

Project Management: Turning Concept into Reality.

Project management bridges the gap between strategy and tactics. It's the difference between having a good idea, and actually being able to execute on that idea.

Wayne Smith

Project management is the ability to transform concept into reality in a no-surprise environment. It is the skill required to take an opportunity (typically expressed as a proposal, business case, or problem statement) and organize and manage a team of relevant talent to reliably deliver a focused, targeted solution that realizes that opportunity.

While this solution can take any form, due to the importance of information technology to most organizations together with its very specialized management challenges, we will focus on the management of technology projects during the remainder of our discussion. By technology projects we intend to include software package acquisitions, new custom implementations, enhancements to existing applications, and related efforts.

In its most basic form, project management is concerned with the definition, representation, construction, validation, and incremental delivery of a set of work products that meets the project's time cost, quality, and customer satisfaction goals.

Deliverables Carry the Value

Since being able to *deliver* is such a critical element of effective project management, its underlying delivery philosophy emerges as an important area of attention. That is, the philosophy that governs how work is organized, planned, and carried out in order to reliably (i.e., at low risk) produce the tangible value that the project goals require.

Accordingly, all projects can be thought of as a collection of deliverables where each deliverable is a completely self-contained unit that contributes a subset of the project's total value. The key idea here is that a deliverable must contain value as defined by the customer, not by the project team or by some internal process. This customer-value focus is vital to ensure that the project concerns itself with only those activities and tasks that add tangible value to the customer. Further, these deliverables can have their own internal structure as well, so long as in each

case any constituent deliverables also meet the same customer value criterion.

Moreover, each deliverable is assigned a deliverable manager who is fully responsible for its definition, construction, validation, and customer satisfaction. The deliverable manager is, in effect, the individual on the project team who is responsible for ensuring that the conceptual integrity of the deliverable is maintained throughout the project's duration. Accordingly, the deliverable manager prepares the work plan (tasks, effort and cost estimates, schedule, assumptions, constraints, and risks) associated with the assigned deliverable, and then manages the actual work against that plan.

Completing Deliverables

Successfully delivering a project to a customer can be viewed as the process of completing each individual deliverable in that project. Therefore, a project is considered complete when all its deliverables have been completed. But, what do we mean by completing a deliverable?

The completion of any given deliverable can be viewed as a process consisting of a sequence of translation steps (see Figure 1), where each step attempts to refine the deliverable content to increasingly lower levels of detail, precision, and clarity until a level is reached that allows its intended value to be realized. That is, the level that allows the deliverable to actually be put to its intended use.

Each of these translation steps comprises two basic activities, *representation* and *validation*.

The representation activity involves the elaboration of the deliverable from its current level of abstraction (understanding) to the next lower level of abstraction. For software deliverables, a typical elaboration sequence is analysis, design, specification, and construction (i.e., programming or coding). In this sequence, the content is being converted from an initial, very high-level description of a business function (general design), through successive refinement steps, until a level is reached (software code) that can be directly executed on a computer. At this point, the representation is in a form where it can deliver its value.

The validation activity ensures that each translation step is complete, error free, and relevant. That is, the meaning of a representation at any level is exactly equivalent to the meaning of the representation at its prior level. For software deliverables, these validation activities include walkthroughs, peer reviews, inspections, and testing.

Project Management: Turning Concept Into Reality

The process of carrying out these translation steps in a simple, rigorous, and reliable manner is the engine that drives the project delivery philosophy. Further, the specifics of this process are dependent on the type of deliverable. For example, the translation steps for a software deliverable are quite different in form and approach from the translation steps required for a non-software deliverable like a study or analytical report.

