

The Software Process

1. Introduction
2. Requirements Capture
3. Software Processes
 - 3.1 Some Background
 - 3.2 Unified Process
 - 3.3 Extreme Programming
 - 3.4 Open Source Programming
4. Analysis
5. Design

497-3-1

Background about Processes

- Process:
 - a partially ordered set of steps
- Process model:
 - a set of similar processes
- Meta-process
 - Capability Maturity Model (CMM)
- Process principles
 - Quality management
 - Risk management
 - Role-based development
 - Milestones
 - Checkable
 - Low overhead
 - Appropriate to task and team

497-3-2

Processes

- Kinds of clients:
 - in-house
 - contract
 - off-the-shelf software (or: shrink-wrapped)
- Size of project / team
 - influences amount of planning and management
 - influences amount of paperwork
- Process needs to be appropriate for:
 - the product
 - the people
 - the tools

497-3-3

Usual Process Phases

- Software development lifecycle
 - Requirements / Analysis / Specification
 - Design
 - Implementation / Testing
 - Maintenance
- Software use lifecycle
 - Need identification
 - Development
 - Operation
 - Retiring

497-3-4

Products of Development

Activities

Requirements	informal text, understandable by user
Analysis	conceptual model, business model
Design	design model (dependent on programming language, operating system, libraries)
Implementation	program
Testing	fault-free program, test report
Maintenance	updated versions of program

497-3-5

Requirements Capture

- Determine what user/client wants
- Agree on a contract
- What problem is the program supposed to solve?
- Difficulties:
 - Users do not know what they want
 - Users may ask for impossible things
 - Requirements are usually vague, incomplete, contradictory
- Communication between user and developer

497-3-6

Analysis

- Understand the problem domain
 - Business modeling
 - Workflow modeling
- Look for unknown terms and concepts
 - Technical concepts of the user domain
 - Background knowledge needed to understand the problem
- Understand the tasks that users want to perform
- Identify conceptual classes and their relationships
 - Just enough modeling to express the tasks of the program

497-3-7

Design

- Take the analysis model and refine it into something that can be implemented
- Consider concerns of the programming language
 - Multiple inheritance?
 - Interfaces?
 - Built-in data types?
- Consider reuse
 - Standard libraries
 - Commercial software
- Other technical concerns
 - GUI
 - Database
 - Performance
- Add enough detail so that implementation will be easy

497-3-8

Implementation

- Working code!

497-3-9

Testing

- Find all kinds of faults and remove them
- Alpha-testing: in-house
- Beta-testing: by users
- Unit test: by developer
- System test: by opposing tester
- Faults introduced in:
 - Requirements
 - Analysis
 - Design
 - Implementation
- Different testing techniques find different faults

497-3-10

Maintenance

- All development after release
- Upgrades (“perfective”)
 - New features
- Fixes (“corrective”)
 - Problems that should have been solved before
- Reasons:
 - Users change their minds
 - Environment changes
 - Better technologies become available

497-3-11

Some Process Models

- Waterfall
 - sequential with little feedback
- Incremental
 - waterfall in a loop
- Synchronize and Stabilize
 - concurrent
- Spiral
 - risk management and verification
- Unified Software Development Process
 - complex, object-oriented, adaptable
- Extreme Programming
 - light-weight, flexible, small
- Open Source
 - widely distributed development

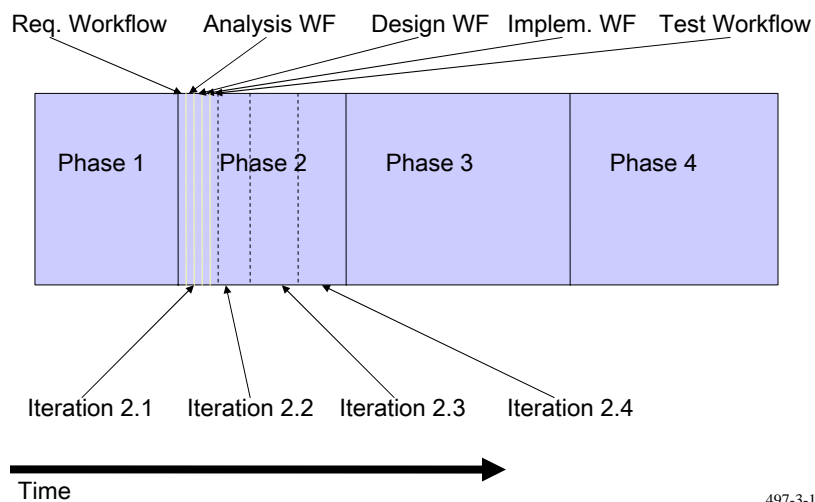
497-3-12

3.2 Unified Process

- Unified Software Development Process (USDP)
- A process to work with the UML
- Three principles:
 - use-case driven
 - architecture-centric
 - incremental and iterative
- Phases and workflows
 - phases structure whole development cycle:
Inception, Elaboration, Construction, Transition
 - core workflows structure tasks
Requirements, Analysis, Design, Implementation, Test
- Several iterations make up a phase
- Complex, configurable

497-3-13

USDP Timeline



497-3-14

USDP Workflows

- Requirements, Analysis, Design, Implementation, Test
- Each iteration goes through all workflows
- Each workflow can happen in all phases
- A USDP workflow is a set of related tasks
 - each kind of task is done by a worker (role)
 - each task produces an artifact
- The following slides list the workers for each workflow, and the kinds of tasks for each worker

497-3-15

Requirements Capture

Workflow

- Workers:
 - System Analyst
 - Find Actors and Use Cases
 - Structure Use Case Model
 - Use-Case Specifier
 - Detail a Use Case
 - User-Interface Designer
 - Prototype User Interface
 - Architect
 - Prioritize Use Cases
- Artifacts:
 - Domain model
 - Use-case model
 - User interface prototype
 - Supplementary specification

