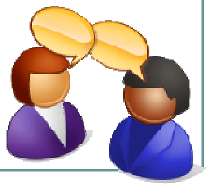
Professor Hausi A. Müller PhD PEng FCAE
Lorena Castañeda
Department of Computer Science
Faculty of Engineering
University of Victoria

http://www.engr.uvic.ca/~seng321/
https://courses1.csc.uvic.ca/courses/201/spring/seng/321

Students must participate in all project presentations in class & labs
No show results in a 25% reduction in the mark for that presentation

Project Deadlines and Marks



~~1.~~	~~Call for Project Proposals~~		~~**6 Jan (Class)**~~
~~2.~~	~~Request for Proposal (RFP)~~		~~**8 Jan**~~
~~3.~~	~~Project selection~~		~~**12 Jan (Lab)**~~
~~4.~~	~~Team selection~~		~~**14 Jan (Lab)**~~
⇨ 5.	Informal Requirements Definition (S0)	5%	**21 Jan (Lab)**
⇨ 6.	Project website up and running (S0)	5%	**21 Jan (Lab)**
7.	Customer Feedback on S0 (C0)	5%	**26 Jan (Lab)**
8.	Formal Requirements Spec (S1)	10%	**16 Feb (Lab)**
9.	Customer Feedback on S1 (C1)	5%	**18 Feb (Lab)**
10.	Detailed Requirements Spec (S2a)	10%	**1 Mar (Lab)**
11.	Prototype demo (S2b)	5%	**3 Mar (Lab)**
12.	Customer Feedback on S2a-b (C2)	5%	**8 Mar (Lab)**
13.	Final Requirements Spec (S3a)	15%	**15 Mar (Lab)**
14.	User Manual (S3b)	10%	**22 Mar (Lab)**
15.	Customer Feedback on S3a-b (C3)	5%	**24 Mar (Lab)**
16.	Demo Final Project (S4)	10%	**29,31 Mar (Lab)**
17.	Customer Feedback on S4 (C4)	5%	**29,31 Mar (Lab)**
18.	Instructor and TA Evaluations (S5)	5%	**1 Apr**

What is Quality (Pressman)?

- Conformance to explicitly stated requirements, standards, and implicit characteristics
- Functional and non-functional **requirements**
  - Foundation from which quality is measured
  - Lack of conformance ← → lack of quality
- Explicitly documented development **standards**
  - Development criteria guide manner software engineered
  - Criteria not followed → lack of quality
- Implicit characteristics expected of professionally developed software
  - Often go unmentioned (e.g., desire for good maintainability)
  - Even if explicit requirements met, failing to meet implicit requirements suggest suspect software quality

Software qualities

- Software engineering is concerned with software qualities
- Qualities (a.k.a. “ilities”) are goals in the practice of software engineering
- The qualities are usually expressed as non-functional requirements during the early design stages

Software qualities …

- External qualities
  - visible to the user
  - reliability, efficiency, usability
- Internal qualities
  - the concern of developers
  - they help developers achieve external qualities
  - verifiability, maintainability, extensibility, evolvability, adaptability
- Product qualities
  - concern the developed artifacts
  - maintainability, understandability, performance
- Process qualities
  - deal with the development activity
  - products are developed through process
  - maintainability, productivity, timeliness

Software Development Life Cycles (SDLC)

“You’ve got to be very careful if you don’t know where you’re going, because you might not get there.”

Yogi Berra

Capability Maturity Model (CMM)

- A bench-mark for measuring the maturity of an organization’s software process
- CMM defines 5 levels of process maturity based on certain Key Process Areas (KPA)

CMM Levels

Level 5 – Optimizing (< 1%)

- -- process change management
- -- technology change management
- -- defect prevention

Level 4 – Managed (< 5%)

- -- software quality management
- -- quantitative process management

Yogi Berra

Level 3 – Defined (< 10%)

- -- peer reviews
- -- intergroup coordination
- -- software product engineering
- -- integrated software management
- -- training program
- -- organization process definition
- -- organization process focus

Level 2 – Repeatable (~ 15%)

- -- software configuration management
- -- software quality assurance
- -- software project tracking and oversight
- -- software project planning
- -- requirements management

Level 1 – Initial (~ 70%)

CMMI (Capability Maturity Model Integration)

