Critical Factors Characterizing Projects and Lifecycle Models

Kalman C Toth

kalmanctoth@gmail.com

Herman Migliore

herm@cecs.pdx.edu

Abstract

Systems and software engineering lifecycle process methodologists boost their favorite approaches, but rarely discuss which problems these methods are best suited to solve or when they might indeed represent an ill-advised choice. From more traditional waterfall lifecycles to those popularized by the agile development community, proponents describe their benefits, often with built-in bias. Some methodologies claim that they can be customized to fit large, highly critical applications as well as very simple ones. Rarely do the claims present a balanced perspective about how the method addresses the needs of a given project, or how it can be adapted to fit the problem at hand.

The overriding aim of the work that led to this paper was to explore how critical factors characterizing projects might be applied by engineers to assist in the selection of an appropriate lifecycle process for the project. This paper summarizes the common lifecycle processes advocated by systems and software engineers, identifies eight critical selection factors for characterizing lifecycles and projects, and suggests an approach for systematically matching projects to lifecycle processes for the purpose of selecting the most suitable lifecycle for a given project. The authors have also illustrated the first step of such an approach by characterizing the most popular lifecycle models in terms of these eight selection factors.

Biography

Kal Toth has over 30 years of technical, consulting and management experience working for small, medium and large-sized technology companies including aTrust Inc., Hughes Aircraft, Datalink Systems Corp., the CGI Group Inc., the Software Productivity Centre (Vancouver, BC), and Intellitech Canada. Past Executive Director of the Oregon Master of Software Engineering (OMSE) program at PSU and for 12 years delivered systems/software engineering and project management courses at PSU, OSU, TechBC, Simon Fraser, U of Alberta, and UBC, he completed his Ph.D. in systems engineering at Carleton University (Canada), and is a professional engineer with a software engineering designation in B.C.

Herman Migliore has forty years' experience in engineering design, application of computational mechanics (part of his graduate studies and research activities), and development of design methodologies (part of his role in design education). Since 1997, he has been Director of Systems Engineering at Portland State University - an online masters program intended for experienced, practicing engineers. In his capacity as director, he has participated in many projects that apply systems engineering to a wide variety of areas for small, medium, and large industry and government sponsors.

1 Introduction

Software and systems engineers have systematically defined and progressively improved their processes, methods, and techniques over the years. However, many open issues for study remain. One particularly challenging area has to do with how to go about selecting a suitable lifecycle process (or processes) for the problem at hand [1] [2]. For example, when should one adopt a highly disciplined lifecycle, versus a much more agile process, or one that balances discipline and agility [3]?

When preparing to undertake a new project, the prudent engineering manager should ask:

- Which life-cycle processes are acceptable candidates? (Do competencies/resources fit?)
- Which process is most likely to yield positive project economies and competitive advantage?
- Will a given candidate be more or less vulnerable to project over-runs, failures, or lost business?
- Is it feasible to assess the properties of alternative lifecycles and make trade-offs among them?

More specifically, the engineering manager should look for objective evidence that a given lifecycle development model will indeed be the most suitable choice for the planned project.

The approach we have taken to tackle this problem has been a practical one – review what others' have said and done (including Boehm [3], Kruchten [8], Cockburn [9] and others) with respect to the application of popular system and software development models; observe the challenges they have encountered to enhance understanding as to what worked, what didn't, and when; and propose a strategy that could lead to practical methods and guidance for the purpose of lifecycle selection.

Our estimates are offered with some caution as they are experience-based and can only be interpreted as subjective. However, they have been derived from our collective 50-years of systems and software engineering experience including the transition of a $1B air traffic control project [10] from a waterfall lifecycle to an iterative/incremental lifecycle using Rational methods and tools, as well as numerous smaller e-commerce development projects that adopted incremental and evolutionary process models to accomplish their objectives.

Grounded on experience and subjectivity, we nevertheless expect that quantitative and empirically driven methods could emerge from additional study. Readers are encouraged to rebut the ideas we have put forward, thereby contributing to this effort to discover systematic methods for characterizing projects and selecting suitable lifecycle development models for them.

An important factor that we have not factored into our proposed approach is the enterprise's existing process culture. Certainly, the overriding factor for many smaller companies will be what they know, what they have, and what they are good at. However, it is safe to say that an increasing number of medium-sized to large enterprises, especially system-integration firms, possess the skills and know-how across many if not all of the lifecycles discussed in this paper. And nimble practitioners and learning organizations are exploring their process options going forward – progressively improving and adapting their existing processes to keep pace and beat out the competition. This paper may be of particular interest to them.

2 Candidate Lifecycle Models and Selection Factors

Our review of systems and software engineering lifecycles considered the waterfall, iterative, incremental, spiral, evolutionary and agile lifecycle development models. Collectively they cover virtually the entire range of lifecycle models in practice today if one excludes ad hoc development methods. Therefore we have constrained the scope of our discussion of lifecycle selection factors to these models.

In the context of these models we reviewed the attributes of each. After some reflection we subjectively derived our preferred (top-eight) critical factors for characterizing projects and selecting a suitable lifecycle for a given project, namely:

- Quality/Maintainability
- Application Domain
- Size / Complexity
- Requirements Uncertainty
- Progress Visibility
- User Involvement
- Requirements Volatility
- Urgency

The following paragraphs define, in practical terms, the context, meaning and implications of these selection factors.

Quality/Maintainability: Projects constructing complex systems often demand high levels of delivered quality and built-in maintainability as they are highly driven by the need to meet stringent criticality thresholds and contain total lifecycle costs. For example, customers in the aerospace and military sectors will write explicit quality and maintainability targets, standards, and documentation into their requests for proposals and acceptance criteria which must be met by the contractor. At the other end of the spectrum, solutions for experimental and concept validation purposes care little about operational performance and down-stream maintainability effects. Of course, a majority of technology projects lay on a grey-scale between these extremes.

Application Domain: The category of application is also a very significant project driver. High-end applications are those characterized by critical performance, reliability, availability, security, and safety requirements; moderate applications include business and enterprise applications; low criticality applications include proof-of-concept demonstrations, experimental prototypes, and vanilla-flavored web-sites.

Size / Complexity: For simplicity, we consider these two factors together as they are often, but not always, correlated fairly linearly. Certainly, an air traffic control system would be a highly complex and consists of a very large number of hardware components and lines of software code. Relatively speaking, a real-time robotics system would be proportionately smaller and less complex. One could assume, for example, that a planned 500K lines-of-code (LOC) software system would be considered both large and highly complex; a 10K LOC subsystem would be considered small and relatively simple; and medium-sized projects would range in-between.

Requirements Uncertainty: Many projects will commit to resources and meeting the needs of a customer before they properly comprehend the functions and features to be developed. Let us take the position that requirements are highly uncertain if they are only verbalized or if they have been expressed in a few pages with little input from users. In contrast, requirements should be relatively certain when considerable resources and schedule have been devoted to studying user needs and developing evaluation prototypes.

