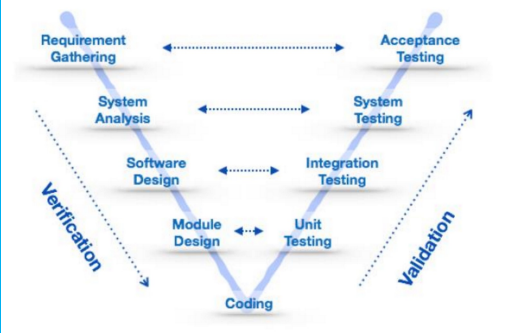


Software Models

September 19, 2023 10:26 AM

Model Type	Principles	Advantages/Most Appropriate	Disadvantages/Least Appropriate	When to use
Waterfall	<ol style="list-style-type: none"> 1. System & Software Requirements: Captured in requirements documentation 2. Analysis: models, schema and business rules 3. Design: Software architecture is the result 4. Coding: Development, proving and integration of software 5. Testing: Structured discovery & debugging of defects 6. Operations: installation, migration, support, and maintenance of complete systems <p>A more realistic model is the Waterfall Model with "iterative feedback loops"</p> <ul style="list-style-type: none"> • Where after each step, you go back and check 	<ul style="list-style-type: none"> • Project has clear objectives & solutions • Large and complicated • Requirements are unchanging • Required for formal approvals at designated milestones 	<ul style="list-style-type: none"> • Large projects where requirements are not well understood or changing for any reasons • Event-driven systems • Real-time system • Can only move to next stage when previous are already completed 	
Prototyping	<ul style="list-style-type: none"> • About creating incomplete versions of the software being developed • Basics are: <ul style="list-style-type: none"> • Not standalone, complete development methodology; rather an approach to try particular features in context of a full methodology • Attempts to reduce inherent risk to project by dividing it into smaller segments and ease-of-change during the development process • Client is involved during the development process; likelihood of acceptance increases of final iteration • Possible some cases to evolve from prototype to working system; some prototypes can be discarded 	<ul style="list-style-type: none"> • Users are active in development • Working model of system is provided; user get experience with system being developed • Errors can be caught earlier • Quicker feedback is available; leads to better solutions • Missing functionality can be found easily • Confusing/difficult functions can be identified 	<ul style="list-style-type: none"> • Leads to implementing, repairing way of building systems • Could increase the complexity of system as scope may expand beyond original plans • Incomplete application may cause app not to be used as the fully system was designed 	<ul style="list-style-type: none"> • When desired system needs to have many interactions with end users • Online systems, web interfaces have high amount of interaction. Could take a while for system to be built that allows easy use and end user to not be trained • Ensures end users constantly work with system and provide feedback which is then implemented in the prototype to result in a useable system • Great for good human computer interface systems
Iterative & Incremental Development	<ul style="list-style-type: none"> • Combination of both iterative design or methods and incremental build model • Essential parts of modified waterfall models • Incremental development <ul style="list-style-type: none"> • Primary objective: reduce project risk by breaking it into smaller segments and providing more ease-of-change during the process • Three main variants <ol style="list-style-type: none"> 1. Series of mini-Waterfalls are performed, all Waterfalls are completed for portion of system before moving to next increment 2. Overall requirements are defined before evolving, mini-Waterfall development of individual increments of a system 3. Initial software concept, requirements analysis, design of architecture and system core are defined via Waterfall, followed by incremental implementation, which culminates in installing final, working version 	<ul style="list-style-type: none"> • Three benefits compared to Waterfall <ol style="list-style-type: none"> 1. Cost of changing customer requirements reduced. Redoing documentation & analysis less than waterfall method 2. Easier to get customer feedback on dev work; customers find it hard to judge progress from just documents 3. More rapid delivery & deployment of useful software to the customer possible, even if all it does not work; Customers able to use & gain value from software earlier than possible from waterfall 	<ul style="list-style-type: none"> • Primary cause of difficulty: large organization wrapped up in bureaucratic red tap that evolved over time; may be a mismatch between procedure & more informal iterative or agile process • Agile: aims at creating a "potentially shippable product" 	
Reuse-oriented software engineering	<ul style="list-style-type: none"> • Good chunk of projects have some software reuse • Look for these, modify them as needed and corporate them into their system • Stages <ol style="list-style-type: none"> 1. Component analysis: Given requirements, components are searched for; usually no exact match so provide only partial functionality 2. Requirements modification: Components are analyzed, then modified to reflect available components; Where modifications are impossible analysis activity may be re started to search for alternative solutions 3. System design with reuse: Framework of system designed/existing framework is reused; components that are reused and organize the framework are catered; some new software may have be designed if reusable components are not available 4. Development & integration: Software that cannot be externally procured is developed, and components and COTS systems are intergrated to create the new system 			
The V-Model	<ul style="list-style-type: none"> • Means to be a testing improvement of waterfall model • Every stage, test plans and cases are created to verify product at current stage • Verification and Validation go in parallel 	<ul style="list-style-type: none"> • Simple & Easy to use • Testing activities (planning, test designing) happen before coding; saves time 	<ul style="list-style-type: none"> • Rigid and least flexible • Software is developed in implementation phase; no early prototypes are produced 	<ul style="list-style-type: none"> • Small - Medium sized projects where requirements are clearly defined and fixed • When ample technical resources are