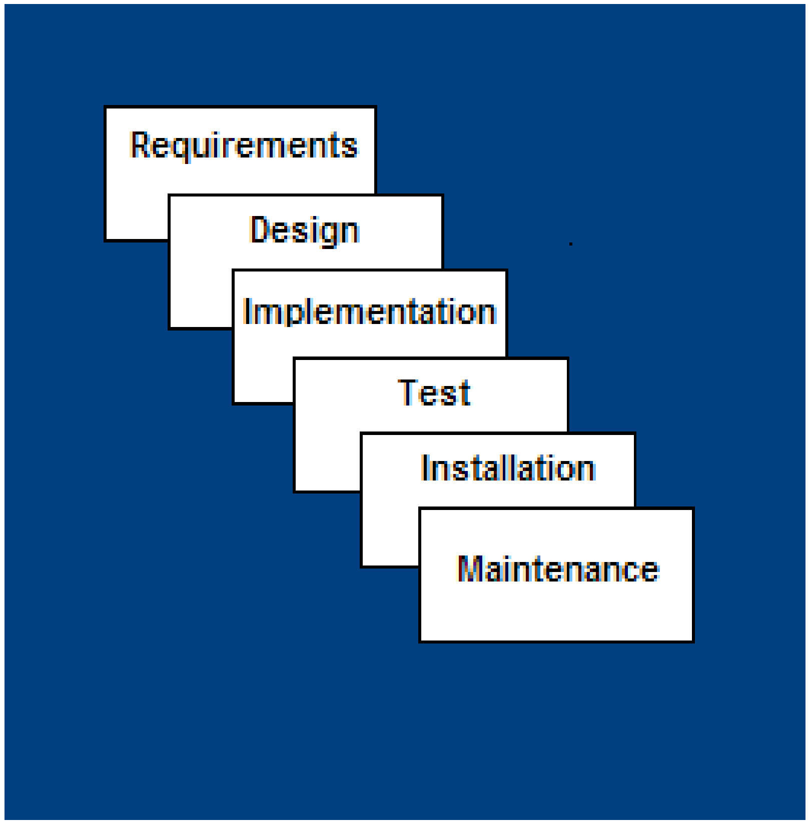
- CMM 1990
  - Lack of integration
  - Limitation of the Key Performance Areas (KPA)
  - Activity-based approach
  - Paperwork
- CMMI 2006, areas of interest:
  - Product and service development
  - Service establishment, management
  - Product and service acquisition

Level 5 - Optimizing
Level 4 - Quantitatively Managed
Level 3 – Defined
Level 2 – Managed
Level 1 – Initial

Software Development Life Cycles (SDLC) Model

A framework that describes the activities performed at each stage of a software development project.

Waterfall Model



- **Requirements** – defines needed information, function, behavior, performance and interfaces.
- **Design** – data structures, software architecture, interface representations, algorithmic details.
- **Implementation** – source code, database, user documentation, testing.

Waterfall Strengths

- Easy to understand, easy to use
- Provides structure to inexperienced staff
- Milestones are well understood
- Sets requirements stability
- Good for management control (plan, staff, track)
- Works well when quality is more important than cost or schedule

