SELECTING A DEVELOPMENT APPROACH

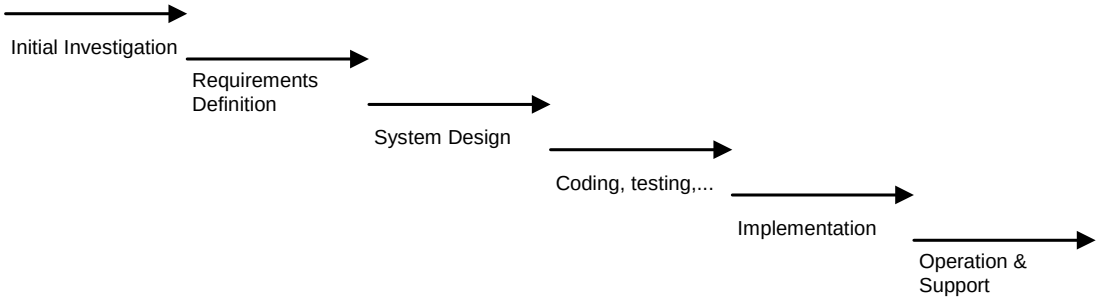
Original Issuance: February 17, 2005 *Revalidated: March 27, 2008*

Introduction

A system development methodology refers to the framework that is used to structure, plan, and control the process of developing an information system. A wide variety of such frameworks have evolved over the years, each with its own recognized strengths and weaknesses. One system development methodology is not necessarily suitable for use by all projects. Each of the available methodologies is best suited to specific kinds of projects, based on various technical, organizational, project and team considerations. CMS has considered each of the major prescribed methodologies in context with CMS' business, applications, organization, and technical environments. As a result, CMS requires the use of any of the following linear and iterative methodologies for CMS systems development, as appropriate.

Acceptable System Development Methodologies

Waterfall



Framework Type: Linear

Basic Principles:

1. Project is divided into sequential phases, with some overlap and splashback acceptable between phases.
2. Emphasis is on planning, time schedules, target dates, budgets and implementation of an entire system at one time.
3. Tight control is maintained over the life of the project through the use of extensive written documentation, as well as through formal reviews and approval/signoff by the

user and information technology management occurring at the end of most phases before beginning the next phase.

Strengths:

1. Ideal for supporting less experienced project teams and project managers, or project teams whose composition fluctuates.
2. The orderly sequence of development steps and strict controls for ensuring the adequacy of documentation and design reviews helps ensure the quality, reliability, and maintainability of the developed software.
3. Progress of system development is measurable.
4. Conserves resources.

Weaknesses:

1. Inflexible, slow, costly and cumbersome due to significant structure and tight controls.
2. Project progresses forward, with only slight movement backward.
3. Little room for use of iteration, which can reduce manageability if used.
4. Depends upon early identification and specification of requirements, yet users may not be able to clearly define what they need early in the project.
5. Requirements inconsistencies, missing system components, and unexpected development needs are often discovered during design and coding.
6. Problems are often not discovered until system testing.
7. System performance cannot be tested until the system is almost fully coded, and under-capacity may be difficult to correct.
8. Difficult to respond to changes. Changes that occur later in the life cycle are more costly and are thus discouraged.
9. Produces excessive documentation and keeping it updated as the project progresses is time-consuming.
10. Written specifications are often difficult for users to read and thoroughly appreciate.
11. Promotes the gap between users and developers with clear division of responsibility.

Situations where <u>most</u> appropriate:

