METHODS & TOOLS

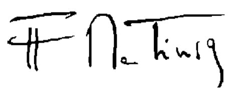
Global knowledge source for software development professionals **ISSN 1023-4918**

Spring 2002 (Volume 10 - number 1)

Application Development Without Users

In 1981, the great guru and trend surfer James Martin wrote a book called "Application Development Without Programmers". With this provocative title, James Martin wanted to demonstrate that in the near future, users will develop themselves their applications with the 4th generation languages - like Natural, Mantis, SQL or Basic - that allow to program without "being a programmer". This should have been the end of the developers. For some (most?) of you, speaking about James Martin, 1981 or Mantis is like talking about the Palaeolithic era of software engineering. The point is to show that the relationships between end users and software developers have been confrontational for a long time. There is a little bit of "us" versus "them" situation, part of both sides considering the others like... printers: something that can provide often more problems than benefits. History has shown that James Martin was not completely right. Today many end users have gained a better understanding of information systems, and many more think that they are experts on this subject. Even if end users have developed some applications on personal computers, software developers perform the vast majority of the development of core production and datawarehousing systems.

However, if the users have moved closer to the IS department, the software developers have made little efforts to "push" their knowledge towards users. I think there is little discussion about the importance of the role of users during specific phases of the software development process like requirements gathering and application validation, but in these areas the culture of information system development has been rarely teached to users. I do not know organisations that have given training on system analysis or test cases creation to their users, even if a better formalisation of requirement and a more complete validation of the application are major success factors for a project. The cynical reader will object that these training classes are also rarely given to software developers... and they are not completely wrong! The situation looks however funny: everybody is emphasising the role of the users in software development, but nobody is giving them the tools to improve their contribution to software development projects... assuming that they want to improve it.



Inside

Understanding the Unified Process (UP)

Sinan Si Alhir, salhir@earthlink.net, http://home.earthlink.net/~salhir

Introduction

The systems engineering discipline focuses on an elegant universe we call reality wherein the two dimensions of time and space establish the landscape for the intertwining dance between the two natural forces of change and complexity. It is within this arena that the key ingredients of teams and people, methodologies and processes, and tools and enabling technologies converge to bridge the chasm between vision and reality. At the core of every mature discipline, from the arts to the sciences and engineering, is a common language and common approaches enabling practitioners to collaborate and the discipline to evolve; and at the heart of this evolution is capturing or acquiring, communicating or sharing, and leveraging or utilizing knowledge. Language establishes the boundaries of thought and behavior, it defines concepts; methodology and process establish behavior within that boundary, they apply concepts; and tools establish the automation of behavior within that boundary, they automate the application of concepts. Quite simply, if we can't think it, we can't do it nor communicate it, and if we can't do it, we can't automate it! Within the information systems and technology industry, the Unified Process (UP), Rational Unified Process (RUP), Unified Modeling Language (UML), and Software Process Engineering Metamodel (SPEM) are at the heart of this evolution.

The Unified Process (UP) and Rational Unified Process (RUP)

The Unified Process (UP) is a use-case-driven, architecture-centric, iterative and incremental development process framework that leverages the Object Management Group's (OMG) UML and is compliant with the OMG's SPEM. The UP is broadly applicable to different types of software systems, including small-scale and large-scale projects having various degrees of managerial and technical complexity, across different application domains and organizational cultures.

The UP emerged as the unification of Rational Software Corporation's Rational Approach and Objectory AB's Objectory process in 1995 when Rational Software Corporation acquired Objectory AB. Rational Software Corporation developed the Rational Approach as a result of various customer experiences, and Ivar Jacobson created the Objectory process primarily as a result of his experience with Ericsson in Sweden.

The UP is an "idea," a process framework that provides an infrastructure for executing projects but not all of the details required for executing projects; essentially, it is a software development process framework, a lifecycle model involving context, collaborations, and interactions. The UP is documented in the book entitled "The Unified Software Development Process" by the Three Amigos (Grady Booch, James Rumbaugh, and Ivar Jacobson) (Addison-Wesley, 1999). The Rational Unified Process (RUP) is a process product developed and marketed by Rational Software Corporation that provides the details required for executing projects using the UP, including guidelines, templates, and tool assistance; essentially, it is a commercial process product providing the details or content for the UP framework. When applying the UP or RUP on a project, a Development Case, an instance of the process framework, specifies what elements of the UP or RUP are utilized throughout the project. A "RUP-based" Development Case is an instance of the RUP (and the UP), a configured or tailored subset of the RUP content (which may possibly be further augmented) that addresses the breadth and depth of the UP framework. A "UP-based" Development Case is an instance of the UP that addresses the breadth and depth of the UP framework.

The Unified Modeling Language (UML) and Software Process Engineering Metamodel (SPEM)

The Unified Modeling Language (UML) is an evolutionary general-purpose, broadly applicable, tool-supported, and industry-standardized modeling language or collection of modeling techniques for specifying, visualizing, constructing, and documenting the artifacts of a system-intensive process. The UML is broadly applicable to different types of systems (software and non-software), domains (business versus software), and methods and processes. The UML enables and promotes (but does not require nor mandate) a use-case-driven, architecture-centric, iterative and incremental process.

The UML emerged from the unification that occurred in the 1990s within the information systems and technology industry. Unification was led by Rational Software Corporation and the Three Amigos. The UML gained significant industry support from various organizations via the UML Partners Consortium and was submitted to and adopted by the OMG as a standard (November 1997).

As the UML is an industry-standardized modeling language for communicating about systems, the Software Process Engineering Metamodel (SPEM) is an industry-standardized modeling language for communicating about processes and process frameworks (families of related processes) but it does not describe process enactment (the planning and execution of a process on a project). The SPEM began to emerge after the UML standardization effort, gained significant industry support from various organizations, and was adopted by the OMG as a standard (November 2001).

System Development, Systems, Models, and Views

The system development lifecycle process involves a problem-solving process at a macro-level and the scientific method at a micro-level. Requirements may be characterized as problems. Systems that address requirements may be characterized as solutions. Problem solving involves understanding or conceptualizing the problem or requirements by representing and interpreting the problem, solving the problem by manipulating the representation of the problem to derive or specify a representation of the solution, and implementing or realizing and constructing the solution or system that addresses the requirements by mapping the representation of the solution onto the solution world. Within each problem-solving step, the scientific method involves planning or predicting a hypothesis, executing or empirically testing the hypothesis, evaluating the hypothesis against the results, and deriving a conclusion that is used to update the hypothesis. These macro-level and micro-level processes are very natural and often occur subtly and sometimes unconsciously in system development!

The UML facilitates and enables the problem-solving process. It facilitates specifying, visualizing, understanding, and documenting the problem or requirements; capturing, communicating, and leveraging strategic, tactical, and operational knowledge in solving the problem; and specifying, visualizing, constructing, and documenting the solution or system that satisfies the requirements. It enables capturing, communicating, and leveraging knowledge concerning systems using models, architectural views, and diagrams.

A system is a purposefully organized collection of elements or units. The architecture of a system entails two dimensions, the structural dimension and behavioral dimension, within its context. The structural or static dimension involves what elements constitute the system and their relationships. The behavioral or dynamic dimension involves how these elements collaborate and interact to satisfy the purpose of the system and provide its functionality or behavior.

A model is a complete abstraction of a system that captures knowledge (semantics) about a problem and solution. An architectural view is an abstraction of a model that organizes knowledge in accordance with guidelines expressing idioms of usage. A diagram is a graphical projection of sets of model elements that depicts knowledge (syntax) about problems and solutions for communication. Within the fundamental UML notation, concepts are depicted as symbols and relationships among concepts are depicted as paths (lines) connecting symbols.

Methodologies and Process Frameworks

A program is a collection or portfolio of projects. A project is a specific problem-solving effort that formalizes the "work hard and hope for the best" approach. A method specifies or suggests how to conduct a project. A method's descriptive aspect specifies or suggests what knowledge is captured and communicated regarding a problem and solution. A method's prescriptive aspect specifies or suggests how knowledge is leveraged to solve the problem. A process is the execution of a method on a project.

A methodology is a discipline or taxonomy, or well-organized collection, of related methods that addresses who does what activities on what work products, including when, how, why, and where such activities should be done. Workers (who), activities (how), work products (what), and the heuristics concerning them are commonly known as process elements. Methodologies group methods as a family, methods describe processes, and processes execute methods on projects.

To provide more flexibility and scalability to address increasingly more diverse problems, where applying a single method may be insufficient and applying a whole methodology may be impractical, a subset of a whole methodology may be applied where the methodology is called a process framework and the actual subset of all of its methods that are applied on a specific project is called a process instance.

A process framework specifies or suggests who does what activities on what work products, including when, how, why, and where such activities should be done for various types of projects. A process instance specifies or suggests who does what activities on what work products, including when, how, why, and where such activities should be done for a specific project. Process frameworks describe process instances as a more flexible and scaleable family of related processes, and process instances execute a subset of a process framework on projects. The UP is a process framework and Development Cases are process instances.

The Unified Process (UP)

To effectively and successfully apply the UP, we must understand collaborations, contexts, and interactions. As an effort or project leverages the UP, collaborations focus on the elements of the project, context focuses on the process framework for the project, and interactions focus on the execution of the project. Figure 1 shows the various elements of the UP.

