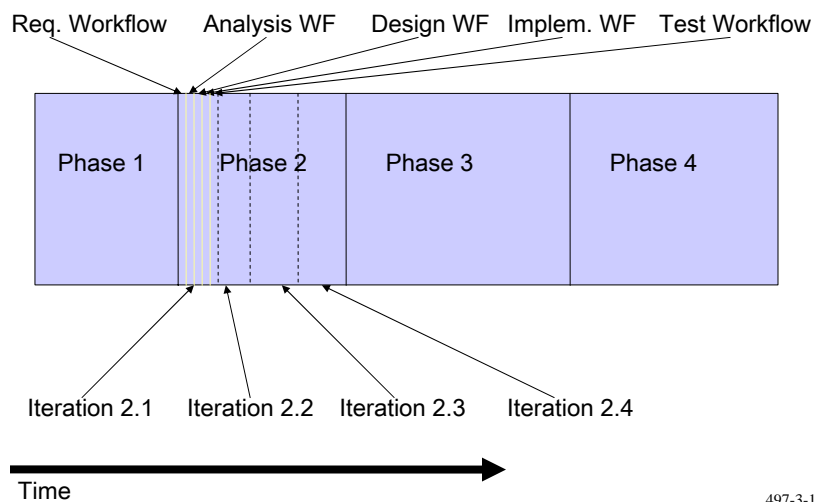


3.2 Unified Process

- 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

3.3 Open-Source Programming

- Open source software
 - gives users access to source
 - often freeware
 - often developed in a distributed way by volunteers
- Examples:
 - Linux: operating system
 - Gnu: file tools, editors, compilers,...
 - Apache: Web server
 - Argo/UML
- Web sites about open source projects
 - sourceforge.net
 - freshmeat.net
 - tigris.org
 - apache.org

497-3-34

Open Source Process

1. Initiator writes first prototype
 2. Release it
 3. Advertise it and find more developers
 4. Initiator delegates parts to other developers
 5. Users report bugs and request features
- Product phases:
 - pre-alpha
 - alpha
 - beta
 - released
 - Project tools
 - mailing list
 - configuration management tool (e.g. CVS, Subversion)
 - issue tracking tool (e.g. Bugzilla)

497-3-35

Principles of Open Source

- Projects start from a developer's personal need
- Reuse
- Plan to throw one away (Brooks, 11)
- Users are co-developers
- Release early, release often
- Any bug is easy to find with enough beta testers
- Create a community

from "The Cathedral and the Bazaar" by E. S. Raymond

497-3-36

Definition of Open Source

Software

- Open Source means: source code is available
- Often: available free of cost
- Often: developed by volunteers
- Different licensing models
 - public domain
 - everything is allowed
 - Gnu license
 - free
 - can modify and redistribute
 - but must stay free of charge and open
 - take a peek license
 - you can see the source
 - you can't copy or modify it

497-3-37

Economics of Open Source

- How do programmers of free software earn their money?
 - fame and fun instead of money...
 - work as consultants, provide services, teach
 - sell proprietary software (or hardware) that works with open-source products
 - for example: IBM
- Idea: Software is not a product, it's a service
 - charge for installation, customization, training
 - for example: Red Hat Linux
 - do not charge for basic development
 - similar to the way many consultants work

497-3-38

Open Source Programming

- Advantages
 - free and open = around forever
 - stability
 - distributed development
 - large number of volunteers possible
 - low bug rate
 - user feedback
- Disadvantages
 - project must be interesting to developers
 - projects are conservative: well-specified, well-known problems
 - limited amount of planning possible
 - no one to complain to

497-3-39