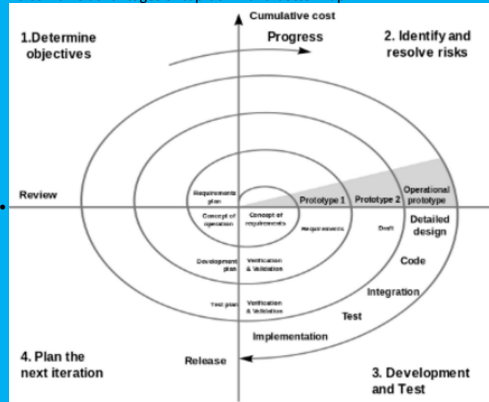
- Proactive defect tracking; defects found early
- Avoids downward flow of defects
- Works well for small projects where requirements are easily understood

- Any changes midway, test documents along with requirement documents has to be updated

- available with needed technical expertise
- High confidence of customer is needed for choosing this model
 - No prototypes are produced; thus high risk in meeting customers expectations

SPIRAL

- Combines some key aspects of waterfall model and rapid prototyping
 - To combine advantages of top-down and bottom-up



- Radial represents cumulative cost incurred in finishing the steps to day; angular represents the progress made in completing each cycle of the spiral
- Focus on risk assessment & minimize project risk by breaking project down
 - Evaluate risks & weigh whether project continuation throughout life cycle
- Each cycle involves progression same sequence of steps for each part
- Each trip around spiral traverses four quadrants
 1. Determine objective, alternatives & constraints of iteration
 2. Evaluate alternatives, identify and resolve risks
 3. Develop & verify deliverables from iteration
 4. Plan next iteration
- Begin each cycle with an identification of stakeholders & "win conditions" and end each cycle with review and commitment

- Large amount of risk analysis
- Good for large or mission-critical projects
- Strong approval and documentation control
- Additional functionality can be added at a later date
- Additional functionality can be added at a later date
- Software is produced early in the life cycle

- Can be a costly model to use
- Risk analysis requires highly specific expertise
- Project's success highly dependent on risk analysis phase
- Doesn't work well for smaller projects

- When costs & risk evaluation is key
- Medium to high risk projects
- Long term 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)