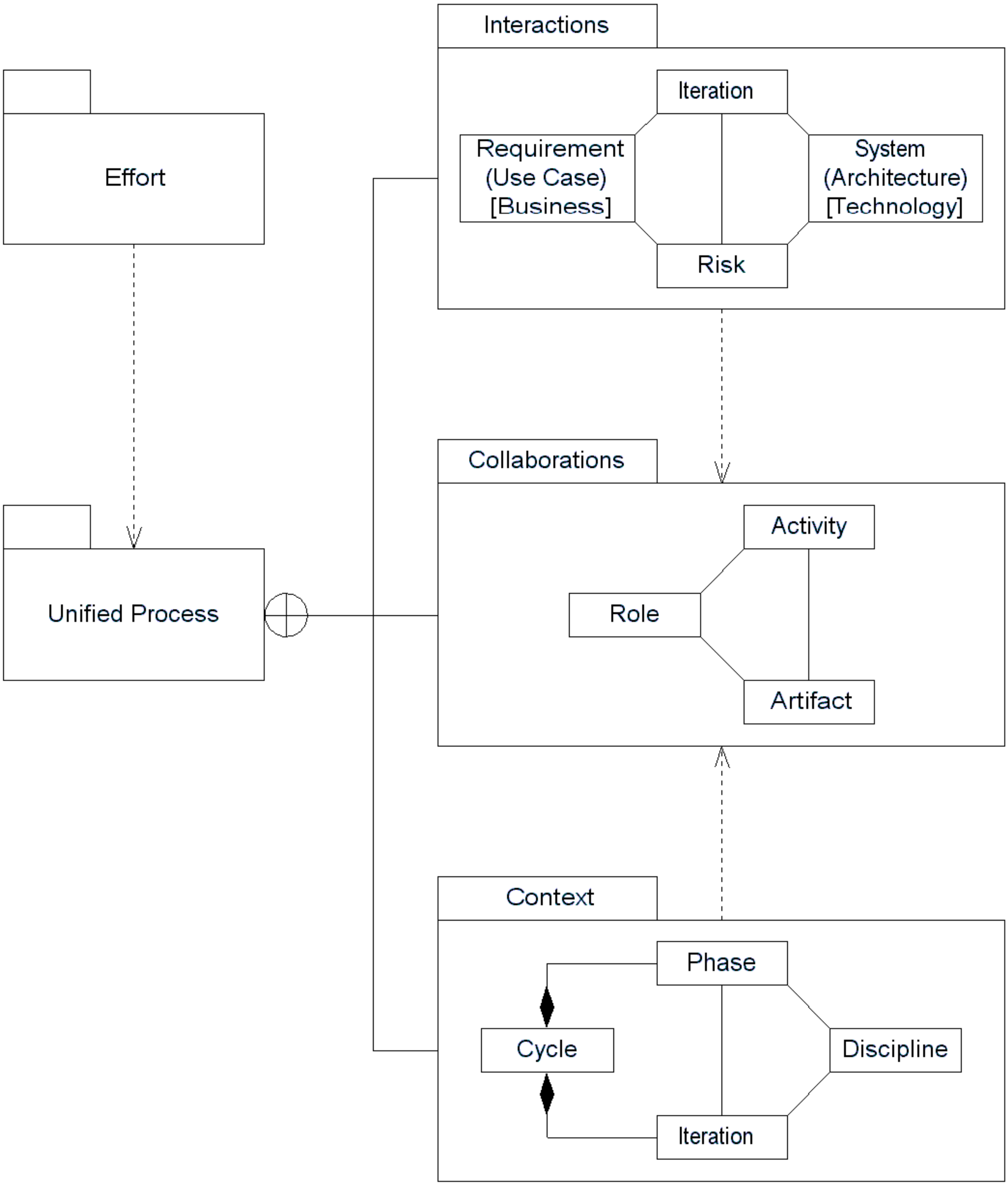


Figure 1: Elements of the Unified Process (UP).

Collaborations

A collaboration involves an interaction within a context. A collaboration captures who does what activities (how) on what work products. Thus, it establishes the elements of a project.
A role is an individual or team who has responsibility for activities and artifacts. An activity is a unit of work, composed of steps, that is performed by a role. An artifact is an element of information that is the responsibility of a role and that is produced or consumed by activities. The UP defines numerous roles, artifacts, and activities.

Contexts

A context emphasizes the structural or static aspect of a collaboration, the elements that collaborate and their conglomeration or spatial relationships. A context captures when and where such activities should be done and work products produced and consumed. Thus, it establishes the context for a project. Figure 2 shows the context established by the UP.

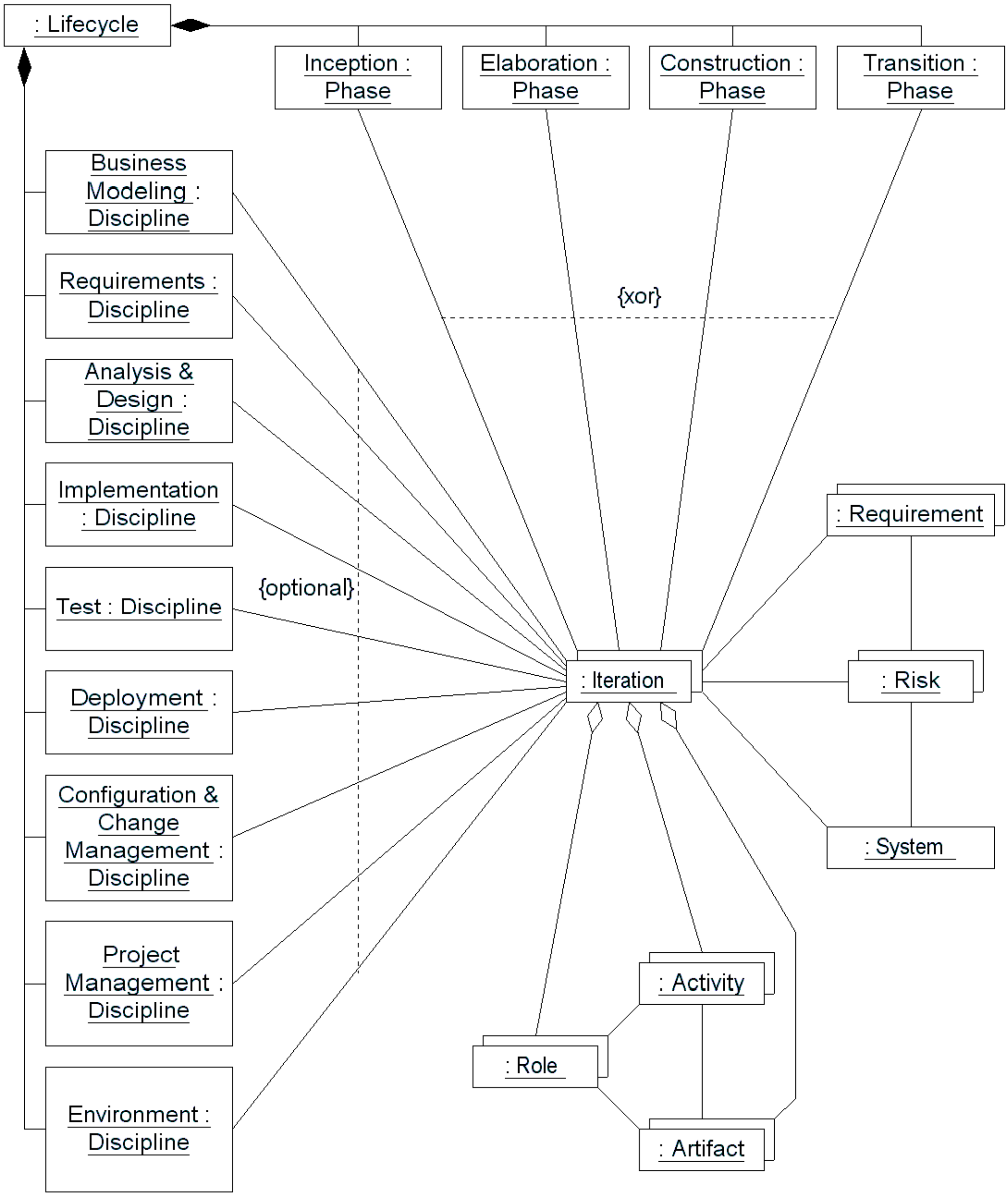


Figure 2: Context established by the Unified Process (UP).

A project requires a management perspective to manage the effort and a technical perspective to execute and perform the technical work. The lifecycle of a project is composed of phases wherein iterations involve disciplines. A development cycle is composed of sequential phases resulting in a major system release called a system generation. For example, system generations may include versions 1.0, 2.0, 3.0, and so forth. A phase is a major milestone, a management decision point focused on managing business risk. Phases embody the macro-level problem-solving process. An iteration is a minor milestone, a technical decision point focused on managing technical risk, resulting in a minor system release called a system increment. For example, system increments may include versions 1.1, 1.2, 2.5, and so forth. Iterations embody micro-level applications of the scientific method. A discipline is an area of concern or theme wherein workflows describe the flow of work and wherein workflow details describe the collection of activities (with their associated roles and artifacts) often done together.

The UP defines the following four phases:

- The Inception phase, concluding with the Objective milestone, focuses on establishing the project's scope and vision; that is, establishing the business feasibility of the effort and stabilizing the objectives of the project.
- The Elaboration phase, concluding with the Architecture milestone, focuses on establishing the system's requirements and architecture; that is, establishing the technical feasibility of the effort and stabilizing the architecture of the system.
- The Construction phase, concluding with the Initial Operational Capability milestone, focuses on completing construction or building of the system.
- The Transition phase, concluding with the Product Release milestone, focuses on completing transitioning or deployment of the system to the user community.