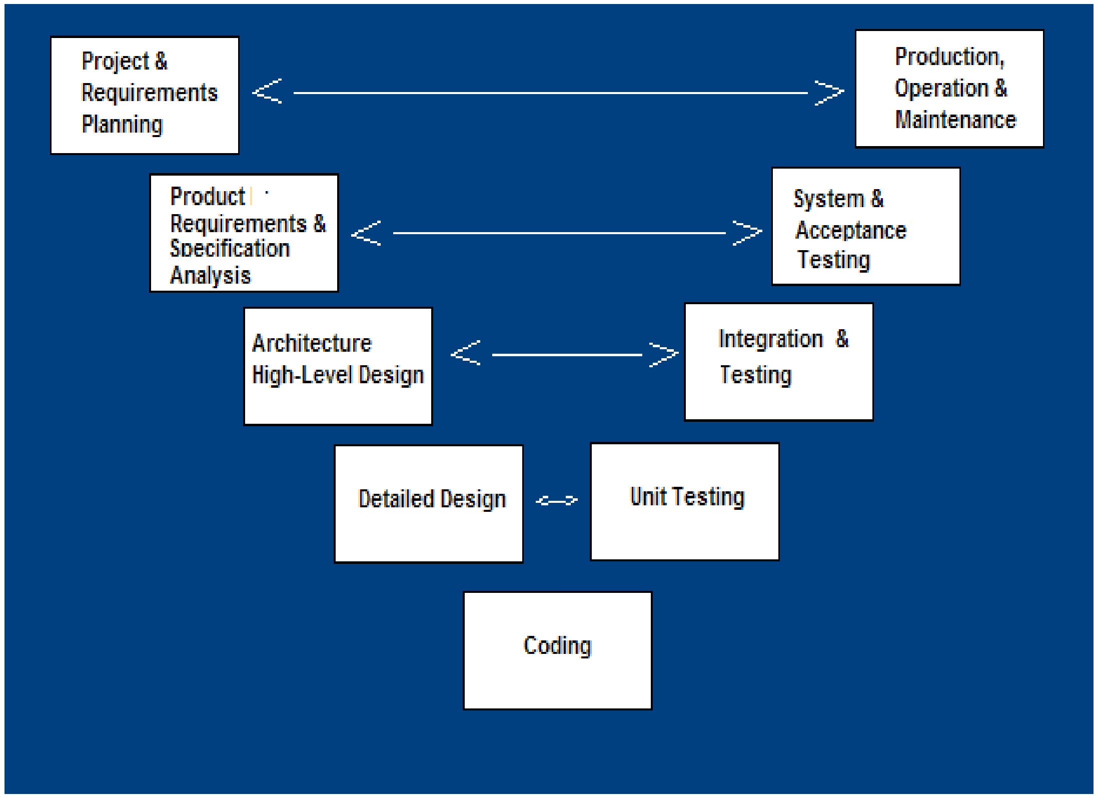
Waterfall Deficiencies

- All requirements must be known upfront
- Deliverables created for each phase are considered frozen – inhibits flexibility
- Can give a false impression of progress
- Does not reflect problem-solving nature of software development – iterations of phases
- Integration is one big bang at the end
- Little opportunity for customer to preview the system (until it may be too late)

When to use the Waterfall Model

- Requirements are very well known
- Product definition is stable
- Technology is understood
- New version of an existing product
- Porting an existing product to a new platform.
  - High risk for new systems because of specification and design problems.
  - Low risk for well-understood developments using familiar technology.

V-Shaped SDLC Model


Project & Requirements Planning
Production, Operation & Maintenance
Product Requirements & Specification Analysis
System & Acceptance Testing
Architecture High-Level Design
Integration & Testing
Detailed Design
Unit Testing
Coding

- A variant of the Waterfall that emphasizes the verification and validation of the product.
- Testing of the product is planned in parallel with a corresponding phase of development

V-Shaped Steps

- Project and Requirements Planning – allocate resources
- Product Requirements and Specification Analysis – complete specification of the software system
- Architecture or High-Level Design – defines how software functions fulfill the design
- Detailed Design – develop algorithms for each architectural component

- Production, operation and maintenance – provide for enhancement and corrections
- System and acceptance testing – check the entire software system in its environment
- Integration and Testing – check that modules interconnect correctly
- Unit testing – check that each module acts as expected

•Coding – transform algorithms into software

V-Shaped Strengths

- Emphasize planning for verification and validation of the product in early stages of product development
- Each deliverable must be testable
- Project management can track progress by milestones
- Easy to use

V-Shaped Weaknesses

- Does not easily handle concurrent events
- Does not handle iterations or phases
- Does not easily handle dynamic changes in requirements
- Does not contain risk analysis activities

When to use the V-Shaped Model

- Excellent choice for systems requiring high reliability – hospital patient control applications
- All requirements are known up-front
- When it can be modified to handle changing requirements beyond analysis phase
- Solution and technology are known

Prototyping: Basic Steps

- Identify basic requirements
  - Including input and output info
  - Details (e.g., security) generally ignored
- Develop initial prototype
  - UI first
- Review
  - Customers/end –users review and give feedback
- Revise and enhance the prototype & specs
  - Negotiation about scope of contract may be necessary

Dimensions of Prototyping

- Horizontal prototype
  - Broad view of entire system/sub-system
  - Focus is on user interaction more than low-level system functionality (e.g. , databsae access)
  - Useful for:
    - Confirmation of UI requirements and system scope
    - Demonstration version of the system to obtain buy-in from business/customers
    - Develop preliminary estimates of development time, cost, effort