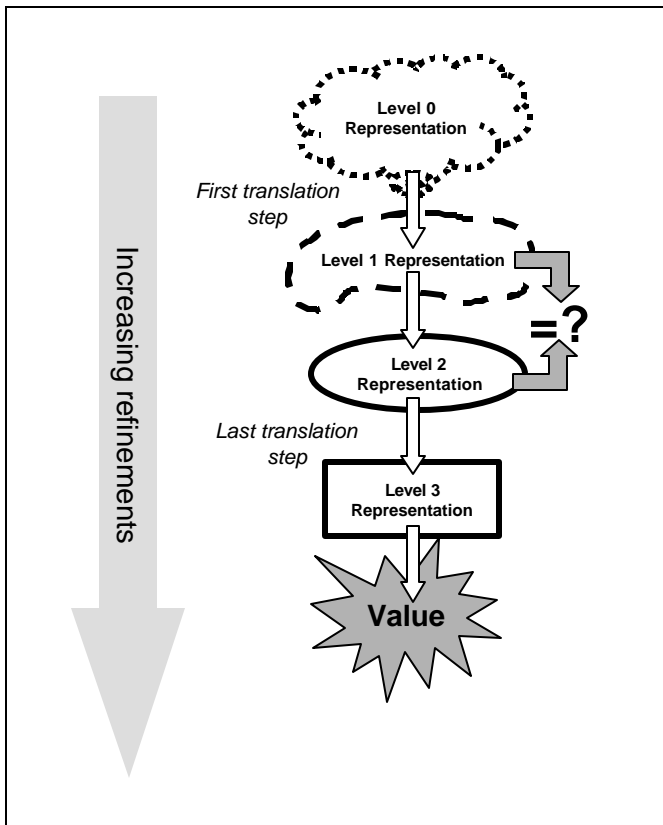


Figure 1. The process of completing a deliverable.

This philosophy and its underlying process are typically expressed in a project's life cycle methodology or software engineering approach. Unfortunately, in many cases, this information is highly fragmented and unfocused which makes it difficult to extract and visualize.

One of the key messages in this discussion is the importance of understanding the dynamics of the delivery philosophy being contemplated for the project, i.e., what do these translation steps look like, how do we know when we have reached the level where value can be realized, what tools and techniques will be employed for the representation and validation activities at each step, what work

products capture the results of the corresponding activities, etc.

Packaging Plan

The packaging plan (Figure 2) is a tool for organizing these deliverables into chunks of customer-usable functionality. Packaging planning is the analysis and preparation of optimal packaging alternatives to reduce risk and increase flexibility in project delivery. Packages are simply containers for allocating delivered functionality and value.

For software deliverables there are four package classes:

- **Modules**, collections of individual software programs, subroutines, tables, and related technology components; modules are the smallest units of packaging in technology projects
- **Buils**, groups of logically related modules that deliver a well-defined subset of the total functionality, i.e., operational sets of business requirements; buils are the smallest units of usable functionality that can be delivered to a customer
- **Releases**, ordered sequence of buils that define a particular production implementation stage or version of the application; releases are the major units of operational implementation
- **Project** itself, the definition of the set of releases that comprise the project's full operational scope

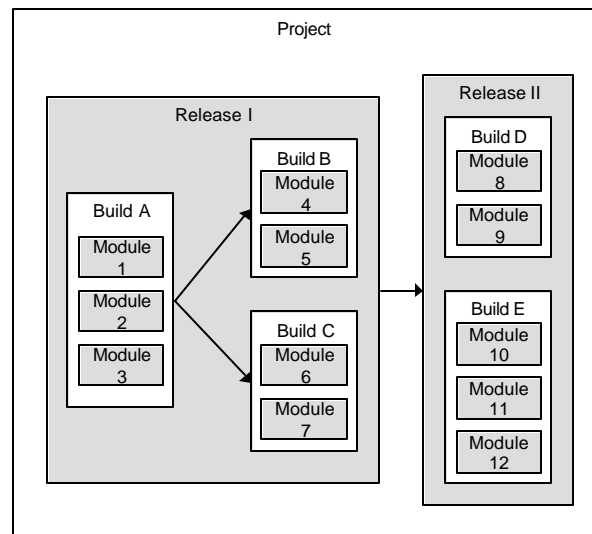


Figure 2. A sample packaging plan for a technology project.

Project Management: Turning Concept Into Reality

The packaging plan shown in the figure illustrates a project comprising two releases (I and II). Release I is defined to be a prerequisite for Release II (as indicated by the directed arrow connecting the two releases). This means that Release I must be completed and validated before work on Release II can begin.