The UP defines the following three supporting disciplines:

- The Configuration & Change Management discipline focuses on managing the configuration of the system and change requests.
- The Project Management discipline focuses on managing the project.
- The Environment discipline focuses on the environment for the project, including the process and tools.
- The UP defines the following six core disciplines:
- The Business Modeling discipline focuses on understanding the business being automated by the system and capturing such knowledge in a Business model.
- The Requirements discipline focuses on understanding the requirements of the system that automates the business and capturing such knowledge in a Use-case model.
- The Analysis & Design discipline focuses on analyzing the requirements and designing the system and capturing such knowledge in an Analysis/Design model.
- The Implementation discipline focuses on implementing the system based on the Implementation model.
- The Test discipline focuses on testing (evaluating) the system against the requirements based on the Test model.
- The Deployment discipline focuses on deploying the system based on the Deployment model.

The distribution of effort across phases, iterations, and disciplines focuses on addressing business and technical risks. During the Inception phase, most of the effort is distributed across the Business Modeling and Requirements disciplines. During the Elaboration phase, most of the effort is distributed across the Requirements, Analysis & Design, and Implementation disciplines. During the Construction phase, most of the effort is distributed across the Analysis & Design, Implementation, and Test disciplines. During the Transition phase, most of the effort is distributed across the Test and Deployment disciplines. The supporting disciplines are generally distributed throughout the four phases. The overall objective is to produce the resulting system; therefore, all of the core disciplines are engaged as soon as possible without introducing risk to the project; that is, practitioners are responsible for determining which disciplines to engage and when they should be engaged.

Interactions

An interaction emphasizes the behavioral or dynamic aspect of a collaboration, the elements that collaborate and their cooperation or temporal communication. An interaction captures when and why such activities should be done and work products produced and consumed. Thus, it establishes the execution of a project as it is governed by various forces.

As minor milestones occur within major milestones, technical decision points occur within management decision points such as to align technical tactics and operations with business strategy and objectives -- essentially, establishing a bridge between business and technical forces.

An iteration is a step or leg along a path or route to a destination. An iteration is planned and is not ad hoc, has evaluation criteria, and results in demonstrable progress. An iteration is iterative in that it is repetitive and involves work and rework, incremental in that it is additive and involves more than rework alone, and parallel in that work may be concurrent within the iteration.

A use-case is a functional requirement. For example, functionality to login or logout of a system, input data, process the data, generate reports, and so forth. As the UP is use-case driven, use cases drive or feed iterations. That is, iterations are planned and evaluated against "chunks" of functionality (or parts thereof) such as to manage agreement with users and trace project activities and artifacts back to requirements. Thus, accounting for business forces by planning and evaluating iterations against functional requirements. Non-functional requirements (usability, reliability, performance, and other such characteristics) are incrementally considered as use cases evolve through the disciplines.

A system has an architecture. For example, the architecture of a system includes a collection of elements and how they collaborate and interact, including various subsystems for handling security, input and output, data storage, external communications, reporting, and so forth. As the UP is architecture-centric, iterations focus on architecture and evolving the system. That is, iterations demonstrate progress by evolving a puzzle of "chunks" such as to manage the complexity and integrity of the system. Thus, accounting for technical forces by demonstrating progress via the production and evolution of the real system.

A risk is an obstacle to success, including human, business, and technical concerns or issues. For example, human risks include having insufficient, untrained, or inexperienced human resources, and so forth; business risks include having insufficient funding, time, or commitment from the business community, and so forth; and technical risks include having an insufficient understanding of the requirements or technology, using unproven technology, using technology that will not sufficiently address the requirements, and so forth. As the UP is risk-confronting, iterations confront risk and leverage feedback from previous iterations to confirm progress and discover other unknown risks. That is, iterations confront risk that is derived from use cases and architecture such as to achieve project success,thus reconciling business and technical forces.

An iteration is a time-box with a fixed beginning and end wherein a collection of collaborations are planned, executed, and assessed in order to progressively demonstrate progress. The beginning and end are negotiated among stakeholders, management and technical members of the project community who impact and are impacted by the effort. Use cases that feed an iteration are selected based on the highest risks they confront. A use case may evolve across any number of iterations and may evolve through any number of core disciplines in an iteration. An iteration results in one or more intermediate builds or operational versions of the system. An iteration results in a single internal or external baselined and evaluated release of the system. The feedback and lessons-learned gained from an iteration feed into future iterations. Within an iterative approach, metrics and estimates are also iteratively derived, and trends across iterations form the basis for metrics and estimation for the overall effort. The duration of an iteration is inversely proportional to the level of risk associated with the effort. As iterations execute, they only minimally overlap. Development cycles and phases may also be time-boxed; as development cycles, phases, and iterations are planned, the further the plans are in the future, the less accurate the estimates.

Although iterations are composed of the same disciplines as a "pure waterfall" approach, there are key distinctions. A waterfall approach aims for one hundred percent completeness of activities and artifacts of a discipline before proceeding to the next discipline; however, an iterative approach involves iterative collaboration and aims for incremental refinement and evolving levels of detail of artifacts throughout the lifecycle. A waterfall approach does not offer explicit opportunities for partial deployment of a system or explicit opportunities for introducing change into the lifecycle, and is therefore quite reactive to change; however, an iterative approach does offer explicit opportunities for partial deployment of a system at the end of an iteration and explicit opportunities for introducing change into the lifecycle at the end of an iteration and before the next iteration, and is therefore quite proactive or responsive to change. A waterfall approach progresses serially through disciplines; however, an iterative approach may progress forward or backward across phases to change focus and involves various disciplines in order to address risk.

Iterations

To effectively and successfully apply the UP, we must understand iterations and how they are applied in linear, sequential, and iterative approaches.

An iteration is planned, executed, and evaluated. Use cases and risks are prioritized, and use cases are ranked against the risks they mitigate. When planning an iteration, those use cases that address the highest risks and can be accommodated given the iteration's limiting factors (funding, time, resources, and so forth) are selected for driving the iteration. When executing an iteration, use cases evolve through the core disciplines and the system and its architecture evolve.

However, use cases need not evolve through every core discipline in a single iteration. When evaluating an iteration, actual results are compared against the planned objectives of the iteration, and plans and risks are updated and adjusted. The overall objective is to produce the resulting system; therefore, all of the core disciplines are engaged as soon as possible without introducing risk to the project; that is, practitioners are responsible for determining which disciplines to engage and when they should be engaged.

Linear Approaches

When the first group of iterations focus primarily on business modeling, next group of iterations focus primarily on requirements; and so on through the core disciplines, the team steadily learns more about the problem before learning about the solution as the effort progresses across phases. The effort results in a complete system only at the end of the development cycle. This is commonly known as a linear approach. Figure 3 shows the overall pattern of how effort is distributed using a linear approach.

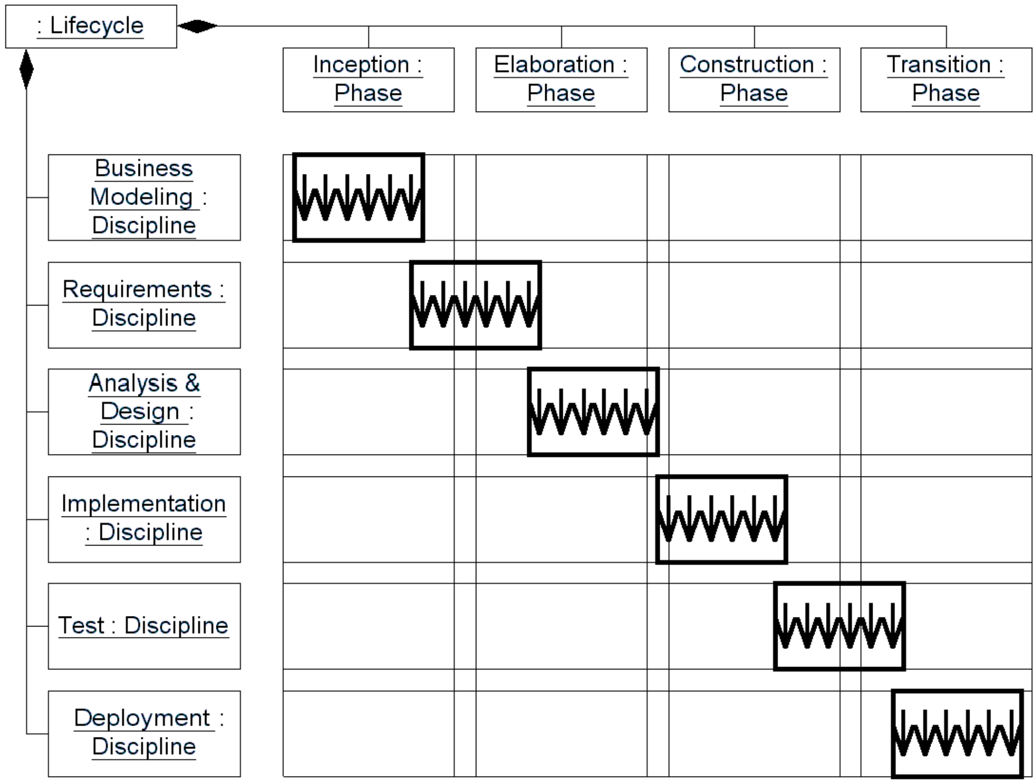


Figure 3: Linear Approach.

Linear iterations are too macro-focused towards phases where disciplines are more discretely distributed across phases; thus, the balance between business and technology is skewed by the management members of the community. The effort attempts to force all use cases through a few disciplines in an iteration often because the management members of the community perceive everything as a business risk that must be immediately controlled.