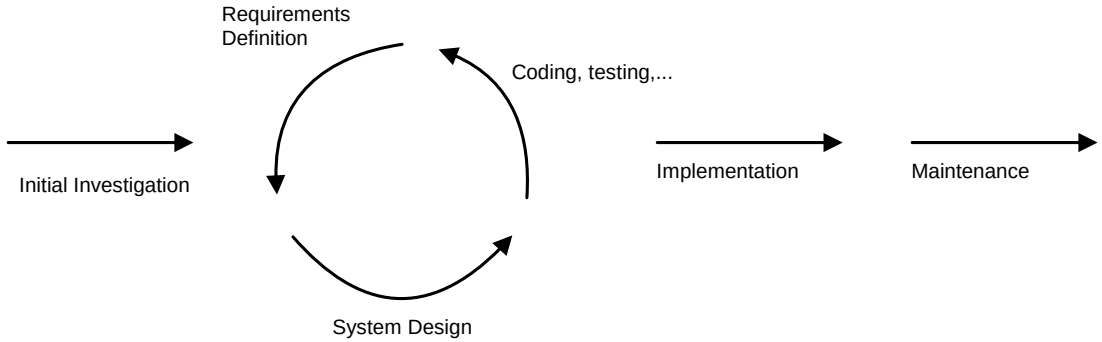
1. Project is for development of a mainframe-based or transaction-oriented batch system.
2. Project is large, expensive, and complicated.
3. Project has clear objectives and solution.
4. Pressure does not exist for immediate implementation.
5. Project requirements can be stated unambiguously and comprehensively.
6. Project requirements are stable or unchanging during the system development life cycle.
7. User community is fully knowledgeable in the business and application.
8. Team members may be inexperienced.
9. Team composition is unstable and expected to fluctuate.
10. Project manager may not be fully experienced.
11. Resources need to be conserved.
12. Strict requirement exists for formal approvals at designated milestones.

Situations where <u>least</u> appropriate:

1. Large projects where the requirements are not well understood or are changing for any reasons such as external changes, changing expectations, budget changes or rapidly changing technology.

2. Web Information Systems (WIS) primarily due to the pressure of implementing a WIS project quickly; the continual evolution of the project requirements; the need for experienced, flexible team members drawn from multiple disciplines; and the inability to make assumptions regarding the users' knowledge level.
3. Real-time systems.
4. Event-driven systems.
5. Leading-edge applications.

Prototyping



Framework Type: Iterative

Basic Principles:

1. Not a standalone, complete development methodology, but rather an approach to handling selected portions of a larger, more traditional development methodology (i.e., Incremental, Spiral, or Rapid Application Development (RAD)).
2. Attempts to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.
3. User is involved throughout the process, which increases the likelihood of user acceptance of the final implementation.
4. Small-scale mock-ups of the system are developed following an iterative modification process until the prototype evolves to meet the users' requirements.
5. While most prototypes are developed with the expectation that they will be discarded, it is possible in some cases to evolve from prototype to working system.
6. A basic understanding of the fundamental business problem is necessary to avoid solving the wrong problem.

Strengths:

1. "Addresses the inability of many users to specify their information needs, and the difficulty of systems analysts to understand the user's environment, by providing the user with a tentative system for experimental purposes at the earliest possible time." (Janson and Smith, 1985)
2. "Can be used to realistically model important aspects of a system during each phase of the traditional life cycle." (Huffaker, 1986)
3. Improves both user participation in system development and communication among project stakeholders.

4. Especially useful for resolving unclear objectives; developing and validating user requirements; experimenting with or comparing various design solutions; or investigating both performance and the human computer interface.
5. Potential exists for exploiting knowledge gained in an early iteration as later iterations are developed.
6. Helps to easily identify confusing or difficult functions and missing functionality.
7. May generate specifications for a production application.
8. Encourages innovation and flexible designs.
9. Provides quick implementation of an incomplete, but functional, application.

Weaknesses:

1. Approval process and control is not strict.
2. Incomplete or inadequate problem analysis may occur whereby only the most obvious and superficial needs will be addressed, resulting in current inefficient practices being easily built into the new system.
3. Requirements may frequently change significantly.
4. Identification of non-functional elements is difficult to document.
5. Designers may prototype too quickly, without sufficient up-front user needs analysis, resulting in an inflexible design with narrow focus that limits future system potential.
6. Designers may neglect documentation, resulting in insufficient justification for the final product and inadequate records for the future.
7. Can lead to poorly designed systems. Unskilled designers may substitute prototyping for sound design, which can lead to a "quick and dirty system" without global consideration of the integration of all other components. While initial software development is often built to be a "throwaway", attempting to retroactively produce a solid system design can sometimes be problematic.
8. Can lead to false expectations, where the customer mistakenly believes that the system is "finished" when in fact it is not; the system looks good and has adequate user interfaces, but is not truly functional.
9. Iterations add to project budgets and schedules, thus the added costs must be weighed against the potential benefits. Very small projects may not be able to justify the added time and money, while only the high-risk portions of very large, complex projects may gain benefit from prototyping.
10. Prototype may not have sufficient checks and balances incorporated.

