

Unified Process

- software development process
- who, what, when and how
- Jacobson, Booch, Rumbaugh
- OPEN,
- Object-oriented Software Process (OOSP)
- Agile Process
- Extreme Programming (XP)

Unified Process

- use case driven
 - user requirements captured using Use Cases
 - sequences of actions performed by system
 - basis for design and implementation
- architecture centric
 - structure of system and sub-systems
 - component interaction and communication
 - hardware deployment

Unified Process

- iterative
 - mini-project
 - analysed, designed, implemented, tested
 - baseline generated
- incremental
 - difference between consecutive baselines
 - gradual development to delivered system
 - step-wise refinement

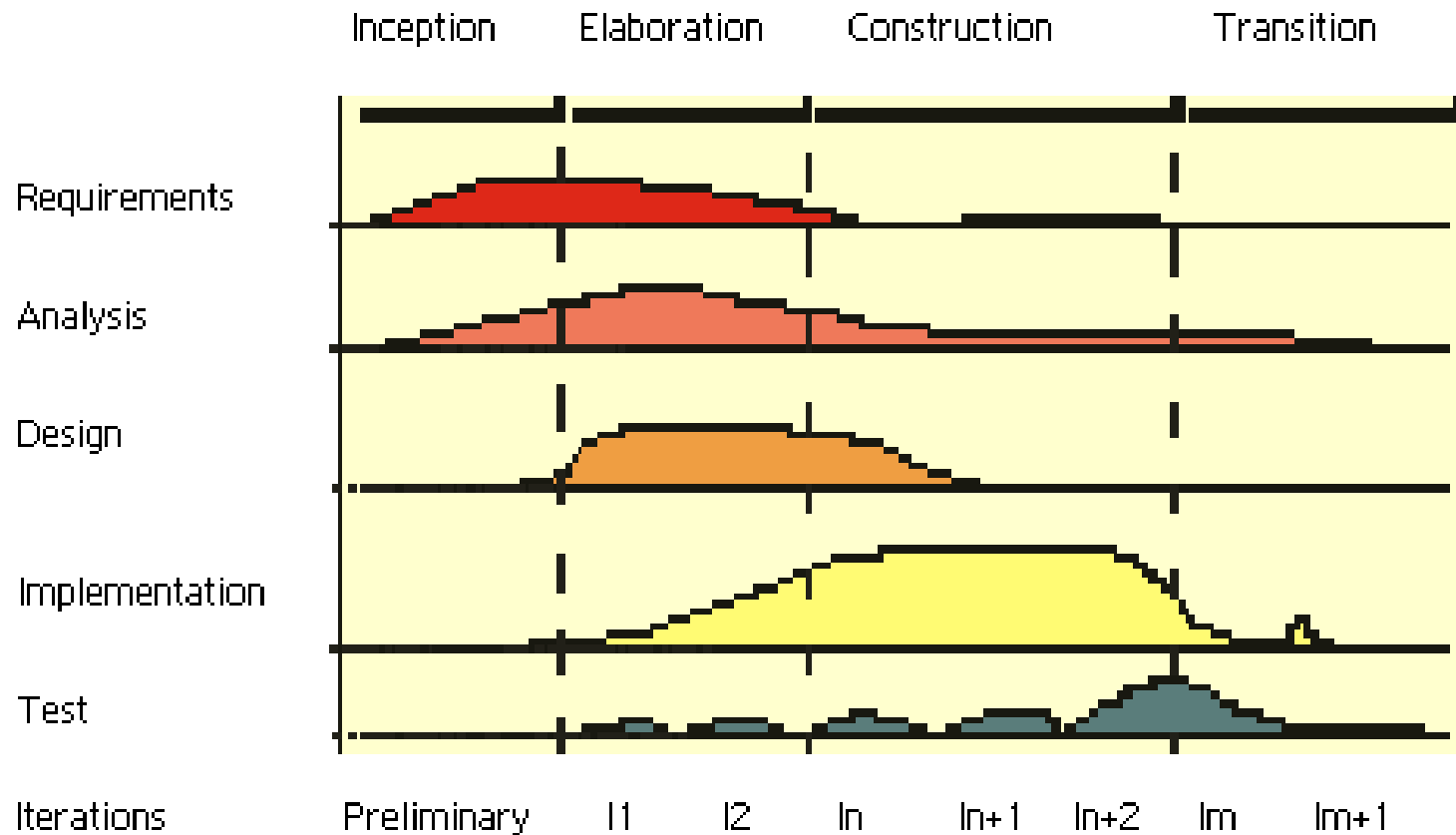
Unified Process

- iteration workflows
 - requirements, capture what system should do
 - analysis, refine requirements
 - design, realisation of requirements
 - implementation, producing software
 - test, verification of implementation

Unified Process

- UP lifecycle
 - inception
 - elaboration
 - construction
 - transition
- workflows each have associated lifecycle
- emphasis dependent on position of iteration within project

Unified Process



Inception

- establish feasibility - prototypes
- business case - quantifiable benefits
- capture requirements
- identify risks
- project manager, system architect
- requirements and analysis workflows
- test workflow not applicable

Inception

- deliverables
 - document stating requirements, features and constraints
 - initial use case model
 - project plan
 - business case
 - risk assessment document
 - prototypes (probably discarded)
 - initial architecture document

Elaboration

- executable architectural baseline - evolve into final system
- refine risk assessment
- define quality attributes
- produce use cases from functional requirements
- detailed plan of construction phase
- estimate costs, resources etc.