Linear iterations tend to delay architecture-related risk-confrontation and risk-resolution while perceiving everything as a business-related or use-case-related risk. However, this approach delays necessary validation of the system and its architecture, and precludes opportunistic deployment of the system throughout the lifecycle. Essentially, this is a "pure waterfall" approach where disciplines are distributed across iterations.

Sequential Approaches

When use cases evolve through every core discipline in a single iteration, the team steadily learns more about the solution for a limited portion of the problem as the effort progresses across phases. The effort results in a system that only addresses a subset of the requirements, which may or may not be deployable or usable throughout the development cycle, and results in a complete system only at the end of the development cycle. This is commonly known as a sequential approach.

Figure 4 shows the overall pattern of how effort is distributed using a sequential approach.

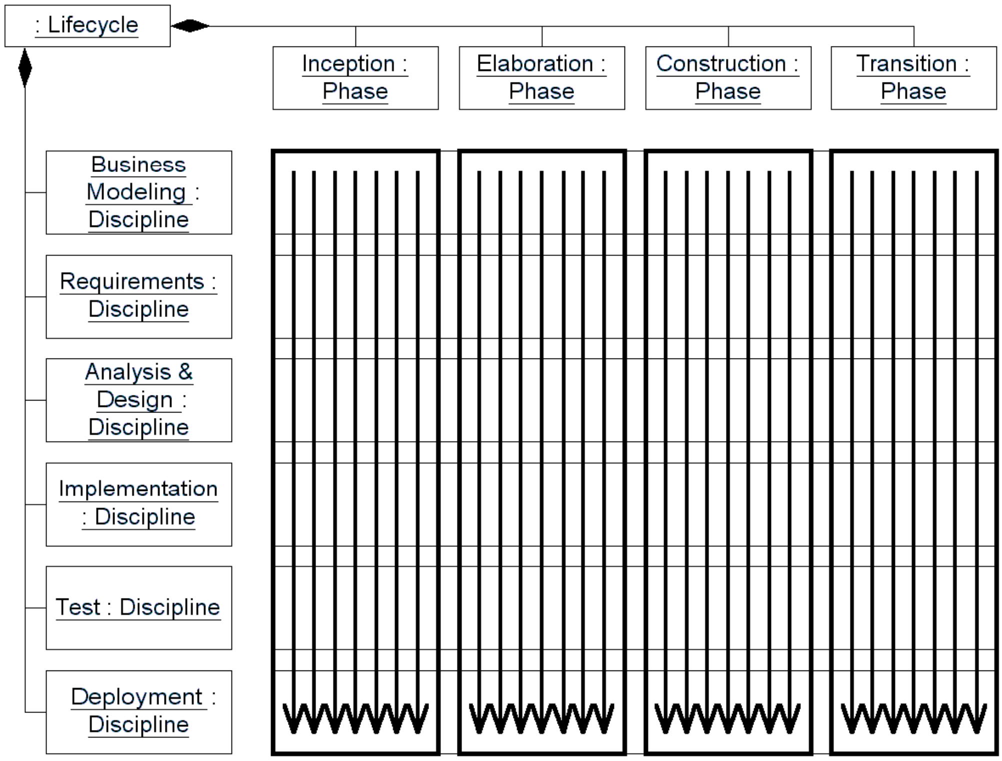


Figure 4: Sequential Approach.

Sequential iterations are too micro-focused towards iterations where disciplines are more discretely distributed within iterations; thus, the balance between business and technology is skewed by the technical members of the community. The effort attempts to force a few use cases through all disciplines in an iteration often because the technical members of the community perceive everything as a technical risk that must be immediately addressed.

Sequential iterations tend to delay use-case-related risk-confrontation and risk-resolution while perceiving everything as a technology-related or architecture-related risk. However, this approach results in a system that may be difficult to integrate and validate, and delays sufficient coverage when exercising the architecture. Essentially, a lack of architectural coverage increases the probability of encountering a use case that completely invalidates the architecture derived from preceding iterations.

Iterative Approaches

An iterative approach involves using a mixture of sequential and linear approaches where linear approaches focus on the problem and sequential approaches focus on the solution. Figure 5 shows the overall pattern of how effort is distributed using an iterative approach, resulting in a parallelogram shape where the corners of the parallelogram are adjusted based on the specific project. When all of the sides of the parallelogram "collapse" into a diagonal line, a "pure waterfall" approach results with disciplines are distributed across iterations.

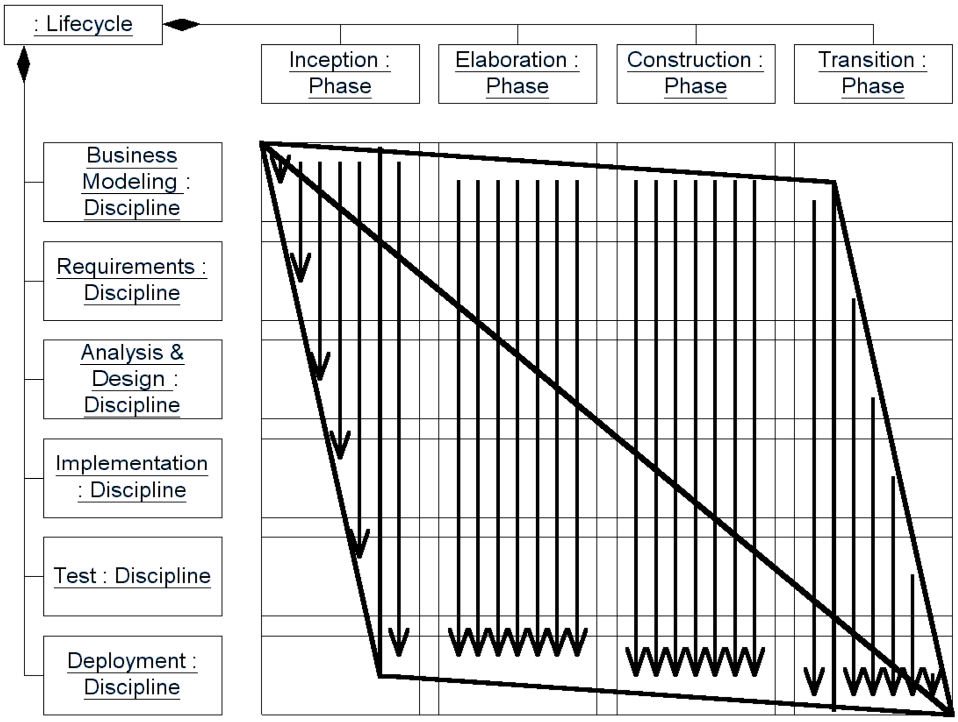


Figure 5: Iterative Approach.

An iterative approach focuses on a stepwise refinement of knowledge throughout the lifecycle. During the Inception phase, linear approaches focus on scope and sequential approaches focus on an architectural proof-of-concept. During the Elaboration phase, linear approaches gain architectural coverage and sequential approaches focus on addressing architectural risk. During the Construction phase, sequential approaches promote deployment opportunities. During the Transition phase, linear and sequential approaches focus on system completion and project closure.

Generally, an effort ramps up at the start of a development cycle, reaches an optimum where all core disciplines are being performed in parallel and all supporting disciplines are operating as appropriate, and then ramps down at the end of the development cycle. As iterations execute, their content involves collaborations among roles, activities, and artifacts where activities are related via a producer-consumer relationship and may overlap in time such that a consumer activity may start as soon as its inputs from producer activities are sufficiently mature.

Effectively and Successfully Applying the Unified Process (UP)

To effectively and successfully apply the UP, we ought to be aware of various guidelines (lessons learned) for applying the process framework.

Given the collaboration among roles, activities, and artifacts, the principal dynamics occur between the roles of the Project Manager, Architect, and Process Engineer. The other roles are not particularly secondary to these roles, but collate around these roles. The Project Manager is responsible for the overall project. The Architect is responsible for the system and its architecture. The Process Engineer is responsible for applying the UP and Development Case. The quintessential factor for effectively and successfully applying the UP is the collaboration and interaction among these roles in the context of a specific project. Their collaboration involves the Architect defining the system, the Process Engineer suggesting the roles, activities, and artifacts required for delivering the system, and the Project Manager applying resources for executing the activities against the artifacts to deliver the system. Their interaction involves leveraging each other's knowledge to successfully execute the effort. These roles must focus on bridging the chasm between culture and vision while balancing various contextual forces in a stepwise approach; that is, they must focus, balance, and iterate to achieve success. Otherwise, linear iterations result if the Project Manager is overly rigid, sequential iterations result if the Architect is overly rigid, and general anarchy results if the Process Engineer is overly rigid. Such rigidity results in compromising and failing to realize the benefits of an iterative approach.

Traditionally, projects have combined the Project Manager and Process Engineer roles, which distorts these principal dynamics and causes a "conflict of interest" amongst these roles (as each role has a distinct focus, objectives, and so forth); thus, increasing the potentiality of project failure. Figure 6 shows the principal dynamics of the UP.