Dimensions of Prototyping

- Vertical prototype
  - More complete elaboration of a single sub-system or function
  - Useful for:
    - Obtaining detailed requirements for a given function
    - Refining database design
    - Obtaining info on system interface needs
    - Clarifying complex requirements by drilling down to actual system functionality

Types of Prototyping

- Throwaway /rapid/close-ended prototyping
  - Creation of a model that will be discarded rather than becoming part of the final delivered software
  - After preliminary requirements gathering, used to visually show the users what their requirements may look like when implemented
- Focus is on quickly developing the model
  - not on good programming practices
  - Can Wizard of Oz things

Throwaway Prototyping steps

- Write preliminary requirements
- Design the prototype
- User experiences/uses the prototype, specifies new requirements
- Repeat if necessary
- Write the final requirements
- Develop the real products

Evolutionary Prototyping

- Aka breadboard prototyping
- Goal is to build a very robust prototype in a structured manner and constantly refine it
- The evolutionary prototype forms the heart of the new system and is added to and refined
- Allow the development team to add features or make changes that were not conceived in the initial requirements

Evolutionary Prototyping steps

- Developers build a prototype during the requirements phase
- Prototype is evaluated by end users
- Users give corrective feedback
- Developers further refine the prototype
- When the user is satisfied, the prototype code is brought up to the standards needed for a final product.

Evolutionary Prototyping Strengths

- Customers can “see” the system requirements as they are being gathered
- Developers learn from customers
- A more accurate end product
- Unexpected requirements accommodated
- Allows for flexible design and development
- Steady, visible signs of progress produced
- Interaction with the prototype stimulates awareness of additional needed functionality

Incremental Prototyping

- Final product built as separate prototypes
- At the end, the prototypes are merged into a final design

Extreme Prototyping

- Often used for web applications
- Development broken down into 3 phases, each based on the preceding 1
  - Static prototype consisting of HTML pages
  - Screens are programmed and fully functional using a simulated services layer
    - Fully functional UI is developed with little regard to the services, other than their contract
  - Services are implemented

Prototyping Advantages

- Reduced time and cost
  - Can improve the quality of requirements and specifications provided to developers
    - Early determination of what the user really wants can result in faster and less expensive software
- Improved/increased user involvement
  - User can see and interact with the prototype, allowing them to provide better/more complete feedback and specs
  - Misunderstandings/miscommunications revealed
  - Final product more likely to satisfy their desired look/feel/performance

Disadvantages of Prototyping 1

- Insufficient analysis
  - Focus on limited prototype can distract developers from analyzing complete project
  - May overlook better solutions
  - Conversion of limited prototypes into poorly engineered final projects that are hard to maintain
  - Limited functionality may not scale well if used as the basis of a final deliverable
    - May not be noticed if developers too focused on building prototype as a model

Disadvantages of Prototyping 2

- User confusion of prototype and finished system
  - Users can think that a prototype (intended to be thrown away) is actually a final system that needs to be polished
    - Unaware of the scope of programming needed to give prototype robust functionality
  - Users can become attached to features included in prototype for consideration and then removed from final specification

Disadvantages of Prototyping 3

- Developer attachment to prototype
  - If spend a great deal of time/effort to produce, may become attached
  - Might try to attempt to convert a limited prototype into a final system
    - Bad if the prototype does not have an appropriate underlying architecture

Disadvantages of Prototyping 4

- Excessive development time of the prototype
  - Prototyping supposed to be done quickly
  - If developers lose sight of this, can try to build a prototype that is too complex
  - For throw away prototypes, the benefits realized from the prototype (precise requirements) may not offset the time spent in developing the prototype – expected productivity reduced
  - Users can be stuck in debates over prototype details and hold up development process

Disadvantages of Prototyping 5

- Expense of implementing prototyping
  - Start up costs of prototyping may be high
  - Expensive to change development methodologies in place (re-training, re-tooling)
  - Slow development if proper training not in place
    - High expectations for productivity unrealistic if insufficient recognition of the learning curve
  - Lower productivity can result if overlook the need to develop corporate and project specific underlying structure to support the technology