Progress Visibility: Visibility into progress can be achieved through demonstrations and through documentation – both have their shortcomings. Demonstrations are favored by many customers but if not supported by other measures of progress can give the illusion of more progress than is actually being achieved. Documentation is harder to translate into real progress but simplifies contracting – hence it is favored by those with accounting and legal mindsets. For this discussion we assume that ongoing and frequent demonstrations to customers and users accompanied by some useful documentation can provide effective visibility into progress. In contrast, infrequent demonstrations of functionality combined with scant documentation provide little or no visibility into achieved progress.

User Involvement: User involvement can be with respect to developing requirements specifications, validating requirement specifications, inspecting prototypes, supporting detailed design and development, reviewing specifications, and accepting released products. Users heavily committed to supporting most of these development efforts are considered to be “highly involved”. Marginal contributions from users in a few of these areas should be considered to be “low” involvement.

Requirements Volatility: This factor requires a good understanding of the problem domain, especially with respect to the maturity of the customer, users, and the development team. If the application is unprecedented or if the problem is in an emerging area, stakeholders will not be sure of their priorities and what can be feasibly accomplished. Project management and stakeholder vacillation can exacerbate requirements volatility. Often, but not always, requirements uncertainty will go hand-in-hand with requirements volatility.

Urgency / Time-to-Market: Demanding market forces will put pressure on many commercial software projects to create and release software functionalities and features as early as possible – often pushing the development to take shortcuts rather that down-scoping the project. Schedule pressure will tend to be moderated in mission-critical projects because they tend to adopt disciplined oversight and accountability measures.

3 Characterizing Lifecycle Development Models and Projects

We reasoned that it should be possible to characterize both lifecycle models and projects in terms of the above eight factors and then match a given project's characterization data to each lifecycle model's characterization data to locate and thereby select the best fit. Sensitivity analyses could be conducted by varying the project's characterization data; and trade-off studies could be conducted by estimating project costs and schedules under a given lifecycle.

We began by first exploring the challenge of characterizing lifecycle development models. Such characterization would involve assigning appropriate values ("estimates") or ranges for each lifecycle selection factor. These assigned values could be tabulated and presented graphically to help visualize and compare each characterization. We postulated that it would be feasible to develop an empirical characterization model for each lifecycle by compiling and analyzing data from a large enough number of projects. To be clear, our goal was to explore possibilities – not compile reams of project data.

Next we began to reflect on the types of methods and guidance that would need to be developed for estimating appropriate characterization data for each project. Presumably, these methods would be related to the above-mentioned empirical characterization data that would be complied for lifecycles. A significant additional challenge would be to develop an effective matching technique that quantifies the degree of project-to-lifecycle fit. We anticipated that some of the methods used in software estimating [4] and COTS selection [5] could be adapted to accomplish this goal.

4 Postulated Lifecycle Selection Process

We postulate that a lifecycle model selection process would flow along these lines:

1. As a pre-condition, that baseline characterization data ("profile") for each lifecycle model would need to be available, likely in a parameterized form, derived from project data collection and analysis.
2. Next, the project objectives, context, assumptions, resources, constraints and priorities would be collected from project stakeholders (e.g. customers and users) and compiled.
3. The list of candidate lifecycles would then be pruned, if possible, to eliminate any obvious incompatibilities between the project attributes and lifecycle attributes.
4. Using the project data, each of the selection factors would be estimated by assigning nominal values to them, possibly by applying adjustment factors (e.g. "multipliers").
5. A matching algorithm comparing the project characterization data with each lifecycle model's characterization profile would be executed to estimate the degree of fit and select the best fit.
6. Sensitivity and trade-off analyses could be performed by varying the characterization data and conducting independent project cost and schedule estimates.

Clearly the development of such a lifecycle selection process would require considerable effort to vet and refine postulated process, compile and analyze project data, and develop suitable guidance and matching methods.

5 Characterization of Each Lifecycle Development Model

In the sections that follow, we have provided our rough-cut characterization data estimates for the principle lifecycles (see [3], [8] and [9]) to illustrate the first step of this approach. These estimates characterizing each lifecycle have been tabulated (see annex) and graphically presented for each lifecycle. The table in the annex contains the estimates and rationale used to subjectively estimate each factor for each lifecycle model. Observe that some of the estimates in the tables are expressed in very broad ranges (e.g. Medium-to-High).

6 Waterfall Development

Waterfall development is relatively sequential - characterized by development phases, major milestones, and specified deliverables reviewed by stakeholders (depicted in Figure 1 and characterized in Figure 2). Typically, looping back to the previous phase to correct problems is permitted (waterfall is almost never purely sequential). The Waterfall Model has been used in its most disciplined form in the aerospace, military, and related application domains. Formal change control procedures are normally mandated to correct problems in earlier phases. The Waterfall Model encourages thorough requirements development and design, and formalizes milestones, documentation and deliverables.

Waterfall is less adaptive than other models to project changes and market demands, and project visibility is primarily achieved through the delivered documentation. This process is typically used for safety and reliability critical systems where quality and lifecycle maintainability are high priorities, the systems tend to be large and complex, and the requirements are fairly well understood - at least in terms of general functionality and scope. Waterfall is applicable to large, mid-sized, and small projects alike.

Figure 1: Waterfall Lifecycle Model [11]