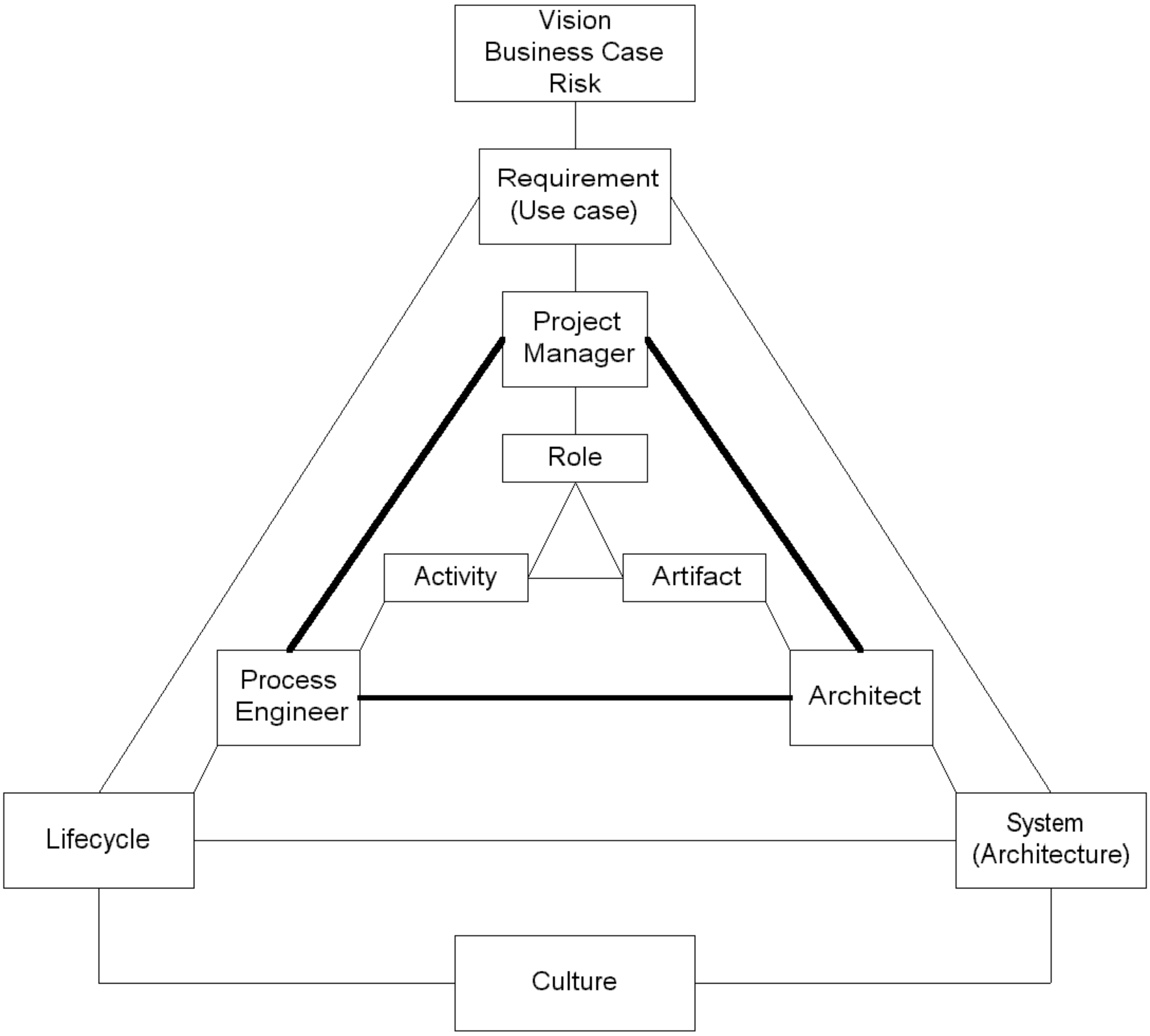


Figure 6: Principal Dynamics of the Unified Process (UP).

While guidelines (lessons learned) concerning specific roles, activities, and artifacts are beyond the scope of this paper, guidance regarding focus, balance, and iterations is provided.

Focus

When applying the UP, we ought to focus and be aware of the following guidelines:

- As everything in the UP is essentially optional, make decisions based on various factors while considering their ramifications. Always ask the "original question" -- Why? Don't do everything specified or suggested by the UP and only do something when there is a reason. A Process Engineer must be able to address why a particular role, activity, or artifact is utilized. Always ask the "original question that may-be" -- What-if? That is, explore what ought to be done. A Process Engineer must be able to address the ramifications if a particular role, activity, or artifact is or is not utilized. Always ask the "original question to-be ('next original question')" -- What-next? Given what-if, what-next (and why)? A Process Engineer must be able to address what particular roles, activities, and artifacts ought to be utilized next. Failure or inability to address these questions indicates or is symptomatic of a lack of focus on the context of a specific project.

- Focus on context, then essential content, and then bridge the chasm between context and content iteratively and incrementally. Without knowledge of the context of a specific project upon which the UP is applied or without knowledge of the essential elements of the UP, the potential of project failure using the UP is heightened. A Process Engineer must be able to bridge the chasm between the project and the UP; that is, apply the essential elements of the UP in the context of the specific project. If the Process Engineer does not have knowledge of the context or essential elements of the UP, they must be able to delegate to those who do have such knowledge and then leverage their input to bridge the chasm. The essential elements of the UP have been emphasized throughout this paper.

- Focus on the "spirit of the UP" and not simply the "letter of the UP." The UP is not loose or chaotic and not rigid or stagnant, but flexible or dynamic. The UP only specifies or suggests, practitioners make decisions and execute. Failure or inability to balance indicates or is symptomatic of being overly focused on the "letter of the UP" rather than the "spirit of the UP." This is a common Achilles heal of many Process Engineers and those applying the UP; that is, they are unable to balance.

- Empower the Project Manager, Architect, and Process Engineer to bridge the chasm between the community's culture and project's vision. When empowered, the localization of forces in-between these roles and the rest of the team significantly heightens the potential for project success because it establishes a context for achieving balance. The Project Manager must be a leader and not simply a project administrator or rigid dictator. The Architect must be a leader and not simply a theoretician or technologist, not overly pedantic or overly pragmatic. The Process Engineer must be a facilitator or enabler and not a process enforcer. The team must be able to stretch to address challenges and seize opportunities, but not break! Each discipline has a role who leads the overall effort within the discipline and who owns and maintains the model ("big picture") associated with the discipline, and each discipline has other roles who own and maintain the details ("small picture") within the model.

Even though many guidelines apply to the Process Engineer specifically, they may apply to other roles. Furthermore, other guidelines may be applied in addition to those above.

Balance

When applying the UP, we ought to be balanced and be aware of the following guidelines:

- Always seek balance; it is not all or nothing without reason and justification! A Process Engineer must consider those roles, activities, and artifacts that necessarily, sufficiently, and consistently address risk and enable project success. Something is necessary if it is required; sufficient if it is enough to satisfy a given purpose; and consistent if it does not contradict or conflict with other things. Consistency may be managed via the use of guidelines. Failure or inability to facilitate necessity, sufficiency, and consistency indicates or is symptomatic of a lack of focus on the essential elements of the UP and understanding the value each process element contributes within the process framework. A Process Engineer who suggests utilizing everything in order to ensure maximum coverage for addressing risk and making sure nothing has been overlooked is impractical and demonstrates this failure and inability!

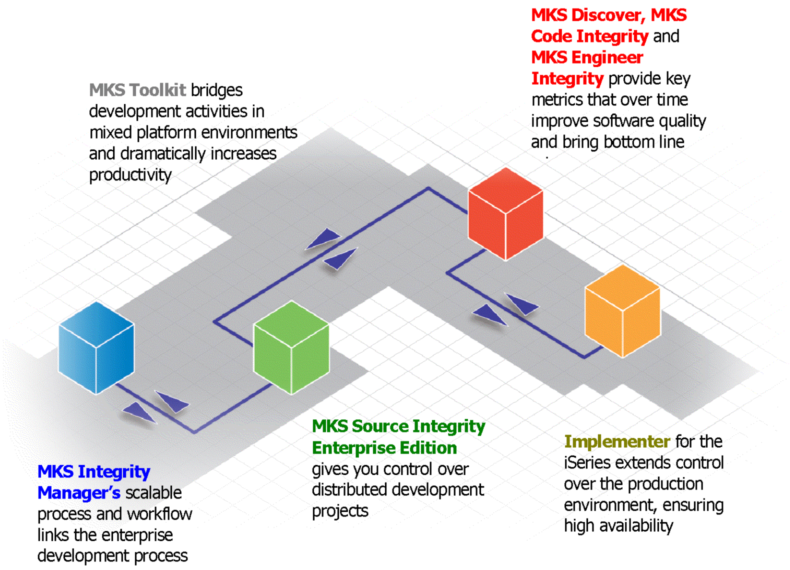
- For roles, it is not typically the case that a project requires all or none of the roles. For roles, it is not typically the case that all or none of the team members are assigned to all or none of the roles. Always ask the "original question" regarding roles!
- For activities, it is not typically the case that a team does all or none of the activities. For the activities that a team does, is not typically the case that the team does them in all their detail instantaneously, but only as sufficiently necessary. Always ask the "original question" regarding activities!
- For artifacts, it is not typically the case that a team produces all or none of the artifacts. For the artifacts that a team produces, is not typically the case that the team produces them in all their detail instantaneously, but only as sufficiently necessary with evolving levels of detail. Always ask the "original question" regarding artifacts!
- For iterations, it is not typically the case that there is constant or no rework or change. For rework or change, it is not typically the case that nothing or everything is reworked or changed. For rework or change, it is not typically the case that such things occur without a reason or for any and every reason. Always ask the "original question" regarding iterations.

mks

Build Better Software™

MKS Solutions Deliver:

- **Lower Total Cost of Ownership**
  More affordable, easier to implement, deploy and administer across the enterprise
- **Flexible Process Strategy**
  Successful implementation and acceptance based on needs of individual, team or project
- **Enterprise scalability and adaptability**
  Java-based, multi-tiered architecture, enterprise DB connectivity, rich integrations



w w w . m k s . c o m

- Beware of a Process Engineer who can "justify" everything without qualification. To the "original question," such a person often replies with "Well ...!" Beware of a Process Engineer who can't "justify" anything. To the "original question," such a person often replies with "Trust me ...!"
- Beware of purists and extremists, those who focus on the "letter of the UP" rather than the "spirit of the UP." Pragmatically, sooner or later, such purists and extremists will be forced to move toward a more balanced middle ground of compromise in order to facilitate project success. Failure or inability to move toward a more balanced middle ground indicates or is symptomatic of a very significant risk to the project. This is a common Achilles heal of many Process Engineers and those applying the UP; that is, they are unable to balance.