Release I consists of three builds. Build A must be completed and validated before the other builds can be initiated. After Build A has been validated, then work on both Build B and Build C can proceed in parallel.

Finally, Build A comprises three modules that can all be implemented simultaneously. (Any dependencies among modules would be specified in a manner similar to that shown for builds and releases in the example.)

The completion of a project is now simply the completion of its constituent releases. The completion of a release is simply the completion of its constituent builds, etc. The project delivery philosophy governs how these packages (i.e., the specialized *deliverables* of a technology project) are completed.

Moreover, the design and organization of the builds (i.e., the grouping and allocating of specific requirements to a specific build) is a central issue for the packaging plan. Consequently, it is essential to think carefully about the number and composition of builds since this structure directly governs the flexibility, risks, and productivity of the entire implementation.

Each build should be designed so that it exhibits the following characteristics:

- Comprises an integral number of modules, i.e., a module is wholly in one and only one build
- Provides a relatively small, but functionally complete subset of the total solution, in other words a given build might comprise a grouping of requirements that support a single major user function, transaction class, or processing step, or alternatively a sequence of builds might offer successive levels of capability in every area of the proposed solution, or finally, a build might support a certain level of hardware, software, technology, or environmental capability
- Allocates critical high risk modules to early builds so that they can be validated quickly to reduce rework and especially redesign

- Exploits parallelism and the ability to implement builds simultaneously so as to offer the maximum resource loading and scheduling flexibility
- Supports a sequence that optimizes benefit delivery

Builds can also be used as a method for *prototyping* subsets of the solution. For example, a particular build can be allocated a set of requirements considered to be so poorly understood and ill defined that the team can benefit from some initial, but limited, operating experience before the final requirements can be determined. In this case the build mechanism (as an element of the overall packaging plan) is used as a requirements analysis technique to facilitate user understanding and improved feedback.

One final point on the size and scope of a build: Size is important, because ideally a project manager will almost always prefer a greater number of smaller builds, rather than only a few large builds. This is for three reasons:

- Allows builds to move more rapidly through the implementation process (typically, every three to six weeks), which creates a momentum of delivery and success
- Presents a smaller conceptual “footprint” to the customer so that it’s content is easier to understand, embrace, and accept
- Increases delivery sequence flexibility by permitting more granularity for rearranging builds to better accommodate changes in project and user priorities, unplanned scheduling contingencies, delays in other builds, etc.

Software Economics

The economics of technology acquisition can be complex. But, in fact, there are actually only a small number of principles that overwhelmingly drive schedule, effort, and cost.

Consequently, if one learns nothing else about software economics but these three principles, then a project manager will have in their possession the knowledge to dramatically affect the project’s results:

Project Management: Turning Concept Into Reality

- **Assemble rather than develop.** The cheapest, most reliable way to build software is not to build it at all, but rather to assemble and integrate previously validated components. This is a rather obvious point (though seldom exploited) learned from experience in the manufacturing industry. In the software world, this places enormous value on re-use and component libraries. One simple technique to advance this principle is to require that every project either deliver at least, say, 25% of its functionality through reusable components, or alternatively, require that each project contribute new components to the library for subsequent reuse by other projects.
- **Don't modify packaged software.** The economic benefits of packaged software rarely work in your favor if you change the software after you acquire it. In other words, it is much better to change the way you operate (i.e., your internal business processes) to align with the work flows and information relationships defined by the product, than it is to materially change the architecture and structure of the package to fit your way of doing business. Of course, this is not to suggest that one should blindly adopt the business rules contained in your selected product simply because they are there, but rather to point out that where differences are non-material (or, certainly where they represent an improvement in your work flows), then the economic benefits of your package software choice can better be preserved if you limit changes. The trade-offs start to work against you as your changes reach the 20% threshold (i.e., changes affecting 20% of the delivered functionality of the package).
- **Never ship defects downstream.** The longer it takes to actually isolate and identify a problem in any piece of software the exponentially larger are the associated repair costs (Figure 3). In other words, if it costs \$1 to fix a bug during the early planning and analysis stages of a project, it can cost over \$1000 to fix that same problem after the technology is in operational use.