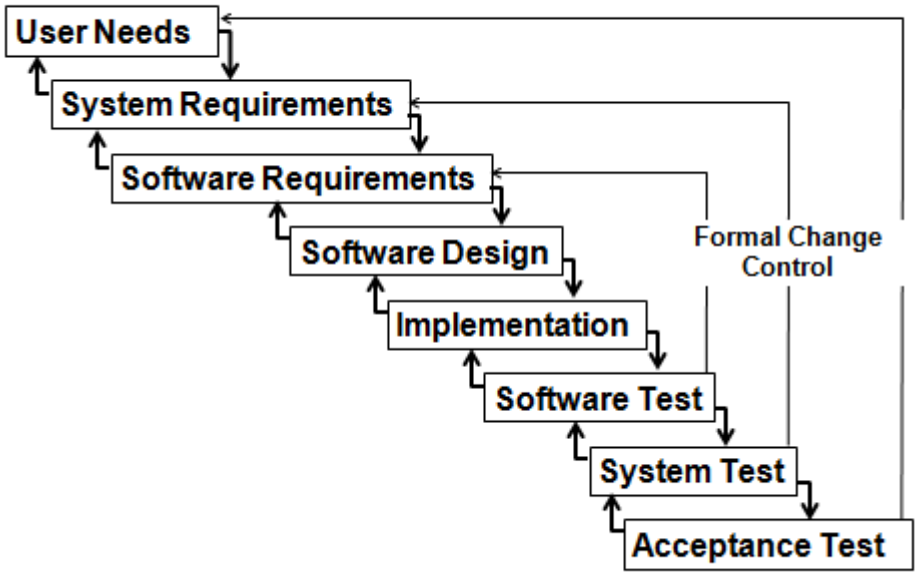
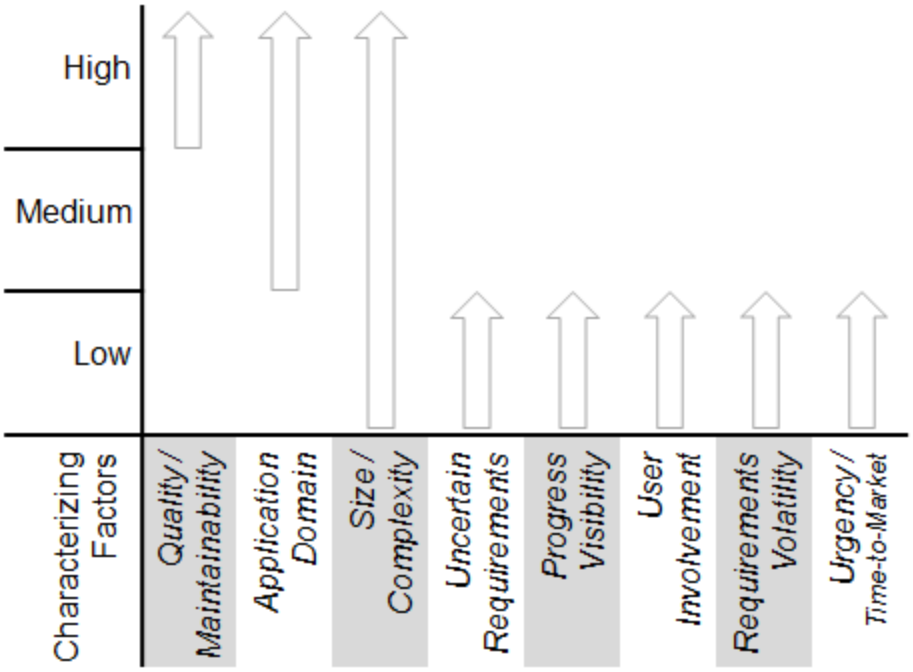


Figure 2: Waterfall Lifecycle Model Characterized



7 Iterative Development

Various iterative models have been developed over the last two decades. They include the incremental, spiral, evolutionary and agile development lifecycles described below. The waterfall model itself has been adapted to be somewhat more iterative than originally interpreted and practiced. The common characteristics of iterative models include: repeated cycles of development, ongoing rework, and parallel (concurrent) development. They generally enable better schedules, early discovery of problems, better customer and user feedback, and visibility into progress by way of demonstrations. However, they can be challenging to manage. The iterative model variants follow.

8 Incremental Development

The traditional view of the Incremental Model is that it is a planned iterative lifecycle that partitions large, complex problems into independent parts, concurrently develops these partitions, and progressively integrates the parts. This should be contrasted with Agile development which has adapted certain, but not all (e.g. up-front requirements and design), aspects of traditional incremental development and is described below. The traditional incremental model is depicted in Figure 3 and characterized in Figure 4.

The traditional Incremental Model first develops a thorough understanding of the requirements, then develops a stable system architecture, and then partitions the system into independent partitions or subsystems for development. Incremental lifecycles are well-suited to concurrent development and both partial and progressive delivery of system capability and have been extensively applied in aerospace and military domains to scale up the waterfall process for very large and highly critical projects. Mapping requirements to increments can be challenging and unanticipated changes to requirements and the architecture can break across increments and imply major rework later on.

Incremental delivery allows for a degree of requirements uncertainty since poorly understood requirements can be pushed off to later increments and product releases. The Incremental Model also

offers positive visibility into progress at each release. Observe, however, that management and technical processes must be able to modularize the project to be able to run such projects without losing control. This implies devoting early effort to baselining as much of the requirements as possible and stabilizing foundation components and the most critical architectural modules as early as possible. In comparison to the waterfall, an incremental lifecycle will increase both technical and management overhead, will yield longer schedules, and increase overall project costs. However, large and very larger projects can be tackled very effectively by way of the incremental lifecycle model.

Figure 3: Incremental Lifecycle Model [11]

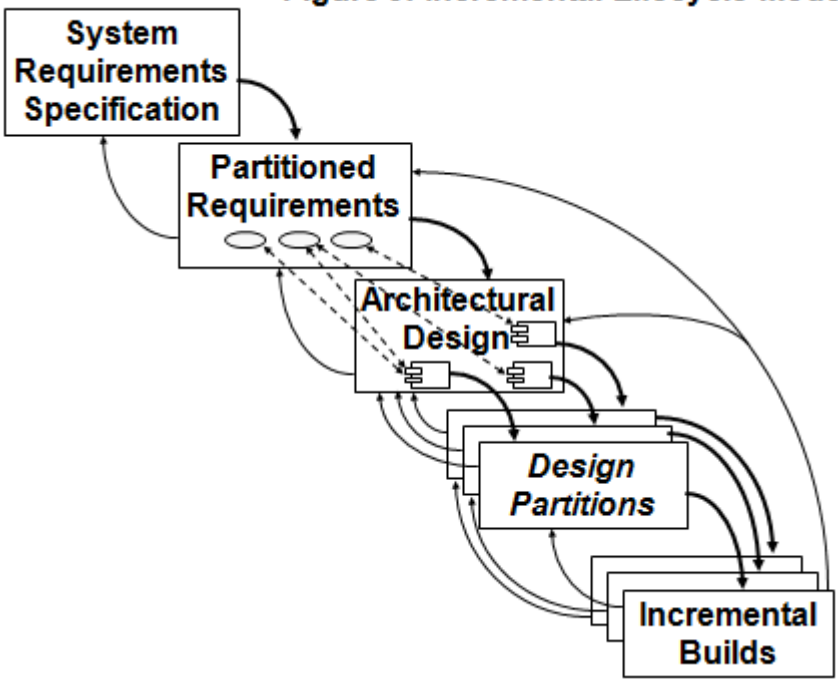
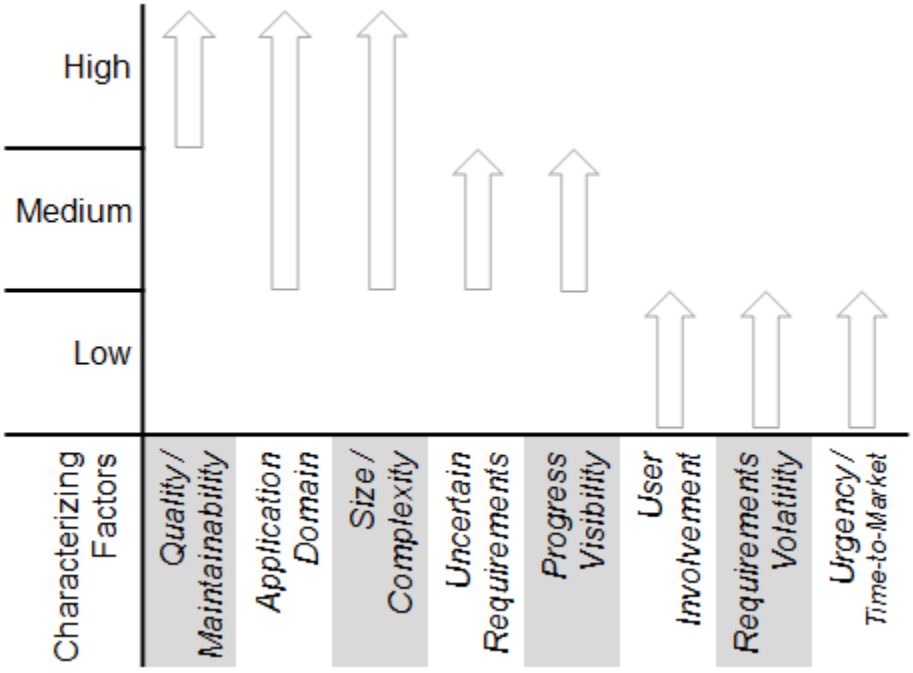