Situations where <u>most</u> appropriate:

1. Project is for development of an online system requiring extensive user dialog, or for a less well-defined expert and decision support system.
2. Project is large with many users, interrelationships, and functions, where project risk relating to requirements definition needs to be reduced.
3. Project objectives are unclear.
4. Pressure exists for immediate implementation of something.
5. Functional requirements may change frequently and significantly.
6. User is not fully knowledgeable.
7. Team members are experienced (particularly if the prototype is not a throw-away).
8. Team composition is stable.
9. Project manager is experienced.
10. No need exists to absolutely minimize resource consumption.

11. No strict requirement exists for approvals at designated milestones.
12. Analysts/users appreciate the business problems involved, before they begin the project.
13. Innovative, flexible designs that will accommodate future changes are not critical.

Situations where <u>least</u> appropriate:

1. Mainframe-based or transaction-oriented batch systems.
2. Web-enabled e-business systems.
3. Project team composition is unstable.
4. Future scalability of design is critical.
5. Project objectives are very clear; project risk regarding requirements definition is low.

Incremental

Framework Type: Combination Linear and Iterative

Basic Principles:

Various methods are acceptable for combining linear and iterative system development methodologies, with the primary objective of each being to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process:

1. A series of mini-Waterfalls are performed, where all phases of the Waterfall development model are completed for a small part of the system, before proceeding to the next increment; OR
2. Overall requirements are defined before proceeding to evolutionary, mini-Waterfall development of individual increments of the system, OR
3. The initial software concept, requirements analysis, and design of architecture and system core are defined using the Waterfall approach, followed by iterative Prototyping, which culminates in installation of the final prototype (i.e., working system).

Strengths:

1. Potential exists for exploiting knowledge gained in an early increment as later increments are developed.
2. Moderate control is maintained over the life of the project through the use of written documentation and the formal review and approval/signoff by the user and information technology management at designated major milestones.
3. Stakeholders can be given concrete evidence of project status throughout the life cycle.
4. Helps to mitigate integration and architectural risks earlier in the project.
5. Allows delivery of a series of implementations that are gradually more complete and can go into production more quickly as incremental releases.
6. Gradual implementation provides the ability to monitor the effect of incremental changes, isolate issues and make adjustments before the organization is negatively impacted.

Weaknesses:

1. When utilizing a series of mini-Waterfalls for a small part of the system before moving on to the next increment, there is usually a lack of overall consideration of the business problem and technical requirements for the overall system.

2. Since some modules will be completed much earlier than others, well-defined interfaces are required.
3. Difficult problems tend to be pushed to the future to demonstrate early success to management.

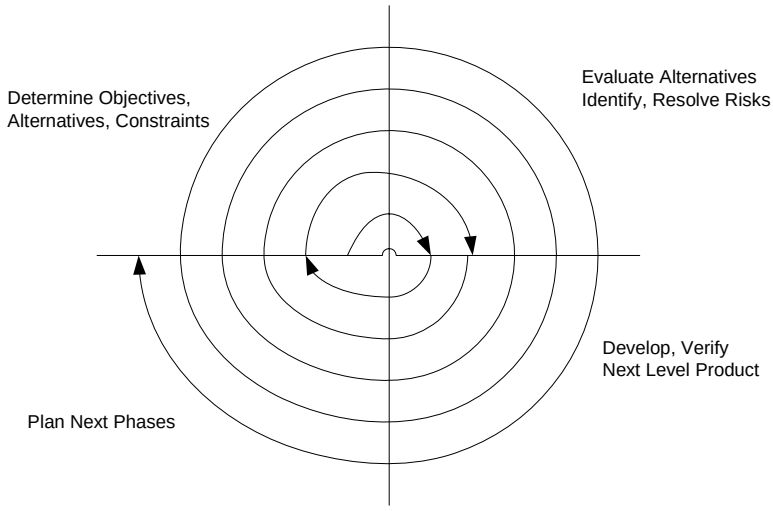
Situations where <u>most</u> appropriate:

1. Large projects where requirements are not well understood or are changing due to external changes, changing expectations, budget changes or rapidly changing technology.
2. Web Information Systems (WIS) and event-driven systems.
3. Leading-edge applications.

Situations where <u>least</u> appropriate:

1. Very small projects of very short duration.
2. Integration and architectural risks are very low.
3. Highly interactive applications where the data for the project already exists (completely or in part), and the project largely comprises analysis or reporting of the data.