This is a particularly powerful force that directly and aggressively influences the economics of technology delivery.

This important phenomenon is driven by several cost and complexity factors. First, the earlier in any project's life cycle, the numerically fewer objects that, in fact, exist, so defect removal and its associated impacts are limited to substantially fewer elements, thus keeping identification and repair costs low in the early stages. In addition, the earlier in the life cycle, the less complex

these software work products and their relationships with other components tend to be, so that defect identification and removal is far simpler at the beginning than it will be as the effort advances and the structure, architecture, and component interactions become increasingly intertwined and complex.

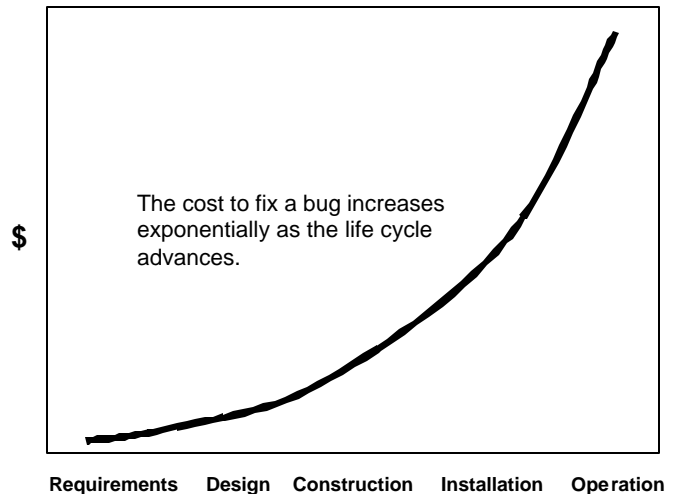


Figure 3. The cost of quality.

Accordingly, as work advances, more components exist whose assumptions and design are based on prior work products, consequently, as the technology effort moves forward it becomes much more likely that defects in one area will also result in waste and re-work in previously completed, and already validated, areas.

Also, the longer the technology acquisition effort, the greater is its overall visibility to the world in general—both internally within the company, and externally among its suppliers and pilot customers—and the correspondingly greater are their expectations and perceptions of its value. Thus, the more expensive any one failure becomes as its effects ripple through that community.

Furthermore, as customers become more involved, which typically occurs with increasing frequency, intensity, and depth as the project advances, the greater the risk of lost business, lost markets, and reduced brand equity due to failures. This substantially increases cost and schedule impacts in a non-linear manner.

Finally, it should be pointed out that fully 60%-70% of all defects that are present in a piece of software are inserted during the early requirements and design stages of the project. Consequently, not only are

Project Management: Turning Concept Into Reality

flaws much cheaper to fix the earlier they are identified, but there are, in fact, a great number of them actually present in those early stages. As a result, it is very seldom economic to ship defects downstream. Thus, by having a project team and process that focuses on preventing defects from being inserted into the technology as well as uncovering flaws early, a project manager can dramatically reduce overall costs.

The Project Repository

Delivering technology solutions typically involves a large number of work products (requirements documents, design materials, test plans, source code, etc.). These work products are complex and highly inter-related. If the project manager does not establish an organizing structure (e.g., a database) for these items, it can be administratively so burdensome that it can materially affect the quality, schedule, and cost of the effort. Further, if this type of organizing structure is not put in place early in the project (when it is often not yet apparent how important it will be), it becomes extremely expensive to retrofit such a mechanism downstream.