497-3-16

Analysis Workflow

- Workers:
 - Architect
 - Architectural Analysis
 - Use-Case Engineer
 - Analyze a Use Case
 - Component Engineer
 - Analyze a Class
 - Analyze a Package
- Artifacts:
 - Analysis model: classes, packages
 - Use-case realizations
 - Architecture description

497-3-17

Design Workflow

- Workers:
 - Architect
 - Architectural Design
 - Use-Case Engineer
 - Design a Use Case
 - Component Engineer
 - Design a Class
 - Design a Subsystem
- Artifacts:
 - Design model: subsystems, classes, operations, attributes, relations
 - Use-case realizations
 - Architecture
 - Deployment model

497-3-18

Implementation Workflow

- Workers:
 - Architect
 - Architectural Implementation
 - System Integrator
 - Integrate System
 - Component Engineer
 - Implement a Class
 - Implement a Subsystem
 - Perform Unit Test
- Artifacts:
 - Implementation of classes and interfaces (unit tested)
 - Updated architecture

497-3-19

Test Workflow

- Workers:
 - Test Engineer
 - Plan Test
 - Design Test
 - Evaluate Test
 - Component Engineer
 - Implement Test
 - Integration Tester
 - Perform Integration Test
 - System Tester
 - Perform System Test
- Artifacts:
 - Test cases
 - Test procedures
 - Test components (for automating tests)

497-3-20

USDP Phases

- Phases structure the process by time
 - Workflows structure the process by type of task
- Each phase consists of several iterations
- Inception, Elaboration, Construction, Transition

- Don't confuse phases, iterations, workflows!

497-3-21

USDP: Phases and Workflows

	Inception	Elaboration	Construction	Transition
Requirements				
Analysis				
Design				
Implementation				
Test				

Filled in: workflow is used a lot in this phase

497-3-22

Inception Phase

- Why do we do this project?
- How long?
- How expensive?
- Can it succeed?
- Deliverables:
 - feature list
 - draft of domain model
 - drafts of use-case model and analysis model
 - possibly: draft of design model, prototype
 - project plan

497-3-23

Elaboration Phase

- Identify most of the use cases
- Design architecture
- Trade-offs between requirements and architecture
- Develop first increments
- Make sure you have necessary resources
- Deliverables:
 - Complete domain model
 - Almost complete use case model
 - Almost complete analysis model
 - Draft of design model
 - First version of implementation, especially user interface
 - Plan of Construction and Transition

497-3-24

Construction Phase

- Create a beta release
- Adds in all the details
- Emphasis shifts from understanding to constructing
- Use cases are prioritized by business need
- Refactoring
- Lots of iterations: a few use cases in each
- Deliverables:
 - Executable, alpha-tested program
 - All models (use-case, analysis, design)
 - Draft of user manual
 - Plan for transition

497-3-25

Transition Phase

- Acceptance testing and beta-testing
- Fixing problems
- Installation
- Teaching users
- Things that cannot be done iteratively
- Deliverables:
 - Final release of the executable program
 - Installation software
 - Updated models and documentation
 - Setup of user support organization

497-3-26

3.3 Extreme Programming

- Agile, lightweight process for small teams
- Motivation:
 - old times: change was expensive
 - now: change is quick and cheap
- Lots of small iterations
- “Embrace Change”
- Change happens...
 - users change their minds
 - designs turn out not to work
 - project plans have to be updated
 - bugs are found
- In a waterfall process
 - change is expensive
- In XP
 - change is essential

497-3-27

XP Timeline

- Iteration:
 1. Customer picks user story with highest priority
 2. Split them into tasks
 3. Develop test cases for each task
 4. Implement task and evolve design of whole system
- Time Scale:
 - Release: months
 - Iteration: 1-4 weeks
 - User story: days
 - Test case: hours
 - Integration: several times a day
- User story
 - one paragraph describing a feature that the customer wants
 - similar to use case

497-3-28

12 XP Practices

- Planning Game
 - quickly estimate next release
 - prioritize user stories
 - negotiate with customer
- Small Releases
 - build the smallest working system possible
 - then upgrade it
- Metaphor
 - the common idea of the system
 - example: “an instant messenger is like a phone with text instead of voice”

497-3-29

12 XP Practices

- Simple Design
 - implement only the current user stories
 - remove features that are no longer needed
 - solve problems only when they occur
 - no redundancy
- Automated Testing
 - write unit tests before implementing
 - use JUnit or a similar tool
 - nothing is done until all the test run
- Refactoring
 - definition: change the design without changing the behavior
 - needed to keep the design clear when features are added

497-3-30

12 XP Practices

- Pair Programming
 - all production code is written with two programmers on one computer
 - improves code quality and code understanding
 - one person coding, one person thinking about the design
- Collective Ownership
 - anyone can change any piece of code at any time
 - automated tests show that it still works
- Continuous Integration
 - integrate each time a working piece of code has been written

497-3-31

12 XP Practices

- 40-hour Week
 - being awake is more important than working a lot
- On-site Customer
 - to answer questions about requirements
 - to negotiate the project plan
- Coding Standards
 - easily readable code
 - can be exchanged between programmers

497-3-32

Extreme Programming

- Workers:
 - Programmer
 - Customer
 - Tester
 - Coach
 - Tracker
- Advantages of XP
 - Reduces risk
 - Low overhead
 - Deals very well with changing requirements
 - Testing and refactoring keeps system simple
- Disadvantages of XP
 - Requires customer to be present
 - Not for big or complex projects
 - Not for programs that can't be tested automatically

497-3-33