Figure 4: Incremental Lifecycle Model Characterized



9 Spiral Development

Originally proposed by Dr. Barry Boehm [7], the Spiral Model is an innovative process-rich model that possesses iterative and incremental features and can be challenging to interpret. Several newer models have spun-off from Boehm's model. The Rational Unified Process (RUP) [8] is a comprehensive iterative adaptation of this model – subsequently retro-fitted to fit better with Agile development. The various Agile methods [9] are also spin-offs in certain respects.

The Spiral Model, depicted in Figure 5 and characterized in Figure 6, advocates a risk-driven and plan-oriented iterative approach where each spiral is a development iteration that aims to establish a plan for the next spiral (a.k.a. iteration). Risk assessments prior to each spiral determine the activities scheduled for a given spiral/iteration. Reviews at the end of each spiral include an assessment of "lessons learned" that feed the next spiral. Early spirals systematically focus on consolidating the requirements and exploring technical problem areas through prototyping and simulating. Later spirals transition in more waterfall-like iterations of development. Concurrent spirals can represent increments of development.

The spiral lifecycle process model is flexible in that it to accommodate adaptation – the process engineer can tailor as much or as little discipline into the spirals as they may wish. This model encourages, but does not mandate, the notion of continuous risk assessment and the incorporation of both incremental and evolutionary development approaches. The Spiral Model enables many of the capabilities of the incremental and waterfall models while offering better project visibility, incorporating more user involvement, and dealing better with changing requirements. However, the Spiral Model can also be adapted to incorporate much technical and management overhead and therefore has the potential of escalating project schedules and costs beyond that of the waterfall and other iterative models. Project management and contracting can become more challenging than some of the other models as the Spiral Model can be adapted to specify disciplined processes for controlling requirements, architecture, risk mitigation, and process improvement (lessons learned) activities.

Figure 5: Boehm's Spiral Model [11]

Adapted from [7] (considerably simplified)

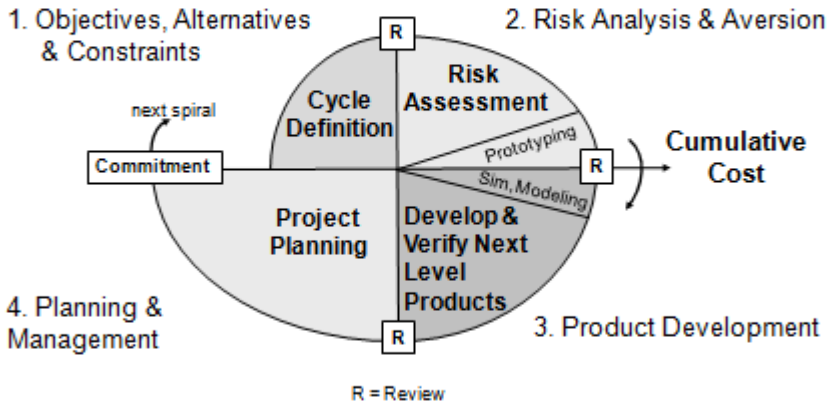
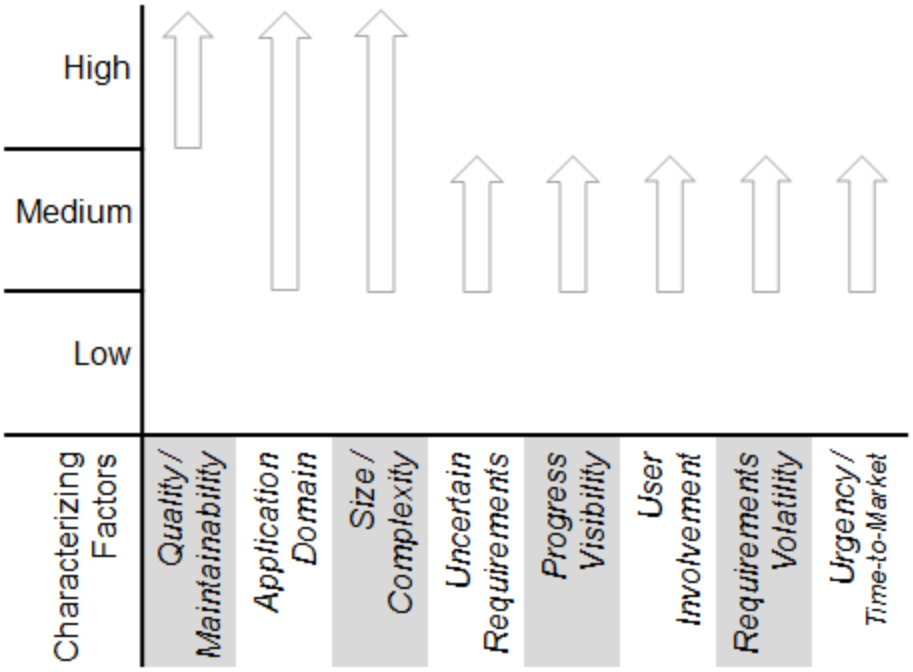


Figure 6: Spiral Lifecycle Model Characterized



10 Evolutionary Development

The Evolutionary Model is an iterative exploratory development model for solving hard (non-trivial) technical problems and uncertainties. Work products of this model are designed to discover technical solutions and elicit feedback. The Evolutionary Model focuses project stakeholders (developers, managers, customers, and users) on feasibility and requirements rather than on the targeted solution. Detailed functions and features, as well as product qualification tasks such as reviews and testing can be avoided. However, there is a danger that managers and customers may assume that the prototypes they are shown are of deliverable quality – when they are typically not - which can unreasonably inflate expectations. The Evolutionary Model is depicted in Figure 7 and characterized in Figure 8.