Elaboration

- all workflows involved, emphasis on first three
- deliverables
 - executable architectural baseline
 - UML models (static, dynamic and use case)
 - updated documents (requirements, risk assessment, business case, project plan)
 - contract (customer agreement)

Construction

- complete requirements, analysis and design
- architectural baseline to final system
- implementation workflow
- testing important
 - unit
 - integration
 - beta test

Construction

- deliverables
 - software product
 - UML model
 - test suite
 - user manuals
 - release details
 - project plan (compare actual to planned)

Transition

- implementation and test workflows
- deployment
- correcting faults found during beta test
- prepare for rollout
- modify software for user environment
- user support
- acceptance testing
- post project review

Transition

- deliverables
 - software product
 - user support plan
 - updated user manuals

Requirements

- inception and elaboration
- what system is required to do
 - user language, system requirements specification
 - high level specification
 - elicit and prioritise
 - negotiation, conflicting requirements
 - functional
 - non-functional (constraints)
- failure - major cause of project failure

Requirements

- non-functional
 - hardware, programming language
 - design objectives, criteria
 - economy
 - reliability
 - maintainability
 - robustness
 - integrity
 - security
 - design decisions
 - how e.g. file structure

Requirements

- use case modelling
 - functional requirements
 - find system boundary
 - identify actors
 - role of external entity
 - people or things
 - labels and short descriptions
 - identify use cases
 - things actors can do with system
 - scenarios
 - labels, verb phrases

Requirements

- finding actors
 - who/what uses system
 - roles, not specific individuals
 - system administration
 - interaction with other systems
 - sources and sinks of information
 - synchronous behaviour

Requirements

- identifying use cases
 - something an actor wants the system to do
 - how actor uses system
 - storage and retrieval of information
 - changes of state
 - external events
- iterative - stepwise refinement
- glossary of terms

Requirements

- detailing use cases
 - specification
 - actors involved
 - pre-conditions
 - flow, standard structures
 - post-conditions
 - scenarios, alternative paths

Requirements

Use case : Name/Scenario
ID :
Actors :
Pre-conditions
Scenario
1. Xxxxxx 2. Xxxxxx 2.1 yyyyyy 2.2 yyyyyy 3. Xxxxxx
Post-conditions

Analysis

- inception and elaboration
- models that capture required behaviour
- analysis classes, use case realisations
- language of customer/business
- elucidate important part of system, overview
- minimise coupling, consider inheritance
- understood by/useful to customer

Analysis

- analysis class
 - map to real world concept/feature
 - clear, obvious - name
 - well-defined set of responsibilities (contract)
 - high cohesion/low coupling
 - 3-5 responsibilities/class
 - not standalone
 - limit inheritance

Analysis

- finding classes
 - analyse documentation, interview
 - practice/experience
 - noun/verb
 - nouns/noun phrases classes and attributes
 - verbs/verb phrases responsibilities or operations
 - synonyms and homonyms
 - project glossary

Analysis

- finding classes
 - CRC analysis
 - class, responsibilities, collaborators
 - in conjunction with noun/verb
 - brainstorming - analyst, customer, domain expert
 - analyse information - analyst, domain expert
 - physical objects
 - paperwork e.g. invoices
 - interfaces e.g. screens, keyboards

Analysis

- use case realisation
 - sets of classes that realise the behaviour specified by a use case, class diagrams
 - interaction diagrams, collaboration and sequence diagrams
 - additional requirements sometimes identified
 - use case may need updating
 - activity diagrams (similar to flowcharts)

Analysis

- analysis model
 - typically 1 to 10% of classes
 - overview of system
 - how user requirements satisfied
 - useful for maintenance
 - logical architecture of system

Design

- elaboration and construction
- how functionality is to be implemented
- more detailed version of analysis
 - all classes identified
 - design classes (fully specified, attributes, operations, parameters, return types)

Design

- design classes
 - refine analysis classes
 - implementation details
 - solution domain
 - utilities - time, date, string
 - gui
 - middleware - communications, databases, components (e.g. CORBA)

Design

- well-formed design class
 - complete and sufficient - contract
 - primitive - simple, atomic unique
 - high cohesion - single, well-defined responsibility
 - low coupling - limited collaboration

Design

- inheritance
 - “is a kind of” relationship between classes
 - (aggregation - role played by)
 - reuse code
 - disadvantages
 - strongest possible coupling
 - weakens encapsulation
 - fixed at run-time

Design

- Relationships
 - aggregation
 - loose relationship, can exist independently
 - can be shared
 - many to one associations
 - composition
 - strong relationship
 - one to one association

Design

- Interface
 - similar to abstract class
 - no attributes
 - set of abstract operations
 - no associations
 - design to a contract, no implementation detail
 - interface realised by class
 - “plugs and sockets” to connect subsystems

Design

- interface
 - common sets of operations realised by many classes
 - reduce dependencies between classes, reduce coupling
 - more flexible solution than inheritance
 - component based systems
 - widely deployable, specifically designed and packaged to be assembled with other such objects
 - interface is key part of above approach

Design

- sub-systems
 - separate design elements
 - cohesive
 - represent large grained components
 - some public classes
 - black box approach, published interface

Design

- use case realisation
 - interaction diagrams
 - class diagrams
 - how, implementation details
 - easier to show more information on sequence diagram rather than collaboration diagram
 - statechart diagrams - dynamic behavior of classes

Implementation

- elaboration and construction, some work in transition
- produce executable code
- allocation of design classes to components
- deployment and implementation models
 - component diagram, dependencies between components
 - deployment diagram, physical computational nodes

Implementation

- components
 - ActiveX controls
 - JavaBeans
 - enterprise JavaBeans
 - servlets
 - JSP

Test

- verifying implementation
- each iteration tested (baseline)
- elaboration, construction and transition
- alpha testing (benevolent)
- beta testing (independent)
- acceptance testing (user site)