELOPMENT
  • ADD TEST, RUN TEST (FAIL), WRITE CODE, RUN TEST (PASS),..., REPEAT, REFACTOR

• REGRESSION TESTING
  • RUN ALL TESTS ON EVERY COMMIT, WHY?

• TEST COVERAGE

• TESTING VS DEBUGGING