Evolutionary lifecycles aim at providing good visibility into the various technical problems encountered as well as encouraging user involvement and customer feedback. However, it is not intended for maintainability or the delivery of operational releases. Rather, this model should be used to explore difficult/risky technical issues and/or to validate uncertain requirements (e.g. user interface requirements, heuristic computations, and fuzzy decision logic). The Evolutionary Model is also very useful in experimental or research settings where the products are for more personal use, for example to produce a proof-of-concept prototype. The work products are not considered production quality, at least for large, medium-sized, complex or mission-critical projects where lifecycle maintainability is a priority. However, evolutionary development is a very useful process for front-ending a more structured waterfall or incremental development process. In fact, iterative models like the Rational Unified Process (RUP) for software development incorporate evolutionary development - but they endeavor to ensure that their work-products go through adequate design, review and testing steps before release into test and production.

Figure 7: Evolutionary Lifecycle Model [11]

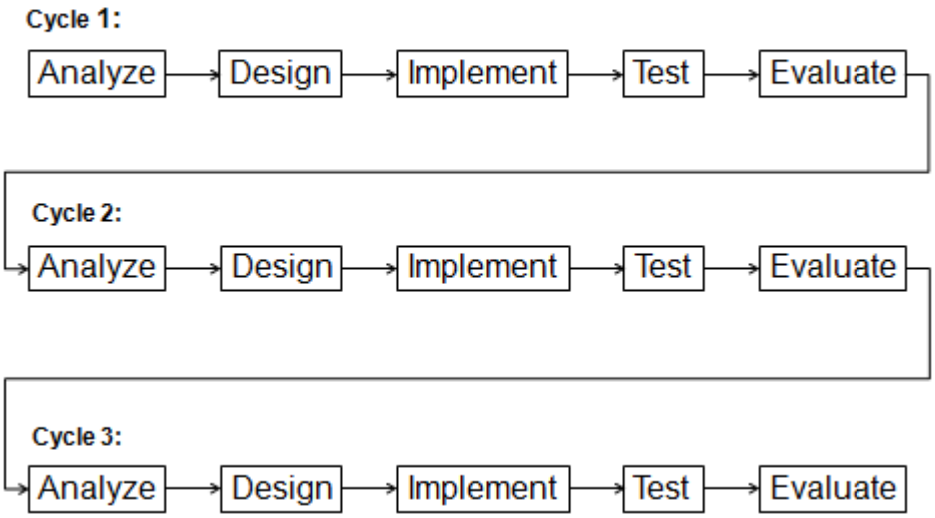
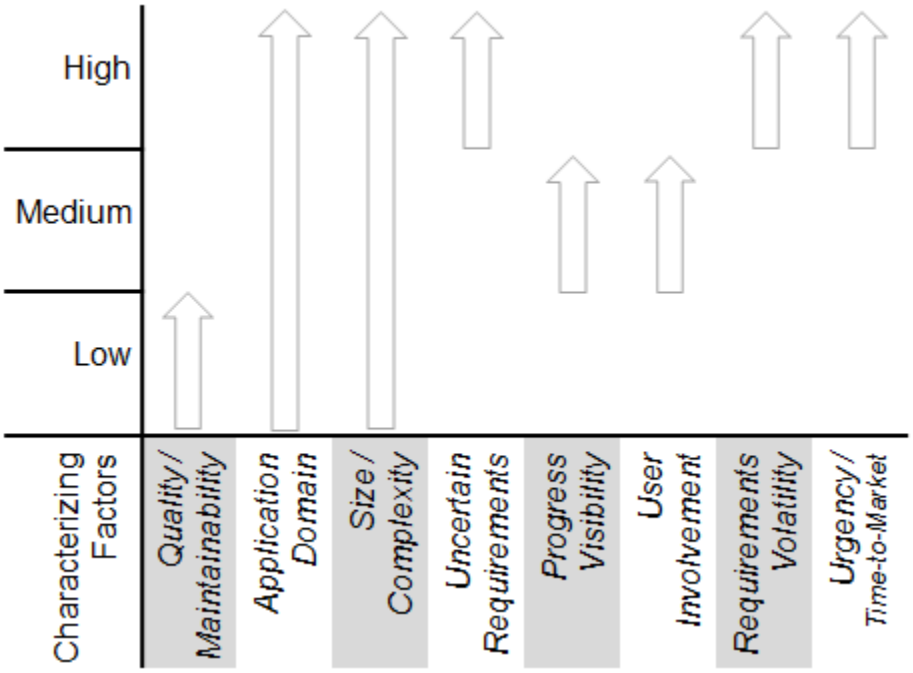


Figure 8: Evolutionary Lifecycle Model Characterized



11 Agile Development

Agile development represents a family of lighter weigh processes (e.g. XP, Scrum, etc.) that have been popularized over the last dozen years. Their roots can be traced back to iterative, evolutionary, and incremental development lifecycles which explains why agile is said to be iterative and incremental. These methods focus on developing working software over documentation, and they are said to embrace change and close customer involvement. Solutions are built up from customer and user-provided "stories" over short development iterations (typically 1-4 weeks). Stories are prioritized and put into a backlog; planning is "time-boxed"; some methods advocate pair-programming (e.g. XP) and/or "test-driven development" (TDD). Agile development is depicted in Figure 9 and characterized in Figure 10.

Like evolutionary development, these models deal with uncertain requirements – typically by requiring intimate customer and user involvement. Agile processes such as Scrum nudge evolving work products towards delivery and release by way of constant testing and re-factoring. The emphasis on working code contributes to their ability to demonstrate evolving functionality while light-weight documentation facilitates adaptability to changing circumstances. Favoring on-going re-factoring as stories are integrated, over big up-front requirements analysis and architectural design, enables early visibility and feedback. For large complex systems such continuous re-factoring could lead to fragile solutions and maintainability problems, especially if documentation is lacking. Also, customers may not participate as promised, and close customer involvement across large-scale projects tends to be more difficult to manage. Given the reduced emphasis on documentation, Agile projects may be more highly dependent on the skill and memory of their developers which may expose projects and companies to staff turnover risk. Overall, agile development appears to be more suitable for small-to-medium-sized projects of moderate criticality where schedule pressure and quickness to market are high priorities.