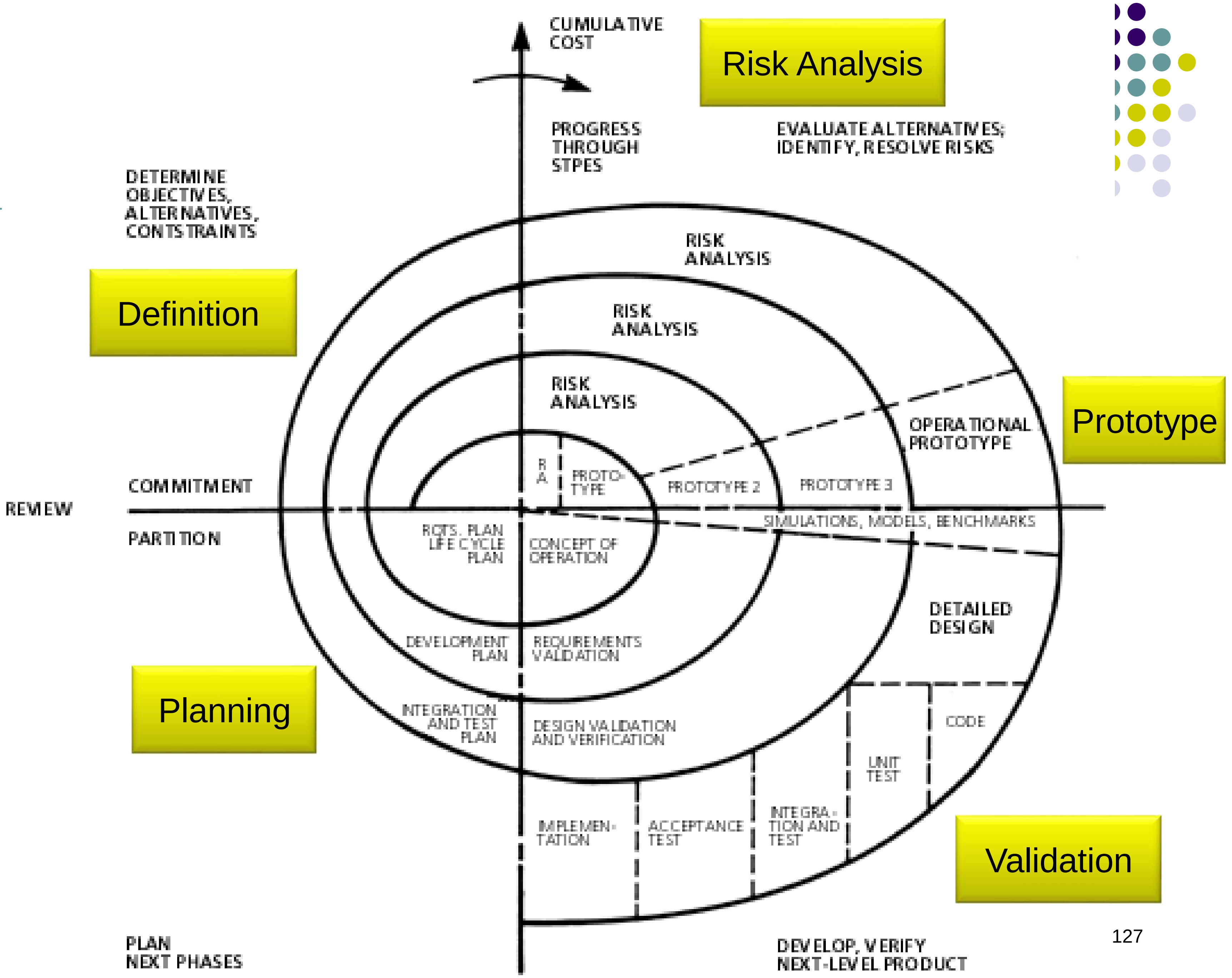
Best Uses of Prototyping

- Most beneficial for systems that will have many interactions with end users
- The greater the interaction between the computer and the user, the greater the benefit of building a quick system for the user to play with
- Especially good for designing good human-computer interfaces

Life Cycle Models
Spiral model

- Barry Boehm
  - A Spiral Model of Software Development and Enhancement, *ACM SIGSOFT Software Engineering Notes,* Aug 1986
  - A Spiral Model of Software Development and Enhancement, *IEEE Computer,* Vol. 21, No. 5, pp 61-72, May 1988
- Basic idea: evolutionary development
  - Using the waterfall model for each step or cycle
  - Intended to help manage risks by providing feedback along the way
  - Don't define in detail the entire system at first; develop prototype
  - Developers should only define the highest priority features
  - Define and implement those, then get feedback from users/customers
  - This feedback distinguishes “evolutionary” from “incremental” development
  - With this knowledge, developers then go back to define and implement more features in smaller chunks