	<table><tr><th></th><th>Risk Item</th><th>Risk Management Techniques</th></tr><tr><td>1.</td><td>Personnel shortfalls</td><td>Staffing with top talent; job matching; teambuilding; morale building; cross-training; pre-scheduling key people</td></tr><tr><td>2.</td><td>Unrealistic schedules and budgets</td><td>Detailed, multisource cost and schedule estimation; design to cost; incremental development; software reuse; requirements scrubbing</td></tr><tr><td>3.</td><td>Developing the wrong software functions</td><td>Organization analysis; mission analysis; ops-concept formulation; user surveys; prototyping; early users' manuals</td></tr><tr><td>4.</td><td>Developing the wrong user interface</td><td>Task analysis; prototyping; scenarios; user characterization (functionality, style, workload)</td></tr><tr><td>5.</td><td>Gold plating</td><td>Requirements scrubbing; prototyping; cost-benefit analysis; design to cost</td></tr><tr><td>6.</td><td>Continuing stream of requirement changes</td><td>High change threshold; information hiding; incremental development (defer changes to later increments)</td></tr><tr><td>7.</td><td>Shortfalls in externally furnished components</td><td>Benchmarking; inspections; reference checking; compatibility analysis</td></tr><tr><td>8.</td><td>Shortfalls in externally performed tasks</td><td>Reference checking; pre-award audits; award-fee contracts; competitive design or prototyping; teambuilding</td></tr><tr><td>9.</td><td>Real-time performance shortfalls</td><td>Simulation; benchmarking; modelling; prototyping; instrumentation; tuning</td></tr><tr><td>10.</td><td>Straining computer-science capabilities</td><td>Technical analysis; cost—benefit analysis; prototyping; reference checking</td></tr></table>		Risk Item	Risk Management Techniques	1.	Personnel shortfalls	Staffing with top talent; job matching; teambuilding; morale building; cross-training; pre-scheduling key people	2.	Unrealistic schedules and budgets	Detailed, multisource cost and schedule estimation; design to cost; incremental development; software reuse; requirements scrubbing	3.	Developing the wrong software functions	Organization analysis; mission analysis; ops-concept formulation; user surveys; prototyping; early users' manuals	4.	Developing the wrong user interface	Task analysis; prototyping; scenarios; user characterization (functionality, style, workload)	5.	Gold plating	Requirements scrubbing; prototyping; cost-benefit analysis; design to cost	6.	Continuing stream of requirement changes	High change threshold; information hiding; incremental development (defer changes to later increments)	7.	Shortfalls in externally furnished components	Benchmarking; inspections; reference checking; compatibility analysis	8.	Shortfalls in externally performed tasks	Reference checking; pre-award audits; award-fee contracts; competitive design or prototyping; teambuilding	9.	Real-time performance shortfalls	Simulation; benchmarking; modelling; prototyping; instrumentation; tuning	10.	Straining computer-science capabilities	Technical analysis; cost—benefit analysis; prototyping; reference checking				
	Risk Item	Risk Management Techniques																																				
1.	Personnel shortfalls	Staffing with top talent; job matching; teambuilding; morale building; cross-training; pre-scheduling key people																																				
2.	Unrealistic schedules and budgets	Detailed, multisource cost and schedule estimation; design to cost; incremental development; software reuse; requirements scrubbing																																				
3.	Developing the wrong software functions	Organization analysis; mission analysis; ops-concept formulation; user surveys; prototyping; early users' manuals																																				
4.	Developing the wrong user interface	Task analysis; prototyping; scenarios; user characterization (functionality, style, workload)																																				
5.	Gold plating	Requirements scrubbing; prototyping; cost-benefit analysis; design to cost																																				
6.	Continuing stream of requirement changes	High change threshold; information hiding; incremental development (defer changes to later increments)																																				
7.	Shortfalls in externally furnished components	Benchmarking; inspections; reference checking; compatibility analysis																																				
8.	Shortfalls in externally performed tasks	Reference checking; pre-award audits; award-fee contracts; competitive design or prototyping; teambuilding																																				
9.	Real-time performance shortfalls	Simulation; benchmarking; modelling; prototyping; instrumentation; tuning																																				
10.	Straining computer-science capabilities	Technical analysis; cost—benefit analysis; prototyping; reference checking																																				
RAD (Rapid Application Development)	<ul style="list-style-type: none">• Favors iterative development & rapid construction of prototypes<ul style="list-style-type: none">• As opposed to construction of prototypes• Fast development & delivery of a high quality system at a relatively low invest cost• Attempts reduction in project risk by breaking into smaller segments & gives ease-of-change during development• Produce systems quickly through iterative prototyping, active user involvement, and computerized development tools• Emphasis on fulfilling business need; tech or engineering excellence is lesser importance• Project control involves focusing on dev and defining delivery deadlines or timeboxes<ul style="list-style-type: none">• If project starts to slip, requirements are reduced to fit timebox; not pushing deadline back• Includes Joint Application Design<ul style="list-style-type: none">• Where users are intensely involved in system design via consensus building in either workshops or electronically facilitated interaction• Active user involvement is imperative• Iteratively produces production software, as opposed to throwaway prototype• Produces documentation necessary to facilitate future development and maintenance• Standard systems analysis and design methods can be fitting into this framework	<ul style="list-style-type: none">• Reduced development time• Increases reusability of components• Quick initial reviews occur• Encourages customer feedback• Integration from very beginning solves a lot of integration issues	<ul style="list-style-type: none">• Depends on strong team & individual performances for identifying business requirements• Only system that can be modularized can be built using RAD• Requires highly skilled developers/designers• High dependency on modeling skills• Inapplicable to cheaper projects as cost of model and automated code generation is very high	<ul style="list-style-type: none">• When you need to make a modular system in 2-3months time• Should be used if high availability of designers for modeling• Budget high enough to afford their cost along with cost of automated code generating tools• Its SDLC should be chosen only if resources with high business knowledge are available and there is a need to produce the system in a short span of time																																		
AGILE	<ul style="list-style-type: none">• Uses iterative development as a basis<ul style="list-style-type: none">• But advocates a lighter and more people-centric viewpoint• Incorporate iteration and continuous feedback that it provides to refine a software system <div></div>	<ul style="list-style-type: none">• Rapid delivery of useful software brings customer satisfaction• People & interactions are emphasized instead of process and tools<ul style="list-style-type: none">• Customers, devs and testers constantly interact with eachother• Working software is delivered frequently• Face-to-face conversation best way to communicate• Close cooperation with business and dev• Continuous attention to technical excellence and good design• Regular adaptation to changing circumstances• Late changes in requirements are welcomed	<ul style="list-style-type: none">• Some deliverables (large ones) are difficult to asses the effort required at the start of the SDLC• Lack of emphasis on necessary desinging and documentation• Project can easily get off track id custom rep is not clear what final outcome they want• Only senior programmers are able at taking kind of decisions during dev process<ul style="list-style-type: none">• Hence no plan for newbie programmers, unless combined with exprienced resources	<ul style="list-style-type: none">• New changes are need to be implemented<ul style="list-style-type: none">• Cost little due to frequent new increments that are produced• Implement a new feature the dev need to lose only the work of a few days, or hours to roll back and implement• Unlike waterfall, limited planning is required<ul style="list-style-type: none">• Assumes end users' needs are ever changing• Both devs and stakeholder alike find they get more freedom of time and options than if software was developed in a more rigid sequential way																																		