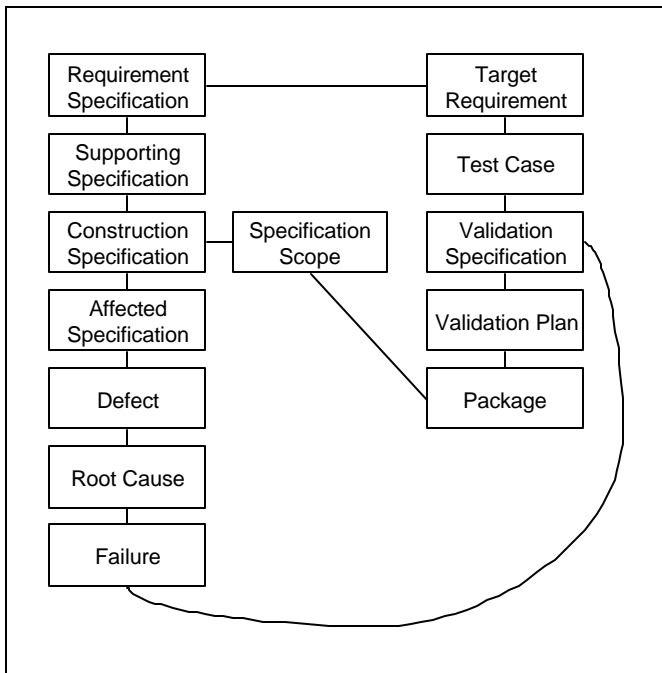


Figure 4. The project repository.

Moreover, the appropriate project repository can become an asset to the project team by simplifying re-use, tracking and controlling revision and modification levels, and providing a reliable source

for all materials that can be used after the solution is operating to dramatically reduce ongoing maintenance and support costs (which costs, typically, can be on the order of four times the original acquisition investment during its first five years of use).

Figure 4 defines the top-level information model for such a project repository illustrating the primary objects and their relationships:

- **Requirement Specification.** Defines what success must look like, i.e., the business, functional, information, process, performance, load, operating, privacy, usability, maintainability, and related needs and constraints of the proposed solution. Each requirement specification consists of a description of a requirement and a set of business rules. For a project to be delivered successfully, its implementation must satisfy all business rules for all requirement specifications. The business rule identifies the conditions that must be satisfied before a given requirement specification can be considered to be successfully implemented. This is used to develop both the supporting specifications that implement that requirement, and the test cases for validating the chosen implementation.
- **Supporting Specification.** Links construction specifications to requirements. Identifies the construction specifications that have been allocated to a given requirement specification for the purpose of identifying the process and information content that is intended to deliver (either fully, or partially) that business requirement.
- **Construction Specification.** Defines the architecture, design, specification, and source code that is the proposed solution. All construction specifications are maintained as a set of abstractions, where each abstraction contains the actual specification content that resulted from the elaboration of that construction specification to that particular level of abstraction. The level of abstraction is the degree of representation detail, where the greater the level, the more refined the representation. In addition, a history is maintained for each abstraction level containing the sequence of modifications that were required to develop that abstraction instance. In other words, the most current modification for all abstractions represents the complete construction specification, with the lower abstraction levels representing the higher-level design material, and the higher abstraction

Project Management: Turning Concept Into Reality

levels containing the actual source code implementation of that design.

- **Affected Specification.** Identifies the specification to which a given defect, and the associated fix, applies.
- **Defect.** A flaw in a specification. Identifies any defects found as the result of a construction specification's validation (i.e., as part of its inspection, walk-through, etc.), or, as the result of failures in package validation plans whose root causes are traced back to a specification. There can be only one specification associated with any given defect.
- **Root Cause.** Root causes link a failure to the defects that generated it. A root cause is the underlying error, mistake, condition, or situation that is the fundamental origin of the failure.
- **Failure.** An instance of actual results not matching expected results in a validation plan. The execution of a validation plan can give rise to any number of failures. Each failure typically results from one or more very logically related root causes. Each root cause leads to a series of defects, where each defect identifies the underlying flaw in a specification that is the original source (or contributing source) of the failure.
- **Specification Scope.** Identifies the specifications that comprise this package's functional scope. This establishes the analytical focus of the package. In other words, when a package is implemented, the specification scope defines the full operational value delivered by this particular package.
- **Target Requirement.** Identifies which requirements a given test case will validate.
- **Test Case.** The specification of a condition and the associated expected results for the purposes of validating one or more target requirements. The test case is the smallest validation unit.
- **Validation Specification.** The class of inspection plans, unit test plans, integration test plans, and acceptance test plans. The validation specification is composed of test cases organized into logical groupings called test runs.
- **Validation Plan.** The collection of validation specifications that are used to validate a package.