Model is risk-driven
Iterative requirements analysis


CUMULATIVE COST
Risk Analysis
PROGRESS THROUGH STPES
EVALUATE ALTERNATIVES; IDENTIFY, RESOLVE RISKS
DETERMINE OBJECTIVES, ALTERNATIVES, CONSTRAINTS
Definition
RISK ANALYSIS
RISK ANALYSIS
RISK ANALYSIS
OPERATIONAL PROTOTYPE
Prototype
R A
PROTO-TYPE
PROTOTYPE 2
PROTOTYPE 3
COMMITMENT
REVIEW
PARTITION
SIMULATIONS, MODELS, BENCHMARKS
RQTS. PLAN LIFE CYCLE PLAN
CONCEPT OF OPERATION
DETAILED DESIGN
DEVELOPMENT PLAN
REQUIREMENTS VALIDATION
Planning
INTEGRATION AND TEST PLAN
DESIGN VALIDATION AND VERIFICATION
CODE
UNIT TEST
IMPLEMEN-TATION
ACCEPTANCE TEST
INTEGRA-TION AND TEST
Validation
PLAN NEXT PHASES
DEVELOP, VERIFY NEXT-LEVEL PRODUCT

Spiral Model Strengths

- Provides early indication of insurmountable risks, without much cost
- Users see the system early because of rapid prototyping tools
- Critical high-risk functions are developed first
- The design does not have to be perfect
- Users can be closely tied to all lifecycle steps
- Early and frequent feedback from users
- Cumulative costs assessed frequently

Spiral Model Weaknesses

- Time spent for evaluating risks too large for small or low-risk projects
- Time spent planning, resetting objectives, doing risk analysis and prototyping may be excessive
- The model is complex
- Risk assessment expertise is required
- Spiral may continue indefinitely
- Developers must be reassigned during non-development phase activities
- May be hard to define objective, verifiable milestones that indicate readiness to proceed through the next iteration

When to use Spiral Model

- When creation of a prototype is appropriate
- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex
- New product line
- Significant changes are expected (research and exploration)