Even though many guidelines apply to the Process Engineer specifically, they may apply to other roles. Furthermore, other guidelines may be applied in addition to those above.

Iterate

When applying the UP, we ought to iterate and be aware of the following guidelines:

- Phases provide focus for iterations. Applying iterations outside the context of phases results in the appearance that iterations are loose and chaotic or rigid and stagnant rather than flexible and dynamic. Applying iterations that focus on nothing or everything hinders the ability to demonstrate progress. When planning, executing, and evaluating an iteration for a specific project, consider the phase of the iteration, the specific objectives of the phase, and how to satisfy the objectives within the context of the specific project.
- Iterations are negotiated time-boxes. When an iteration's beginning, end, and content are not negotiated among stakeholders, stakeholders reject ownership of the iteration, thus impacting their contribution and participation within the iteration and project.
- Focus and balance are critical and essential for a successful iteration and project. Without a purpose and objective, stakeholders don't have the ability to prioritize, dialog, negotiate, agree, and attain consensus. Notice that this order of abilities is fundamentally cumulative; that is, without a purpose and objective, one cannot constructively prioritize; without the ability to prioritize, one cannot have a constructive dialog with other stakeholders; without the ability to dialog, one cannot constructively negotiate; without the ability to negotiate, one cannot reach agreement; and without the ability to reach agreement, one cannot reach broad consensus. Beware of stakeholders who lack such abilities!
- Don't "kill" ("castrate") an iteration, unnecessarily! That is, don't prematurely terminate an iteration because this will impact making the ramifications of the iteration and genuine status of the project visible to stakeholders. Only due to catastrophic changes where completing the iteration simply expends resources without providing demonstrable progress should an iteration be castrated. For example, killing the current iteration of a project may be justified if the project's requirements-related or technical assumptions have been significantly invalidated.
- Don't "pollute" ("adulterate") an iteration, unnecessarily! That is, don't modify the scope of an iteration by adding use cases because this will impact making the ramifications of the iteration and genuine status of the project visible to stakeholders. Only due to unanticipated or unplanned requirements that may have catastrophic results if not introduced (in the current iteration) should an iteration be adulterated with these requirements. However, it is preferable that these requirements be fed into future iterations. For example, polluting the current iteration of a project may be justified if the project will lose funding if a specific unanticipated requirement is not introduced in the current iteration.

- Don't "extend" ("mutate") an iteration, unnecessarily! That is, don't extend the end of an iteration in order to accommodate use cases because this will impact making the ramifications of the iteration and genuine status of the project visible to stakeholders. Only due to requirements that may have catastrophic results if not addressed (in the current iteration) should an iteration be mutated to accommodate these requirements. However, it is preferable that these requirements be fed into future iterations. For example, extending the current iteration of a project may be justified if the project will lose funding if a specific requirement is not addressed in the current iteration, thus requiring the current iteration's end to be extended.
- Don't "fester" (or allow the team to "fester"), but progress! That is, use iterations to enable the team to gain a sense of accomplishment.
- Strategize, execute, assess, and be agile (reactively adapt and proactively evolve the team and individual people, process, and tools) across iterations and throughout the lifecycle. Focus, then use, leverage, and exploit assets! That is, given a project's assets (the team and individual people, process, and tools), use them for operational and tactical purposes, use them to gain a strategic advantage, and use them to maximize this advantage. To succeed, localize forces that facilitate success in the context of forces that impede success; and then focus, balance, and iterate to bridge the chasm between culture and vision.

Furthermore, other guidelines may be applied in addition to those above.

Conclusion

Unequivocally, people are the "original ingredient" necessary for success. Don't standardize and enforce the UP, but empower people to leverage the UP! Don't focus on process then projects, but focus on teams because when teams succeed, their projects succeed using their process! Likewise, the UP is "scientifically" defined, but "artistically" applied!

As the Unified Process (UP) is a use-case-driven, architecture-centric, iterative and incremental development process framework that leverages the OMG's UML and SPEM, by understanding the UP, iterations, and being aware of various guidelines (lessons learned), we have a sound foundation for effectively and successfully applying the UP. Furthermore, it is experience, experimentation, and application of the UP and its various elements that will enable us to realize its benefits.

Assessing Readiness for (Software) Process Improvement

Hans Sassenburg , hsassenburg@se-cure.ch,
SE-CURE AG, www.se-cure.ch.

Abstract

From the middle of the eighties onwards there has been an increasing interest in the application of models and standards to support quality assurance in the software industry. In particular, the growing familiarity with the Capability Maturity Model has led in recent years to large scale Software Process Improvement programs. Positive results are being achieved, but the majority of the improvement programs unfortunately die a silent death. Large investments are lost and the motivation of those involved is severely tested. Case studies carried out in various European companies have revealed a number of critical factors, which determine the success or failure of (Software) Process Improvement programs. Instead of evaluating these critical success factors only once at the start of (Software) Process Improvement programs, it is recommended to assess them periodically throughout the entire lead-time of (Software) Process Improvement programs. By regularly determining where weak points exist or may be imminent and by paying attention to these weak points in time, the probability of (S)PI programs succeeding can be substantially increased. Experiences so far in applying this method may be described as encouraging.

SPI Experiences

Working as external consultants, we have acquired a great deal of experience in various European organisations in recent years. In this respect, we have encountered many badly planned implementations of SPI programs, but also some good ones. Three cases from everyday practice are discussed, each containing a description of the following points:

- the establishment of the SPI program;
- the most important factors for the success or failure of the SPI program;
- the extent to which the SPI program will ultimately lead to structural improvements.

Case A: "Solving Today's Problems First"

The first case relates to an internationally operating industrial company. The R&D department has a matrix structure within which multidisciplinary projects are carried out. Over 250 software engineers work in the software department. At the end of 1999, our firm was called in to carry out a CMM assessment. The study was commissioned by the head of the software department but it was found that most of the problems encountered were much more general in nature: unclear and unstable system specifications, no structured project approach and little attention to the quality of the process and the product. Our findings and recommendations were presented to the Board and the management team. The findings were accepted, but we were asked for proof that the recommendations made would rapidly (read: today) lead to success. The report disappeared into a filing cabinet - unused. The failure factors here were:

- **Senior** management did not realise the strategic importance and the added value of software in the product. They were only surprised that software always creates problems and arrives too late.

- The organisation deliberately avoids pursuing a long-term business strategy because people believe it is impossible to forecast how the market will develop in the coming years.
- There is no willingness to invest in structural improvements of business processes: "formalisation will lead to bureaucracy and this hampers the necessary creativity".

The lack of success in starting an improvement program can in this case be attributed to senior management failing in its role.

Case B: "Quick Results"

The second case study concerns a branch of an internationally operating company. Over 200 software engineers work in the R&D department, allocated to projects in which software is by far the predominant discipline. A CMM assessment was carried out in 2000, after which we were asked to offer support in specific improvement areas in the form of consulting and workshops. The success and failure factors were:

- Management appeared to be aware of the importance of software in the various products and released both capacity and money to achieve improvements in accordance with the CMM.
- A separate group - the Software Engineering Process Group - was appointed to co-ordinate the SPI program. Unfortunately, in practice, there was only one person working full-time for the whole R&D department.
- People were not sufficiently involved in determining the bottlenecks and in thinking about improvements. Proposals for improvements were mainly written by external consultants and after a short discussion with a number of key figures were made compulsory for the entire organisation.

Although the organisation is making substantial progress, there may be some doubt about the long-term return on the capital and effort invested in the project. The primary aim is to eliminate all the findings of the assessment as quickly as possible at the lowest possible cost in order to score within the company. The organisation is also seizing every other opportunity to distinguish itself positively in the market and within the company as a whole, so that many people are becoming snowed under with extra activities. Over-working people in this way may prove counter-productive in the long term.

Case C: "Increasing Maturity as Objective"

The third case study concerns an organisation operating independently within a larger company. Various product programs are developed in different groups. In total, there are almost 100 software engineers. Stimulated by strategic statements at the highest level in the company, the organisation started a Software Process Improvement program in co-operation with our consulting. Senior management recognises the added value of software in the various products and creates the preconditions in which structural improvements are possible. In addition to the crucial role of management, there are other success factors:

- A steering committee, consisting of senior management, line management and SPI co-ordinators, has been appointed within which progress is discussed every quarter. The SPI co-ordinators play a facilitatory role and line management reports on progress.
- Formal assessments on a two-yearly basis alternate with more frequent self-assessments, as a result of which the organisation is taught to make strength/weakness analyses itself and to detect and eliminate bottlenecks.

- A number of people are released on a full-time basis to co-ordinate the SPI activities and everyone involved is regularly sent on training courses to acquire the necessary new knowledge.

The active role of management, the organisation around the SPI program, the release and training of people and the extensive provision of information are very clear factors for success. Nevertheless, the continuity of the program is in danger. As yet, the organisation has not succeeded in quantifying goals and results and bringing them into line with the overall business objectives. As a result, people get bogged down in aiming at wrong goals such as "achieving CMM level 2", so that the high investments cannot be justified.