- **Package.** A class of software deliverables suitable for implementation. More than one package can be implemented at any one time. The implementation of a project consists of the implementation of all its constituent packages.

The project *infrastructure* (i.e., the collection of technologies, data bases, toolkits, and best practices that make up the environment in which the project will operate) provides the facility for the actual development, maintenance, and use of all objects in the project repository. Consequently, the precise form and shape of each object's content is a function of that particular environment.

Project Plan

The project plan represents the contract or agreement between the project manager and the customer or sponsor. This agreement must clarify the nature of what is being delivered and on what basis, and under what set of assumptions.

An effective project plan contains the following elements:

- A detailed description of the problem to be solved, the overall solution approach, and its organizational and business context
- A definition of the project goals, i.e., the conditions which define project success; these goals (see Project Feasibility) must be as objective and as quantitative as possible, and should specify both the values for these goals before the project starts as well as the targeted values that must be achieved after delivery (the gap between the two sets of values is one method for defining the project's benefits)
- The project organization with duties and responsibilities defined for project, user, and customer personnel together with a staffing plan that describes how and when these individuals will join and depart the project
- A definition of the project infrastructure, i.e., the tools, methods, processes, and administrative procedures that will be used to support the project team and facilitate the carrying out of their assigned tasks (software engineering, modeling, status reporting, configuration management, quality, defect and failure reporting, change control, validation, customer acceptance, library management, etc.)
- The packaging plan (or, deliverable definitions for a non-software project)

Project Management: Turning Concept Into Reality

- An overall schedule and milestone chart for the entire effort
- A detailed work breakdown structure (WBS) for the near-term horizon of the project (the typical useful horizon is 2-3 months) which facilitates a more reliable effort, schedule, and cost estimate
- A high level WBS for the remainder of the project with correspondingly decreasing levels of detail and reliability the farther out the planning horizon
- Assumptions upon which the WBS, estimates, and overall approach are based; changes in the assumptions will drive changes in the plan

The WBS should always be organized by package (for non technology projects, this means by deliverable), with each package manager responsible for the corresponding WBS needed to implement that package. Since this portion of the plan is tightly focused on only the production of that package, the set of tasks that make up its WBS is typically very simple and straightforward.

In addition, and this is particularly important, the project manager must be able to quickly re-plan the remaining effort based on the team's evolving understanding of the key project productivity factors (see the discussion on performance capacity metrics in the section on Sizing) so as to continually improve the reliability of the estimates, and to feedback the effects of various changes in project assumptions to gain a more accurate picture of what is really possible to deliver and when.

The project plan is the primary (and living) document defining the project, the problem it is attempting to solve, the approach for implementing the required business solution, and the methodology and processes for carrying it out successfully.

Sizing

No part of technology efforts tends to be more elusive than determining their true scope. There are five primary sizing variables for technology projects. These sizing variables are organized into two groups, base variables and derived variables.

There are two base variables: **mass** (expressed in units of lines of executable source code, function points, source operators and operands, requirements, packages, etc.), and **risk** (expressed in dimensionless units and treated as a proportionality function for scaling each derived variable). These two variables are referred to as base variables because they represent inherent attributes of the solution and its relevant delivery environment.

The mass variable is an attempt to establish the intrinsic magnitude of the solution itself. It is a measure of the information and functionality content of the solution. Solutions that have greater functionality are, in effect, more massive, than solutions with less inherent content. That is, they intuitively require more power to "move" them from one state to another, or to change implementation direction or tempo.

The risk variable (see section on Risk Management) is a measure of the complexity of the solution and the reliability of the delivery environment and its relevance and fit with the solution being contemplated. In other words, risk is an indication of how well suited the overall environment (e.g., people, skills, process, tools, organization, etc.) is to the solution at hand. The more suited, the lower the risk; the less suited, the greater the risk.

Mass and risk are critical variables for a project manager to deeply understand because they drive the computation of the values for the three derived variables: **effort** (expressed in units of person hours, person years), **time** (units of calendar days, weeks), and **cost** (expressed in \$).