Figure 9: The Life of an Agile Story [11]

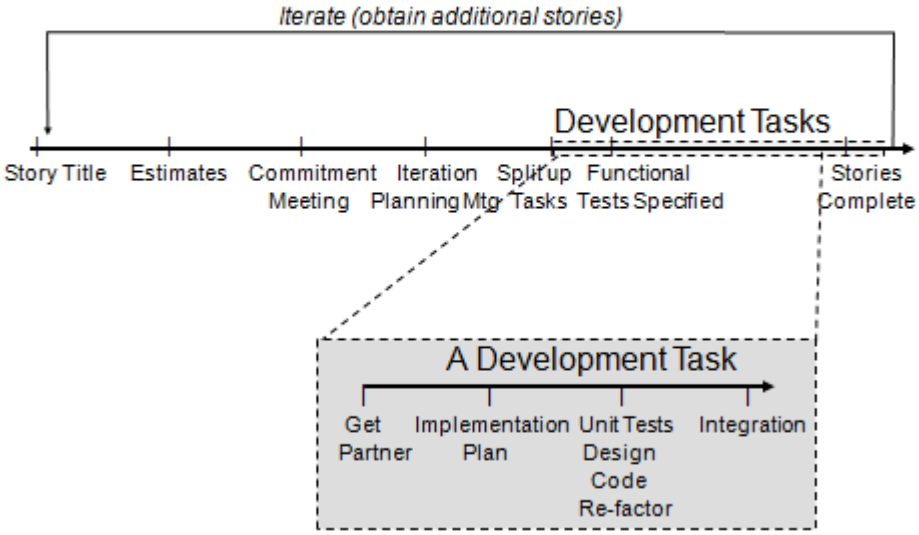
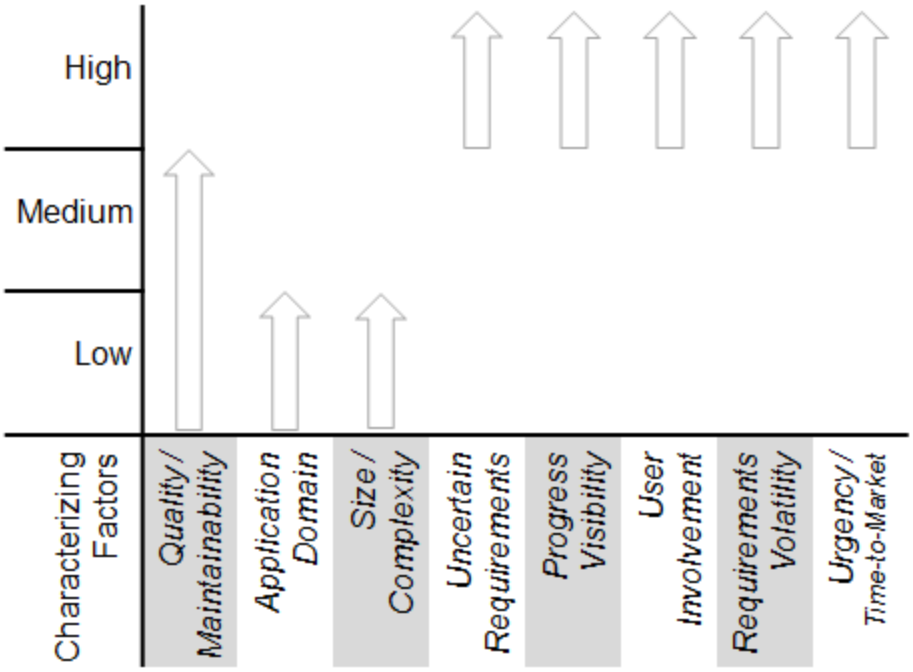


Figure 10: Agile Lifecycle Model Characterized



12 Project-to-Lifecycle Matching

Although we have not explored the problem of matching projects to lifecycles, we have attempted to illustrate the idea in the figures below.

Figure 11 presents a visualization of the project-to-lifecycle match for a hypothetical project that has already been characterized. The characterized project is represented as a single line graph that is superimposed over each of the (five) already characterized lifecycles. Figure 11 illustrates, for example, that the Waterfall Model meets or exceeds the project's needs in 5 out of 8 areas – failing to meet needs in 3 areas. Meanwhile, Agile development appears to meet or exceed project needs in 6 out of 8 areas, and the Spiral model appears to satisfy all eight factors. In search of other lifecycle choices, the matching process could go on to quantify the degree to which each factor is met, exceeded, or fails to be met by each lifecycle; and adjustments could be explored by reassessing or changing the character of the project, and/or tailoring/adapting each lifecycle to better fit project needs.

Meanwhile, Figure 12 illustrates a hypothetical project whose characteristics fail to match any of the discussed lifecycles. Observe in particular that this project aims to deliver a highly maintainable product, expects that the requirements will be very uncertain, and acknowledges that there will be considerable time-to-market pressure. This figure nicely verifies our intuition that none of the lifecycles appear to be capable of delivering such a highly constrained project. These mismatches between the project's apparent needs and what the various lifecycles best support imply that the project needs as articulated are unrealistic or unachievable. Probably the project's needs should be re-assessed. A potentially risky alternative to consider would be to put in place a new or hybrid lifecycle model that somehow overcomes the demands of the project.