Spiral



Framework Type: Combination Linear and Iterative

Basic Principles:

1. Focus is on risk assessment and on minimizing project risk by breaking a project into smaller segments and providing more ease-of-change during the development process, as well as providing the opportunity to evaluate risks and weigh consideration of project continuation throughout the life cycle.
2. "Each cycle involves a progression through the same sequence of steps, for each portion of the product and for each of its levels of elaboration, from an overall concept-of-operation document down to the coding of each individual program." (Boehm, 1986)
3. Each trip around the spiral traverses four basic quadrants: (1) determine objectives, alternatives, and constraints of the iteration; (2) evaluate alternatives; identify and resolve

risks; (3) develop and verify deliverables from the iteration; and (4) plan the next iteration. (Boehm, 1986 and 1988)
4. Begin each cycle with an identification of stakeholders and their win conditions, and end each cycle with review and commitment. (Boehm, 2000)

Strengths:

1. Enhances risk avoidance.
2. Useful in helping to select the best methodology to follow for development of a given software iteration, based on project risk.
3. Can incorporate Waterfall, Prototyping, and Incremental methodologies as special cases in the framework, and provide guidance as to which combination of these models best fits a given software iteration, based upon the type of project risk. For example, a project with low risk of not meeting user requirements, but high risk of missing budget or schedule targets would essentially follow a linear Waterfall approach for a given software iteration. Conversely, if the risk factors were reversed, the Spiral methodology could yield an iterative Prototyping approach.

Weaknesses:

1. Challenging to determine the exact composition of development methodologies to use for each iteration around the Spiral.
2. Highly customized to each project, and thus is quite complex, limiting reusability.
3. A skilled and experienced project manager is required to determine how to apply it to any given project.
4. There are no established controls for moving from one cycle to another cycle. Without controls, each cycle may generate more work for the next cycle.
5. There are no firm deadlines. Cycles continue with no clear termination condition, so there is an inherent risk of not meeting budget or schedule.
6. Possibility exists that project ends up implemented following a Waterfall framework.

Situations where <u>most</u> appropriate:

1. Real-time or safety-critical systems.
2. Risk avoidance is a high priority.
3. Minimizing resource consumption is not an absolute priority.
4. Project manager is highly skilled and experienced.
5. Requirement exists for strong approval and documentation control.
6. Project might benefit from a mix of other development methodologies.
7. A high degree of accuracy is essential.
8. Implementation has priority over functionality, which can be added in later versions.

Situations where <u>least</u> appropriate:

1. Risk avoidance is a low priority.
2. A high degree of accuracy is not essential.
3. Functionality has priority over implementation.
4. Minimizing resource consumption is an absolute priority.

Rapid Application Development (RAD)

Framework Type: Iterative

Basic Principles:

1. Key objective is for fast development and delivery of a high quality system at a relatively low investment cost.
2. Attempts to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.
3. Aims to produce high quality systems quickly, primarily through the use of iterative Prototyping (at any stage of development), active user involvement, and computerized development tools. These tools may include Graphical User Interface (GUI) builders, Computer Aided Software Engineering (CASE) tools, Database Management Systems (DBMS), fourth-generation programming languages, code generators, and object-oriented techniques.
4. Key emphasis is on fulfilling the business need, while technological or engineering excellence is of lesser importance.
5. Project control involves prioritizing development and defining delivery deadlines or "timeboxes". If the project starts to slip, emphasis is on reducing requirements to fit the timebox, not in increasing the deadline.
6. Generally includes Joint Application Development (JAD), where users are intensely involved in system design, either through consensus building in structured workshops, or through electronically facilitated interaction.
7. Active user involvement is imperative.
8. Iteratively produces production software, as opposed to a throwaway prototype.
9. Produces documentation necessary to facilitate future development and maintenance.
10. Standard systems analysis and design techniques can be fitted into this framework.

Strengths:

1. The operational version of an application is available much earlier than with Waterfall, Incremental, or Spiral frameworks.
2. Because RAD produces systems more quickly and to a business focus, this approach tends to produce systems at a lower cost.
3. Engenders a greater level of commitment from stakeholders, both business and technical, than Waterfall, Incremental, or Spiral frameworks. Users are seen as gaining more of a sense of ownership of a system, while developers are seen as gaining more satisfaction from producing successful systems quickly.
4. Concentrates on essential system elements from user viewpoint.
5. Provides the ability to rapidly change system design as demanded by users.
6. Produces a tighter fit between user requirements and system specifications.
7. Generally produces a dramatic savings in time, money, and human effort.