Moreover, the relationship among these five variables for any given project is decidedly nonlinear, thus dramatically complicating their use in project sizing, and in understanding how changes in one variable affect the values of the other variables. For example, several researchers (Walston and Felix of IBM, and Boehm of TRW) have found that the following equation fits many of the relevant software development models that they studied:

$$\text{effort} = k \cdot \text{mass}^f$$

Where effort is expressed in person months, mass is expressed in lines of code, and k and f are empirical factors (similar to our risk variable) ranging from $k=5.2$ and $f=.91$ for the IBM model to $k=3$ and $f=1.12$ for one version of the TRW model. Ignoring for a moment the very real counting issues related to the line-of-code choice of the mass unit (e.g., it is language dependent, etc.), one can easily see that sizing the lines of code for a solution that is still in the early planning and design stages is a daunting task with enormous variability potential.

Fred Brooks in his excellent collection of software engineering essays (*The Mythical Man-Month*, Addison-Wesley, 1982) illustrates another aspect of this non-linearity by pointing out that "men" (i.e., effort) and months (i.e., time) are not interchangeable. Many software development activities share this property: Adding effort does not necessarily result in a corresponding decrease in

Project Management: Turning Concept Into Reality

schedule. His aphorism that “*It takes a woman nine months to have a baby, no matter how many men are assigned to the job.*” is a particularly poetic way of describing this phenomenon.

Brooks goes on to summarize this nonlinear relationship between effort and time with one of his most famous quotes: “*Adding more people to a late software project makes it later.*” In this case, the increase in time (rather than the decrease or preservation of the current schedule that was intended) is due to the substantial increase in interaction and learning curve effects as each new team member enters the project, especially late in the effort where team size is largest. These effects tend to be roughly proportional to n^2 , where n is team size (actually, they are proportional to $n^{(2)}/2 = (n \cdot (n-1))/2$). In other words, if additional people will be needed, then the earlier they are added, the better. Many project managers have ignored this observation at their considerable peril.

There is an entire industry segment devoted to tools and techniques for assisting project managers in computing these five variables for a given project. While these tools can be very useful, the most reliable sizing methods are those based on direct, quantitative experience in the exact environment, and with the same talent, as the proposed effort.

Precise matches of this type are typically impractical, so proxies must be used. The most reliable proxies are quantitative metrics captured by the organization managing the project. These metrics, in effect, characterize the *performance capacity* of a particular software development environment:

- **Productivity** (mass/effort, mass/cost) is a measure of the efficiency with which usable functionality can be produced; unit costs are simply the inverse of the corresponding productivity metric
- **Cycle Time** (time/mass) is a measure of the environment’s latency or the elapsed time between deliveries; the delivery velocity is the inverse of cycle time

In addition to these primary metrics, there is a set of secondary metrics that is exceedingly useful in better understanding the operational characteristics of the environment and the completion status of the project:

- **Cost-To-Complete** (*Rem To Do*, see Progress) is a measure of work remaining; the change in this value per unit of applied effort is a useful indicator of the efficiency of the project

- **Earned Value** (validated mass/total mass) is a measure of completeness, where validated mass refers to those components of the solution that have met a predefined quality threshold; due to its emphasis on delivered value from a customer’s point of view rather than simply effort expended, this can be a more insightful metric into actual progress
- **Variance Volatility** (rate of change of current period variance, see *Var This Period* in Progress) is a measure of the reliability of the project plan and its estimates
- **Defect Density** (defects/mass) is a measure of the process quality of the packages that make up the technology solution; defects tend to cluster, so that the likelihood of finding the second defect in any given section of source code, for example, increases
- **Failure Intensity** (failures/time) is a measure of the efficacy of the validation process; since the goal of testing is to find problems, not prove something works, this metric can be a useful window into the effectiveness of the test case planning and design process
- **Cumulative Validation Rate** (% failure-free test cases) is a measure of the completeness of the validation process; this metric exhibits the so called S-curve shape (Figure 5), and can be a very helpful tool to understand how much testing remains
- **Design Coverage** (requirement specification \times module) is a measure of the degree to which the requirements are supported by the functionality of the proposed solution; this metric (and its underlying correlation table) can be very helpful during the early stages of package elaboration in identifying design gaps, over-designed solutions, and related architecture problems before substantial effort has been spent on constructing the solution
- **Validation Coverage** (test case \times requirement specification) is a measure of the degree to which the requirements are being validated by the test plans; this metric (and its underlying correlation table) is essential in effective test plan and test case design since it ensures that the minimum number of test cases have been devised that will validate the maximum number of requirements