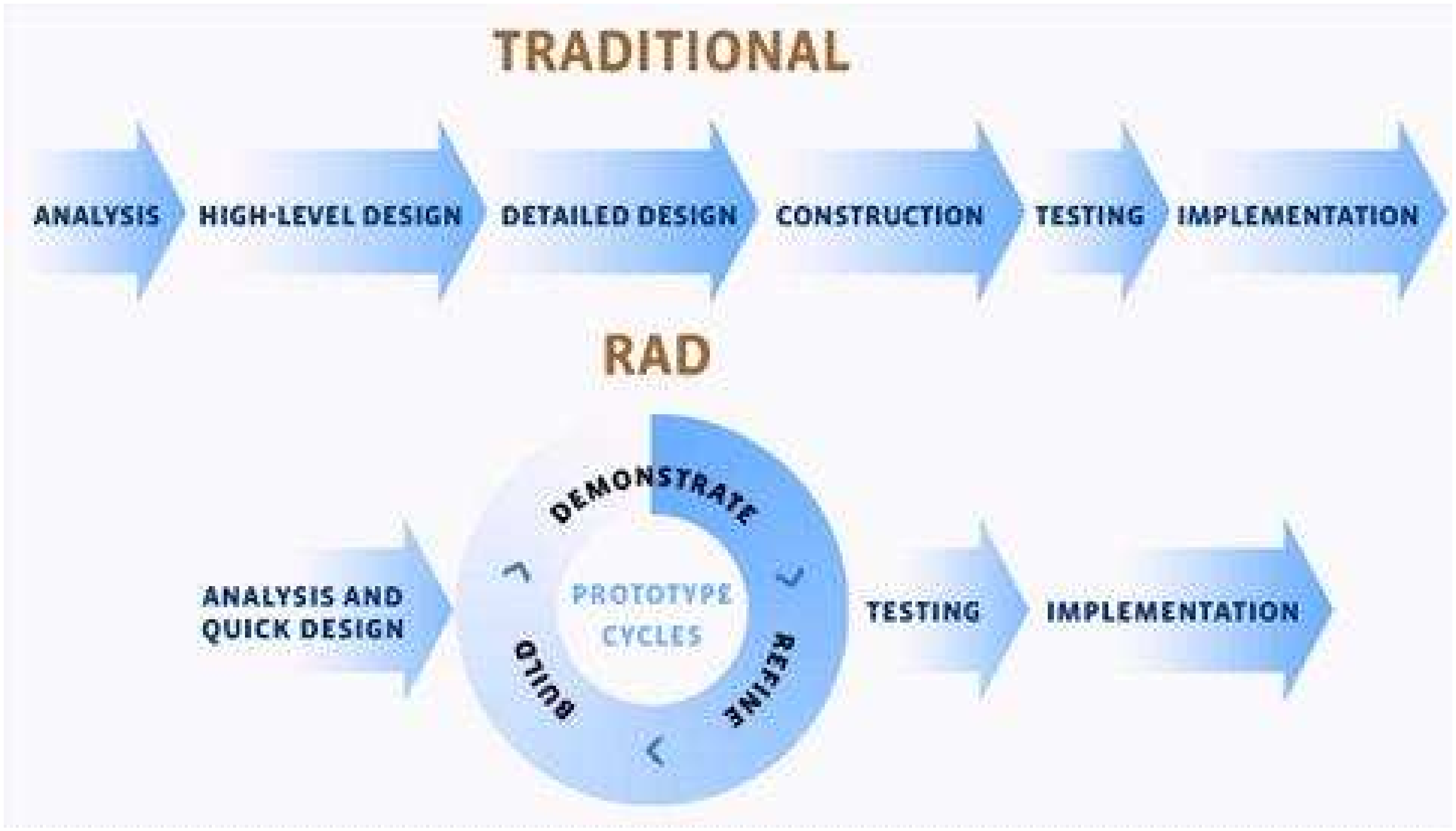
Agile SDLCs

- Speed up or bypass one or more life cycle phases
- Usually less formal and reduced scope
- Used for time-critical applications
- Used in organizations that employ disciplined methods