Figure 11: Illustrating a Project Matched to Lifecycles

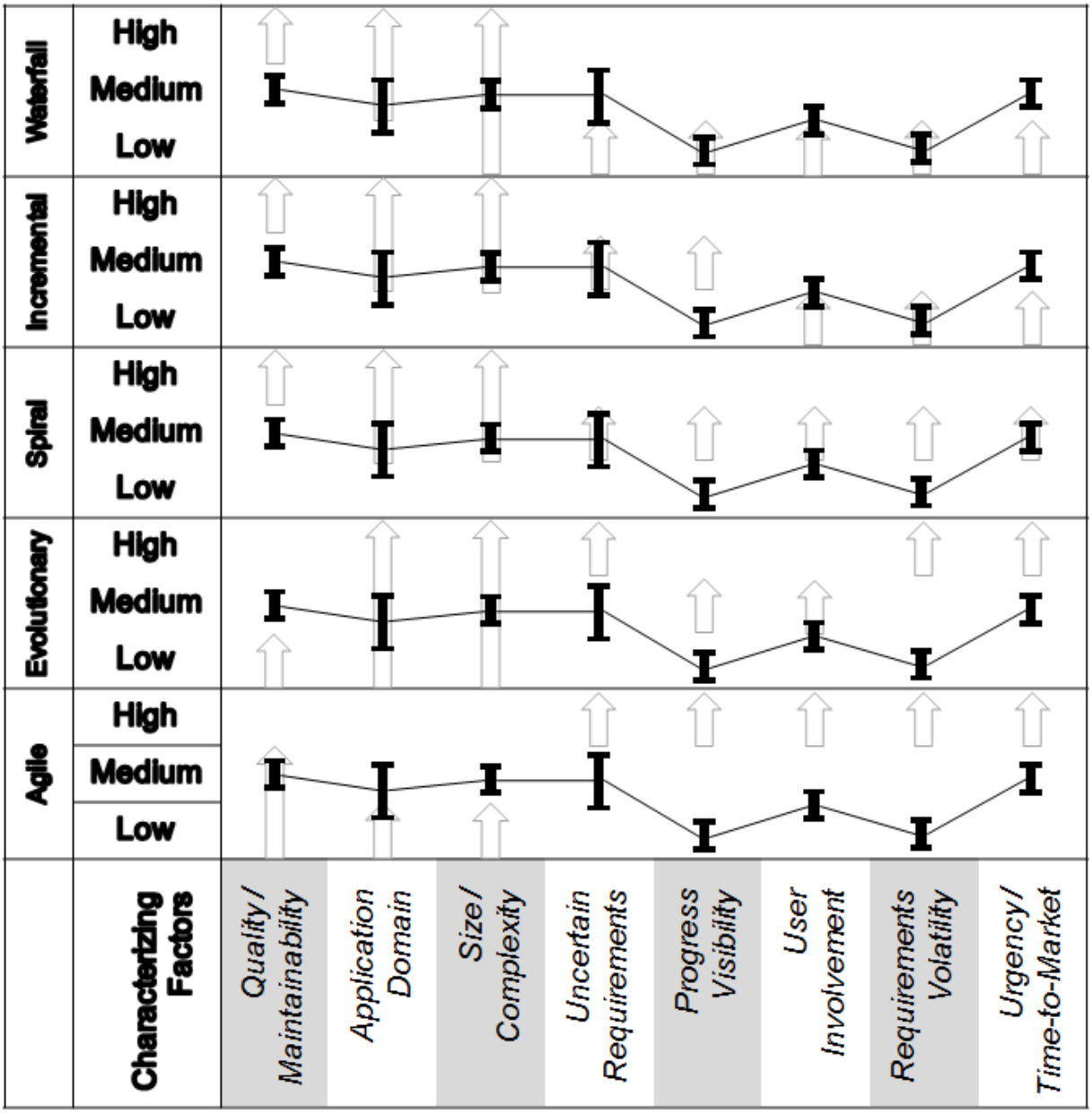
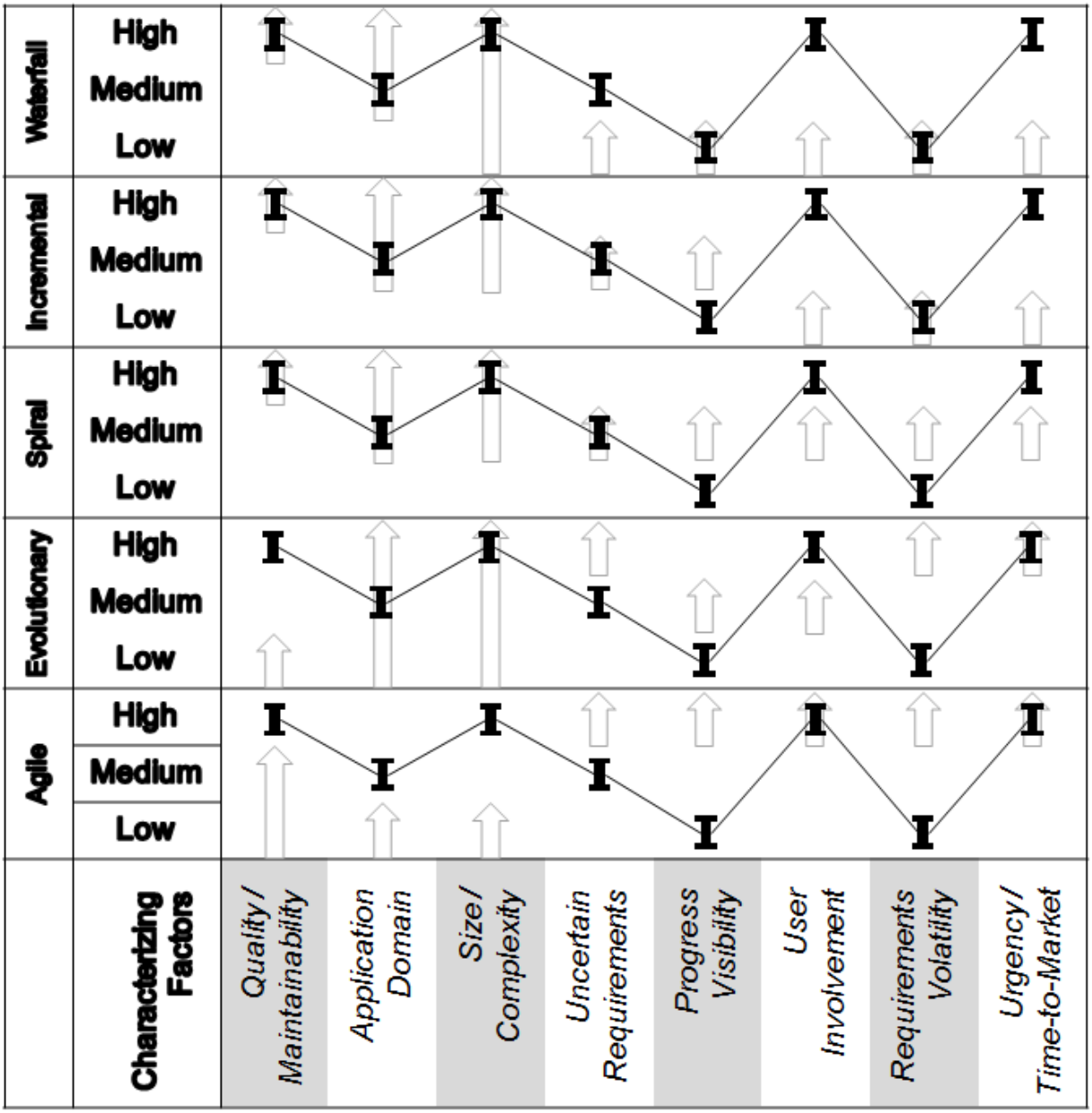


Figure 12: Illustrating a Project that Fails to Match a Lifecycle



13 Summing Up

This paper has suggested a general approach for lifecycle model selection and has explored the first step of such an approach, namely, to characterize the most popular lifecycle models in terms of eight critical selection factors. We acknowledge that the suggested characterization values have been subjectively arrived. Nevertheless, they have been distilled from the authors' 50 years of collective industry experience estimating, planning, managing and supporting systems and software projects. We therefore standby our assertion that the strategy put forth to characterize lifecycle models by way of critical project characterization factors could lead to a practical, balanced approach for selecting a suitable lifecycle model for a given project. We invite constructive criticism and validation from readers to help move towards such a goal.

An empirical approach for determining these factors and making trade-offs when selecting a lifecycle model has not been presented. However, it is hoped that this paper will motivate others to mount research and assessment projects that will build on the ideas presented.