Derived Critical Success Factors

Now, what can be learned from these case studies? It may be concluded that the success of the improvement programs is not guaranteed in any of the three situations discussed. In fact, sustainment is very doubtful. And we believe that the practical situations outlined may be said to be representative of the average situation in many other organisations. It is further evident that the critical success factors may be regarded as important in every improvement program to be started.

Conversely, failure factors must be eliminated. So what exactly are the critical factors for success? A more detail analysis of the case studies discussed, as well as other practical experiences, results in the following overview.

Role of Management

The role of management is crucial in improvement programs. An understanding of the need for improvement must be translated into a clear business strategy from which concrete goals for improvements must be derived. Management undertakes to create the necessary room for investments and plays an active role in the introduction and further implementation of the improvement program.

- Awareness

  Senior and line management are aware of the need for improvement. Strength/weakness analyses have shown what the organisation's position is in the external market as well as the efficiency of the internal management of the business.

- Business Strategy

  The organisation has formulated a clear long-term strategy indicating where it aims to be within a number of years. Concrete, quantified and measurable goals for improvement have been derived from this long-term strategy.

- Commitment

  Management plays a leading, active role in the further development of the business strategy. The necessary space for drawing up, introducing and implementing an improvement program is created so that it is clearly visible for the whole organisation. The management continues to play a leading and active role throughout the implementation.

Project Organisation

Improvement programs do not become successful simply by getting off to a good start. A project team with members drawn from all levels of the organisation must be formed to monitor progress, revise priorities if necessary and perform interim measurements (or have these performed).

- Steering Committee

  The project team, or steering committee, consists of representatives of senior management, line management and the appropriate co-ordinator(s). Where possible, representatives of other parts of the organisation are also involved as listeners or advisers.

- Progress Reviews

  Progress discussions are arranged at regular intervals - e.g. every quarter - by the co-ordinator(s). During these discussions, which are chaired by the senior management, every line manager reports on the progress achieved with respect to the plan, the expected progress in the coming period and problems and risks, which have been identified. Senior management ensures that the quantified goals are actually achieved and adopts a pro-active attitude in solving identified problems and eliminating risks.

- Assessments

  Assessments are made at various points in the improvement process. Formal assessments, carried out by external experts, are used to establish independently how far an organisation has progressed and where the priorities for improvement lie. Self-assessments are made between times so that the organisation learns to perform strength/weakness analyses itself and to identify bottlenecks and take remedial initiatives.

Resource Management

Improvement means change and will encounter resistance. A crucial role is then reserved for the co-ordinator(s) of the improvement program. The 'champions' will be released or will have to be recruited. Another way of thinking and working often results in the need to train people or to provide support by means of tools. Resistance can be removed by involving people as actively as possible in the improvement program.

- Assignments

  The knowledge, experience and skills required by co-ordinators are carefully mapped out. Based on these profiles internal staff are released and/or external consultants are recruited. They are regarded as 'champions' and facilitators who support the organisation in deciding on and implementing improvements: they are therefore not responsible for these. They win respect on the basis of their achievements in the past.

- Training and Tools

  The co-ordinators of improvement programs are confronted with resistance at all levels of the organisation. If necessary, they are trained in change management, together with any other persons involved. The changes in the organisation may possibly lead to changes in the content of the work or to work being distributed differently. Possible training requirements are identified in good time and space is created for following the necessary training courses. The extent to which certain tools can be used to support changes is also looked at.

- Deployment

  All the parties concerned at the various levels of the organisation are actively involved in the improvement program. Everyone participates as far as possible in thinking about possible initiatives, investigating bottlenecks and formulating improved working procedures. The responsibility for these activities is placed at the lowest possible level of the organisation.

Information Sharing

Improvement programs require investments on the part of the management and the motivation of everyone involved. Progress must be regularly reported to all concerned. All other information which contributes to better understanding and motivation should be communicated by means of the available resources. The successes achieved both internally and externally should be brought to everyone's attention.

- Progress Reporting

  The improvement program is launched during a specially convened meeting, attended by everyone involved. Senior management shows its commitment by explaining the need for improvement. These interactive meetings are repeated periodically to report on progress and both senior management and line management are actively present.

- Communication

  Communication media such as notice/bulletin boards and newsletters are available and are known to everyone in order to support the required exchange of information. Their use is encouraged, but care is taken to ensure that the organisation is not swamped with information. Communication is not restricted to one's own organisation: external publicity is deliberately sought.

- Successes

  Demonstrable successes, resulting directly from the improvement program and in line with the overarching business strategy, are regularly achieved. These successes are brought clearly to the organisation's attention. Wherever it appears useful, external organisations are invited to come along and tell their success stories so that the organisation itself can learn from these and people are further motivated.

Method to Periodically Assess the Critical Success Factors

The theorist might possibly conclude that every improvement program to be started should incorporate these success factors as preconditions. The pragmatist, however, will realize that this ideal image can never be achieved and will look to see how this list of success factors can be used as an instrument in successfully starting and sustaining an improvement program. A number of organisations in which we are working has been asked to evaluate this list of success factors every quarter during the progress reviews. By way of preparation for these periodic reviews, all concerned (i.e. members of the steering committee) evaluate the success factors independently of each other by assigning a score. The score is determined by evaluating three separate dimensions (based on the CMM self-assessment method as used in Motorola, see "Achieving higher SEI levels", Michael K. Daskalantonalis, Motorola, IEEE Software, July 1994):

- Approach

  Criteria are the organisation's commitment to and management's support for the success factor, as well as the organisation's ability to implement the success factor.

- Deployment

  Criteria are the breadth and consistency of success factor implementation across the organisation.

- Results

  Criteria are the breadth and consistency of positive results over time and across the organisation.

Guidelines for evaluation are given in Table 1.

The score for each success factor is the average of the three separate scores for each dimension. The scores are collected by the SPI co-ordinator and presented during the Progress Reviews. The scores are discussed and compared with the required status as determined earlier. If the score of a success factor is below this actions are defined for improving the score in the next quarter. Figure 1 gives an example of a practical situation. The Kiviat-plot presents the evaluation results for January 1996.

Score	Evaluation Dimensions		
	Approach	Deployment	Results
0 - Poor	No awareness	None	Ineffective
1 - Weak	Recognition	Limited	Spotty results
2 - Fair	Some commitment	Inconsistent	Intuitive results
3 - Marginal	Support	Deployed	Measurable results
4 - Qualified	Total commitment	Consistent	Positive results
5 - Outstanding	Leadership	Pervasive	Expectations exceeded

Table 1: Score matrix

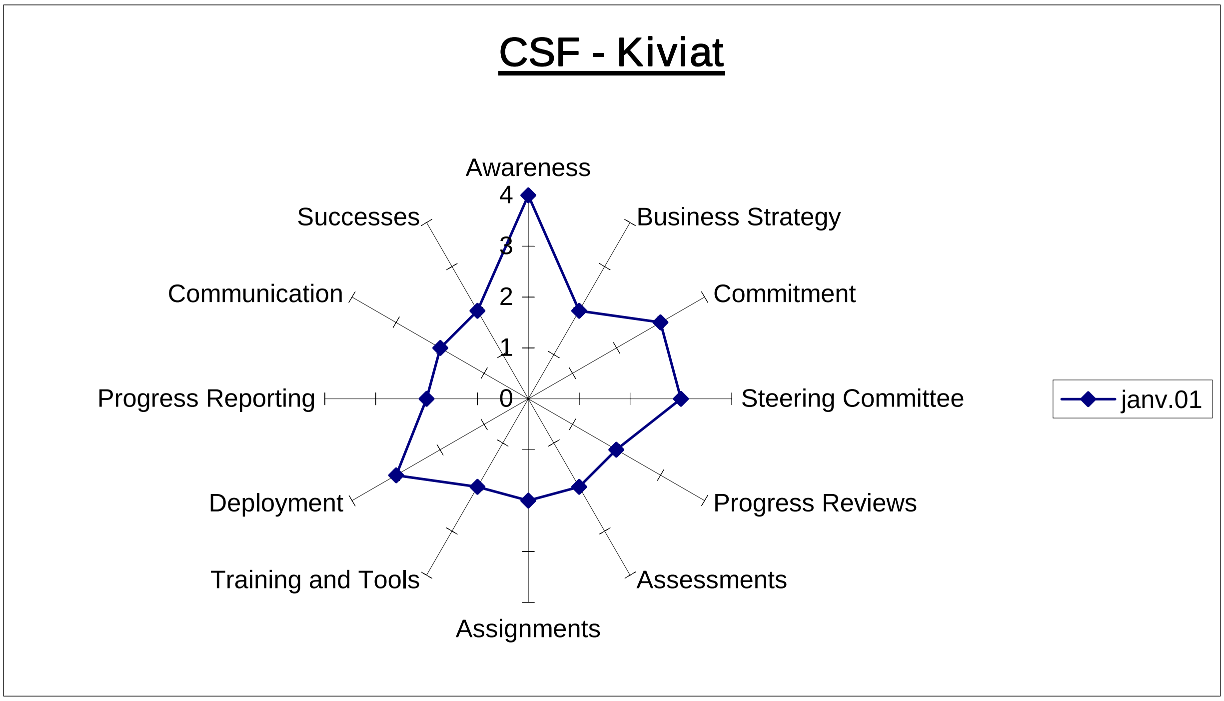


Figure 1: Example evaluation results

Preliminary Results and Experiences

Five organisations were asked to try out the method. Three organisations ultimately agreed to co-operate in doing this. In the other two organisations senior management refused to co-operate: they were not convinced of the possible benefits. The three trials started on January 1995. The provisional results may be described as encouraging:

- Senior management initially appears to be achieving much more positive scores than line management and the people on the shop floor, which is leading to extensive discussions. The result is that everyone has a better understanding of each other's situation.
- In all cases, bottlenecks in the current improvement processes are revealed more quickly. This has led to extra actions, a development which is felt to be both positive and motivating by everyone involved.
- The method has proved to be easy to adapt to the specific wishes of an organisation and is more generally applicable in any randomly selected improvement processes. Success factors can be added, adjusted or omitted as required.