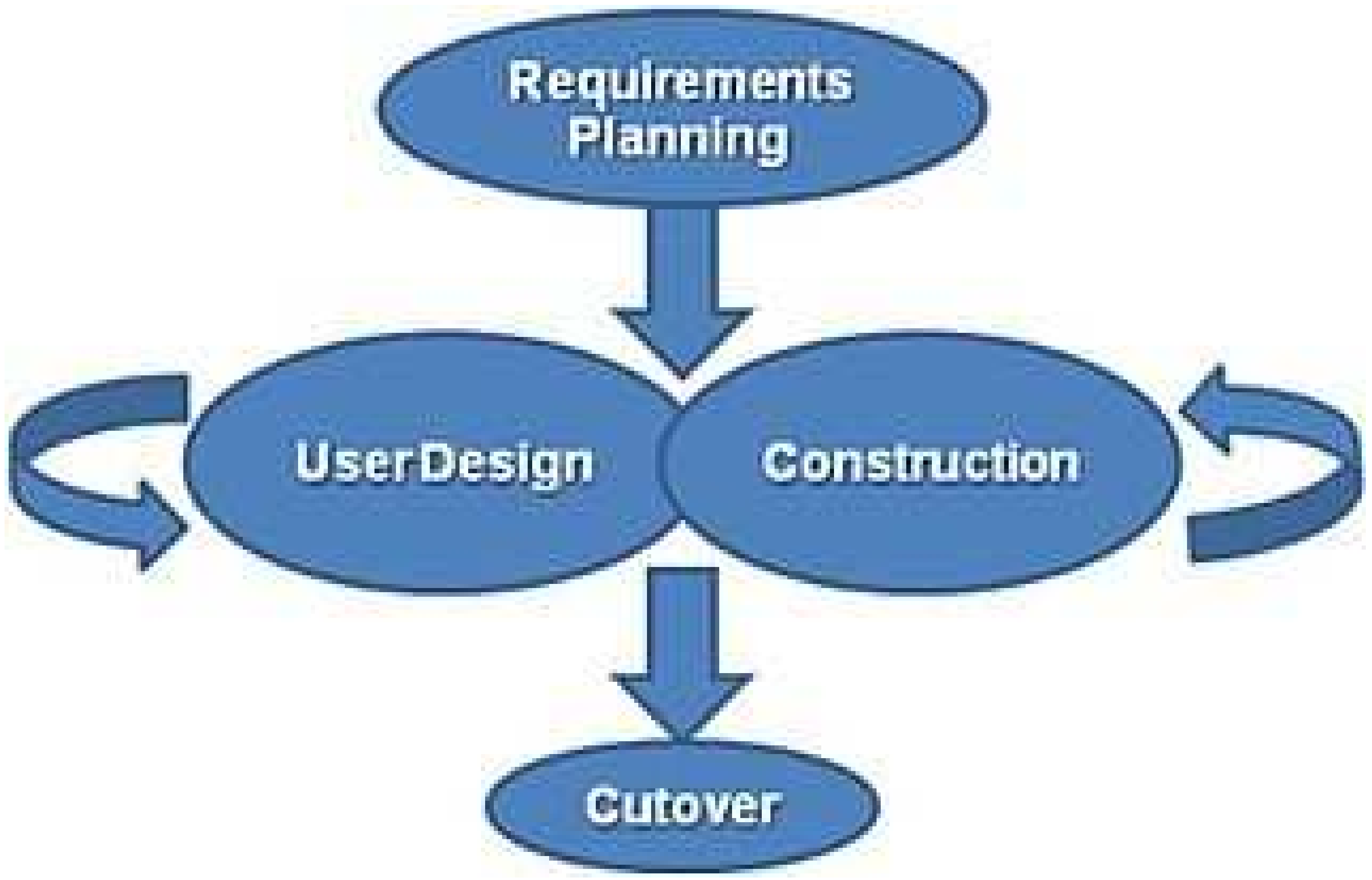
Some Agile Methods

- Rapid Application Development (RAD)
- Incremental SDLC
- Scrum
- Extreme Programming (XP)
- Adaptive Software Development (ASD)
- Feature Driven Development (FDD)
- Crystal Clear
- Dynamic Software Development Method (DSDM)
- Rational Unify Process (RUP)

Rapid Application Development (RAD)



Rapid Application Development (RAD)



Rapid Application Model (RAD)

- Requirements planning phase (structured discussion of business problems)
- User description phase – automated tools capture information from users
- Construction phase – productivity tools, such as code generators, screen generators, etc. inside a time-box. (“Do until done”)
- Cutover phase -- installation of the system, user acceptance testing and user training

Requirements Planning Phase

- Combines elements of the system planning and systems analysis phases of the System Development Life Cycle (SDLC).
- Users, managers, and IT staff members discuss and agree on business needs, project scope, constraints, and system requirements.
- It ends when the team agrees on the key issues and obtains management authorization to continue.

User Design Phase

- Users interact with systems analysts and develop models and prototypes that represent all system processes, inputs, and outputs.
- Typically use a combination of Joint Application Development (JAD) techniques and CASE tools to translate user needs into working models.
- *A* continuous interactive process that allows users to understand, modify, and eventually approve a working model of the system that meets their needs.

Construction Phase

- Focuses on program and application development task similar to the SDLC.
- However, users continue to participate and can still suggest changes or improvements as actual screens or reports are developed.
- Its tasks are programming and application development, coding, unit-integration, and system testing.

Cutover Phase

- Resembles the final tasks in the SDLC implementation phase.
- Compared with traditional methods, the entire process is compressed. As a result, the new system is built, delivered, and placed in operation much sooner.
- Tasks are data conversion, full-scale testing, system changeover, user training.

RAD Strengths

- Reduced cycle time and improved productivity with fewer people means lower costs
- Time-box approach mitigates cost and schedule risk
- Customer involved throughout the complete cycle minimizes risk of not achieving customer satisfaction and business needs
- Focus moves from documentation to code (WYSIWYG).
- Uses modeling concepts to capture information about business, data, and processes.

RAD Weaknesses

- Accelerated development process must give quick responses to the user
- Risk of never achieving closure
- Hard to use with legacy systems
- Requires a system that can be modularized
- Developers and customers must be committed to rapid-fire activities in an abbreviated time frame.

When to use RAD

- Reasonably well-known requirements
- User involved throughout the life cycle
- Project can be time-boxed
- Functionality delivered in increments
- High performance not required
- Low technical risks
- System can be modularized