References

[1] Kal Toth, "Which is the Right Software Process for your Problem?",Software Assoc of Oregon, 2005
[2] Kal Toth, "What's the hard part of software development anyway?",Software Assoc of Oregon, 2007
[3] Barry Boehm, Richard Turner, "Balancing Agility and Discipline", Addison Wesley, 2004
[4] Kal Toth, "Selecting Software Estimating Techniques that Fit the Software Process", PNSQC, 2008
[5] Kontio, A Case Study in Applying a Systematic Method for COTS selection, 1996, ICSE
[6] Dan Brook, Kal Toth, "Levels of Process Ceremony for Software Configuration Mgt", PNSQC, 2007
[7] Barry Boehm, "A Spiral Model for Software Development and Enhancement", Computer, V21,1988
[8] Philippe Kruchten, "The Rational Unified Process: An Introduction", Addison Wesley, 1999
[9] Alistair Cockburn, "Agile Software Development", Addison Wesley, 2002
[10] Kal Toth, "Change in Development Methodology", Hughes Systems Division Report, June 1, 1993
[11] Kal Toth, "Principles of Software Engineering", Lecture Notes, 2005-2010

Annex: Characterized Lifecycle Development Models

Waterfall Model		Relatively sequential with development phases, major milestones, & specified deliverables reviewed by stakeholders
Quality/ Maintainability	H	Advocates thorough requirements, architecture & design specification which facilitates quality reviews and inspections, and fosters quality documentation for downstream maintenance
Application Domain	M, H	Waterfall is best suited for projects that have stringent quality/criticality requirements, that is, those with high reliability, maintainability, availability, safety, and/or security requirements (are “mission-critical”)
Size / Complexity	L, M, H	A common misconception is that waterfall is best suited for large complex projects. It is more correct to say that waterfall is more suited to mission-critical projects of virtually any size
Uncertain Requirements	L	Waterfall assumes the requirements are fairly well understood
Progress Visibility	L	High levels of process formality and document deliverables will satisfy contractual requirements and facilitate progress payments but may give a false impression of true progress being achieved
User Involvement	L	User involvement typically confined to requirements phase
Requirements Volatility	L	Not very responsive to changing requirements due to formal change control process
Urgency	L	Although some iteration is allowed, the planned/controlled character of the waterfall model renders it less responsive to market forces.

Incremental Model		An iterative process that partitions large complex problems into independent parts, some of which may be mission-critical, concurrently develops and integrates the parts, and delivers multiple releases
Quality/ Maintainability	H	Advocates thorough requirements, architecture & design specification which facilitates quality reviews and inspections, and fosters quality documentation for downstream maintenance
Application Domain	M, H	Incremental development is well-suited for projects where mission-critical partitions can be identified and allocated to independent development increments – this is not applicable to small projects
Size / Complexity	M, H	Incremental development is well-suited for larger and more complex projects – project partitioning not needed on smaller projects
Uncertain Requirements	M	If requirements are well-partitioned at the top level, incremental development can localize the problem of uncertain requirements.
Progress Visibility	M	Concurrent development enables independent monitoring of parts which can elevate visibility into project progress
User Involvement	L	User involvement typically confined to requirements phase
Requirements Volatility	L	Not very responsive to changing requirements due to formal change control process
Urgency	L	The planned/controlled character of the incremental model renders it less responsive to market forces.

Spiral Model		A risk-driven plan-oriented iterative model where each spiral is a development iteration that aims to establish a plan for the next spiral (a.k.a. iteration).
Quality/ Maintainability	H	Risk-driven approach with lessons-learned well-suited systematically developing quality specifications for life-cycle maintenance.
Application Domain	M, H	The risk assessment aspect of this progressive development approach is well-suited to address the technical and management risks posed by mission-critical requirements
Size / Complexity	M, H	The risk assessment aspect combined with process decomposition into development spirals and lessons-learned is well-suited to address the technical and management risks posed by large complex applications
Uncertain Requirements	M	Early spirals systematically focus on consolidating the requirements & exploring technical problems thru prototyping & simulation
Progress Visibility	M	Though more challenging for project management, the risk-driven approach provides progress visibility from an additional perspective (in addition to delivered documents and functionality)
User Involvement	M	Though user involvement is not explicitly call up, a majority of technical and management risks encountered will have direct impacts on the customer and/or users which motivates planning and execution explicitly incorporating customer and user involvement
Requirements Volatility	M	The spiral model represents a compromise solution for dealing with requirements volatility. Risk assessment will help stabilize requirements early in the project in the areas that are most changable
Urgency	M	The risk-driven strategy can be employed to identify spirals of development that should be accelerated and developed independently from challenging areas to meet urgency and time-to-market needs

Evolutionary Model		An iterative exploratory development model for solving hard (non-trivial) technical problems and uncertainties – they are not considered to be of operational/deliverable quality
Quality/ Maintainability	L	Evolutionary development does not deliver solutions of deliverable quality – however, this strategy may have a positive impact on the quality of final deliverables and is likely to improve schedules and costs
Application Domain	L, M, H	Evolutionary development is generally applicable across all domains
Size / Complexity	L, M, H	Generally applicable regardless of project size or complexity. Larger projects are will offer more opportunities for evolutionary development; very small projects may not need more that evolutionary development
Uncertain Requirements	H	A key application of evolutionary development is to explore requirements so as to reduce uncertainty, schedules and costs by reducing the need for supporting requirements change
Progress Visibility	M	Evolutionary development encourages focusing on project uncertainties and will improve visibility into progress rather than hiding uncertainties until “just too late”
User Involvement	M	Focusing on usage and user interface issues, especially early in the project, will encourage user and customer involvement and reduce schedule and cost risks
Requirements Volatility	H	Early exploration into requirements with the involvement of users can help stabilize requirements early and reduce requirements churn
Urgency	H	This exploratory approach can be used to focus attention on difficult problems early thereby removing risk from the project’s critical path

Agile Model		Focus is on developing working software over documentation, embracing change and close customer involvement. Stories are prioritized and put into a backlog; planning is "time-boxed"; some methods advocate pair-programming (e.g. XP) and/or "test-driven development" (TDD).
Quality/ Maintainability	L-M	Agile solutions favor frequent re-factoring over developing a stable architecture which may result in brittle operational solutions Emphasis on working software over documentation makes agile development more vulnerable to employee turnover
Application Domain	L	Although various authors have claimed that Agile development is suitable for mission-critical applications, documented evidence is anecdotal at this time
Size / Complexity	L	Although various authors have claimed that Agile development is suitable for large and complex applications, documented evidence is anecdotal at this time
Uncertain Requirements	H	Agile is specifically oriented towards addressing requirements uncertainty - user stories are described very informally, presented as prototypes to users, and refined until accepted by users
Progress Visibility	H	Progress visibility is realized in terms of the released stories and their functions and features. Note that completeness of testing and documentation are not explicitly used to provide progress visibility
User Involvement	H	Agile is heavily dependent on customer buy-in and commitment to ensure that users are involved deeply in the project
Requirements Volatility	H	All stakeholders are committed to embrace change and prepared to re-engineer and re-factor the current software build at any time
Urgency	H	Each story is meant to be releasable to the user base – hence, provided the latest build(s) is of releasable quality, an operational product should be available on a timely basis

Summary Characterization of Lifecycle Models	**Waterfall**	**Incremental**	**Spiral**	**Evolutionary**	**Agile**
Quality/ Maintainability	H	H	H	L	L-M
Application Domain	M, H	M, H	M, H	L, M, H	L
Size / Complexity	L, M, H	M, H	M, H	L, M, H	L
Uncertain Requirements	L	M	M	H	H
Progress Visibility	L	M	M	M	H
User Involvement	L	L	M	M	H
Requirements Volatility	L	L	M	H	H
Urgency	L	L	M	H	H