On the basis of the above results all the participating organisations have reported that the probability of the improvement processes ultimately succeeding has increased. This has given us confidence to start developing the method further. It is expected that an evaluation in the near future will result in a definitive set of success factors and a more finely tuned definition.

Website Mapping

Edward Miller, miller@soft.com
Software Research, Inc., 901 Minnesota Street, San Francisco, CA 94107 USA

Introduction

Many properties of important WebSites are only detectable when they are seen by the user from a browser. For example, on a WebSite with dynamically generated pages that are produced from several different servers and final-assembled just before delivery to the client browser, some parts might actually be missing or unavailable. The server may not see this, but the end user -- looking at the generated page from a browser -- certainly sees this as a problem.

Also, how long a page takes to download -- a factor that is affected by server speed, the speed of the Web, and the speed of the connection from the Web to the client browser -- might check out OK from the servers' point of view, but might result in a too-slow page delivered to the user, who then clicks away to another site.

Theses are just two of many reasons that point to the need to have detailed analysis of pages -- or of collections of pages, i.e. whole WebSites -- analyzed from the perspective of a user, i.e. from a browser.

Why Analyze From A Browser?

The main reason for trying to look at WebSites from a browser is that this is the only accurate way to analyze what users actually see. If the WebSite is composed of static pages, i.e. pages that have fixed HTML content and don't vary over time, then server side analyses can be fairly effective. There are very good utilities, e.g. WebXREF, that serve very well in these cases.

But this option disappears in most of the sites where what you see are dynamically generated pages produced by a variety of means based on requests from a user interactively with the WebSite. These account for most of the important WebSites where eCommerce is a main focus. Such WebSites are where simple timing and availability errors have serious consequences.

But the main reason for trying to look at WebSites systematically entirely from the browser is accuracy. A browser-based analysis really does deal with the users' perspective. *What you see is what you get!* If an analysis from a browser turns up a problem there's no ambiguity: it really is a problem!

What Should We Search For?

If you can do searches from a browser [see below], then the natural question to ask is, what do you want to know?

Here are some browser-side questions that having easy and quick and accurate answers to could be very important:

- Do the pages on the site download and render quickly enough?
- Are there broken images on the pages as they are rendered?

- Does the page contain links that go to unavailable resources?
- Are there obvious typos or misspellings on the page?
- Are there JavaScript mistakes on the page?
- How old are the pages?

Note that this list does not include "HTML Errors" because in most cases the browsers overcome most such errors by simply ignoring incorrect HTML. HTML correction is a different topic entirely and, it may be important to emphasize, a high fraction of perfectly OK WebSite pages score very low on the "HTML Perfection" scale!

Site Analysis Requirements

An automated Site Analysis engine has to meet some basic components:

- A way of deciding where a search is to being, how it is to run and when to stop it.
- A way to record (for completeness checking) which pages really were visited after the first page.
- Some way to decide what to do with pages as they are selected.
- A method for reporting what is found when tests made of pages show some problem.

This kind if a search engine based within a browser is actually a kind of spider program because it would start at some point and then create and internal worklist based on what it has just found as it recursively descends through a series of links. But for the goals of automatic WebSite analysis you really don't want the search to drive all over the Web. You really want the search focused like this:

- From a specified starting WebSite pages,
- For all pages that are linked to that page (below that page),
- And continuing until some limit is hit [see below].

In other words, you want your analyses to be constrained and controlled searches of a WebSite or a sub-WebSite, where you can easily control the search criteria. You don't want your search to run forever, but you do want the search to be over a large enough span of pages so that the results of doing the analysis are interesting and valuable.

Site Analysis Engine Architecture

Here are the general requirements on the Site Analysis Engine that support the above goals:

- *Search Size Specification.* From a user specified starting page, i.e. where the browser is currently positioned. To all links on that page out to a specified depth of search. With and without visiting pages that are not part of the current WebSite under test. Less than a specified total maximum number of pages to be visited. Less than a specified maximum amount of search and/or download time.
- *Search Type Specification.* You should be able to specify both the types of links to search, and the types of protocols to search. Link type specifications should be by type of file, e.g. *.HTML or *.phtml or *.jpg, or by type of protocol, e.g. HTTP or HTTPS or FPT.

- *Search Inclusions/Exclusions.* You should be able to provide a list of specific URL to exclude during a search (e.g. you might do this to prevent going to a page that logs your analysis session out after having logged in. You should also be able to indicate URLs that you wish to add to the search tables if they happen to be encountered (e.g. other sub-WebSites).
- *Search Modes.* You should have the option of seeing everything happen in the current browser (this will require it to have focus), or you should have the option to run in the background. (There may be some performance savings possible if less than full analysis is done in background mode analyses.)
- *Cache Control.* With no cache at all, with cache cleared before starting the search, or with cache fully operational.

Reporting

Certain standard reports are always generated by the site analysis process.

SiteMap Report

This SiteMap report is the record of the pages visited that also shows the way in which the search engine came to visit the page. This report is generated as a result of the process of doing the search. Even if there are not filters running the SiteMap report is an important record of what URLs were and were not reached.

Two kinds of reports that are of particular interest:

- Full Report. This reports shows every page (by its URL) and for that page the set of pages that depend on it (i.e. have links on it). There would be no need to show all the details of each page, but for completeness all of the pages below a page would need to be shown.
- Irredundant Report. This reports shows every page (by its URL) and for that page shows only the pages that were actually visited. You could expect that this would be a much smaller list, particularly for a WebSite that has a lot of connectivity.

Filter Reports

The outputs of the filters need to be very simple, easy to read, and generated in real time, as the search is going on (This gives the user the information as soon as possible and prevents generating reports that contain too much information.) Given the ability of the site analysis process to systematically visit every page in a WebSite -- subject to the limits imposed by the user -- it is pretty easy to imagine the kinds of reports it would be interesting to generate from page by page analysis as each page is presented.

- Pages Loading Slower Than Report. This uses the browser timing capability measure the actual download time of the page. Downloading time can be expected to be correlated with page size, but not necessarily. The best use of the data is to include in the final list only those pages that are slower than a threshold, and then to sort these pages by URL in reverse time order.
- Pages Larger Than Report. If a page is too big, it is a candidate for revision. Page size can be expected to be correlated with downloading time, but not necessarily. The report should show pages total bytecount only if the size exceeds a user-specified thresholds. The reported pages should be sorted in decreasing size order.

- Pages Older Than Report. If pages are old, they may represent out of date information. For pages that are dynamically generated, of course, the page age would be essentially zero. But older pages on a site may be a signal that something important may be wrong.
- Broken or Unavailable pages Report. From the browser's point of view a page could be broken or unavailable for a wide range of reasons. It could be, of course, actually missing (a type 404 error). Or it could be from a server that is temporarily unavailable or cannot be reached right now. All pages that the search shows as unavailable should be marked with an indication of what caused the failure.
- Off-site Pages Report. Many WebSites reference a lot of off-WebSite pages, and it may be useful to know which links are to offsite pages. This filter lists them in the order of discovery.
- Pages Patching Search Criteria Report. This may be the most powerful kind of page by page analysis to have available. Every page would be searched, at varying levels of detail, for a match (or a non-match) on a specified string or strings. The report would show the pages that match (or don't match) the search criteria in the order in which the pages are visited.

  One possible application of this feature is as a security check mechanism. To confirm non-manipulation of pages the server could place a particular code on every page in a hidden field. The site analysis engine could search for pages that do NOT have this code -- thus revealing any pages that are not "original".

  The level of detail of the scanning of individual needs to be controlled. At one end of the spectrum you might want to look at everything in the page, including JavaScript and HTML and hidden fields -- everything. On the other hand, you may be concerned only about what the user sees on the page -- the visible text. Another option might be to search only in the META text fields on the page.

© Copyright 2001 by Software Research, Inc.

Doing the right things early in the project life-cycle is the easiest way to get high ROI on an investment in development tools. ANTS <concept> is for use by .NET developers to load testing early in the project life cycle. By simulating up to 100 concurrent users accessing .NET Web services or other .NET applications it is possible to find out whether they scale before it is too late.

http://www.red-gate.com/ants.htm or sales@red-gate.com.

Advertising for a new Web development tool? Looking to recruit software developers? Promoting a conference or a book? Organising software development related training? This space is waiting for you at the price of US $ 20 each line. Reach more than 20'000 Web-savvy software developers and project managers worldwide with a classified advertisement in Methods & Tools. Without counting those that download the issue without being registered! To advertise in this section or to place a page ad simply send an e-mail to franco@martinig.ch

METHODS & TOOLS is published by **Martinig & Associates,** Rue des Marronniers 25,
CH-1800 Vevey, Switzerland Tel. +41 21 922 13 00 Fax +41 21 921 23 53 www.martinig.ch
Editor: Franco Martinig; e-mail franco@martinig.ch ISSN 1023-4918
Free subscription: www.methodsandtools.com
The content of this publication cannot be reproduced without prior written consent of the publisher
Copyright © 2002, Martinig & Associates