Weaknesses:

1. More speed and lower cost may lead to lower overall system quality.
2. Danger of misalignment of developed system with the business due to missing information.
3. Project may end up with more requirements than needed (gold-plating).

4. Potential for feature creep where more and more features are added to the system over the course of development.
5. Potential for inconsistent designs within and across systems.
6. Potential for violation of programming standards related to inconsistent naming conventions and inconsistent documentation.
7. Difficulty with module reuse for future systems.
8. Potential for designed system to lack scalability.
9. Potential for lack of attention to later system administration needs built into system.
10. High cost of commitment on the part of key user personnel.
11. Formal reviews and audits are more difficult to implement than for a complete system.
12. Tendency for difficult problems to be pushed to the future to demonstrate early success to management.
13. Since some modules will be completed much earlier than others, well-defined interfaces are required.

Situations where <u>most</u> appropriate:

1. Project is of small-to-medium scale and of short duration (no more than 6 man-years of development effort).
2. Project scope is focused, such that the business objectives are well defined and narrow.
3. Application is highly interactive, has a clearly defined user group, and is not computationally complex.
4. Functionality of the system is clearly visible at the user interface.
5. Users possess detailed knowledge of the application area.
6. Senior management commitment exists to ensure end-user involvement.
7. Requirements of the system are unknown or uncertain.
8. It is not possible to define requirements accurately ahead of time because the situation is new or the system being employed is highly innovative.
9. Team members are skilled both socially and in terms of business.
10. Team composition is stable; continuity of core development team can be maintained.
11. Effective project control is definitely available.
12. Developers are skilled in the use of advanced tools.
13. Data for the project already exists (completely or in part), and the project largely comprises analysis or reporting of the data.
14. Technical architecture is clearly defined.
15. Key technical components are in place and tested.
16. Technical requirements (e.g., response times, throughput, database sizes, etc.) are reasonable and well within the capabilities of the technology being used. Targeted performance should be less than 70% of the published limits of the technology.
17. Development team is empowered to make design decisions on a day-to-day basis without the need for consultation with their superiors, and decisions can be made by a small number of people who are available and preferably co-located.

Situations where <u>least</u> appropriate:

1. Very large, infrastructure projects; particularly large, distributed information systems such as corporate-wide databases.
2. Real-time or safety-critical systems.
3. Computationally complex systems, where complex and voluminous data must be analyzed, designed, and created within the scope of the project.

4. Project scope is broad and the business objectives are obscure.
5. Applications in which the functional requirements have to be fully specified before any programs are written.
6. Many people must be involved in the decisions on the project, and the decision makers are not available on a timely basis or they are geographically dispersed.
7. The project team is large or there are multiple teams whose work needs to be coordinated.
8. When user resource and/or commitment is lacking.
9. There is no project champion at the required level to make things happen.
10. Many new technologies are to be introduced within the scope of the project, or the technical architecture is unclear and much of the technology will be used for the first time within the project.
11. Technical requirements (e.g., response times, throughput, database sizes, etc.) are tight for the equipment that is to be used.

References:

"System Development Methodologies for Web Enabled E-Business: A Customization Paradigm"; Linda Night, Theresa Steinbach, and Vince Kellen; November 2001; (http://www.kellen.net/SysDev.htm)

"A Survey of System Development Process Models"; Darryl Green and Ann DiCaterino; Center for Technology in Government; February 1998; (http://www.ctg.albany.edu/publications/reports/survey_of_sysdev)

"System Development Life Cycle Models and Methodologies"; Paul Fisher, James McDaniel, and Peter Hughes; Canadian Society for International Health Certificate Course in Health Information Systems, Module 3: System Analysis & Database Development, Part 3: Life Cycle Models and Methodologies; (http://famed.ufrgs.br/pdf/csih/mod3/Mod_3_3.htm)

"Rapid Application Development: A Review and Case Study"; Paul Beynon-Davies; Kane Thompson Centre; December 1998; (http://www.comp.glam.ac.uk/SOC_Server/research/gisc/RADbrf1.htm)

"Introduction to Systems Analysis, Topic 19, Rapid Application Development"; J. R. McBride; Copyright 2002 Prentice-Hall, Inc.;(http://www.csc.uvic.ca/~jmcbride/c375t19.pdf)