An effective project manager recognizes the importance of quantitative data in understanding the dynamics of the project and in using this fact base to

Project Management: Turning Concept Into Reality

learn, stabilize, and continuously improve the performance capacity of the project environment.

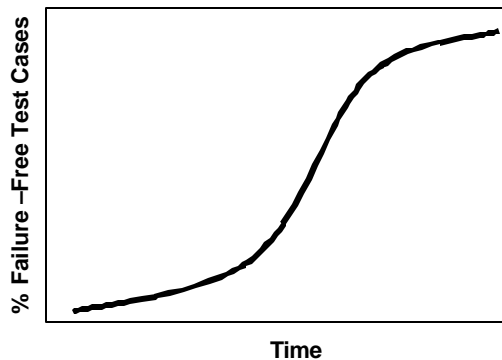


Figure 5. When is testing done?

The Life Cycle Question

There has always been a great deal of discussion and confusion regarding the question of which development life cycle, methodology, or software engineering process to use on a given project. Over the years these life-cycle approaches have been known under various names, such as waterfall, structured design, information engineering, rapid application development, spiral, prototyping, etc. They all have value, and in the right hands, can lead to successful implementations. In fact, it turns out that the more important issue is not which process to use, but rather, once chosen, that the project manager has the experience and discipline to rigorously apply it in the proper way.

When examined closely, however, the primary difference among all these techniques (besides vocabulary) lies simply in how they distribute the representation and validation activities (as outlined in Figure 1) across the project's deliverables. Consequently, there are three key questions that can help a project manager determine the best approach for any given situation:

- How flexible must the customer delivery sequence be?
- How much of the design and specification do you want to complete before any software is developed?
- How important is it for multiple teams (or individuals) to be simultaneously working on separate components of the solution?

A process definition table (Figure 6) can be used to help the project manager answer these questions. This table describes how and when the various objects in the project repository are produced. The first column defines the standard WBS (or list of tasks) that is needed to implement a given package. The second column (*Pri*) indicates the sequence and degree of simultaneity that is desired among these tasks. For example, the values shown indicate that the *Plan* and *Construct* tasks should be carried out in sequence (since they have values 1 and 2 respectively), then followed by the *Cover* and *Implement* tasks carried out in parallel (they both have a value of 3), and then finally after both of those are done, the *Validate* task (with a value of 4) is executed. The remaining columns specify the primary repository objects that are produced for each task-package combination.

Finally, the last row, the *target abstraction level*, indicates the lowest level of refinement and representation that a given package will undergo during its elaboration (i.e., during its *Construct* task). The target abstraction level is one of the key customization choices for the project manager. This choice reflects the particular needs of the project as well as the capabilities (and vocabulary) of the project's underlying software engineering toolset that will be used to produce the objects in the repository.

The process definition table together with the project repository information model defines the development life cycle for the effort. The repository information model specifies all the objects that are important to the project while the process definition table indicates how those objects are created, maintained, and used.

Since the packaging plan defines all the value content of the project, the implementation of a project, and thus the delivery of its intended value, can now be viewed as simply the implementation of its constituent packages.

The question then becomes what does it mean to *implement* a package?

The implementation of any package (at any level), as defined in the process definition table, consists of carrying out the following five tasks for that package:

- **Plan.** The development of the management and organizational environment that supports this package's implementation. This includes the elaboration of the project plan (see Project Plan), building the team, and the establishment of the project repository and infrastructure that will be used to produce and house the project deliverables and repository objects.

Project Management: Turning Concept Into Reality

- Construct.** The elaboration and validation of the requirement and construction specifications that comprise the package's specification scope. This is the primary construction action for a project and consists of analysis, design, specification, and coding activities. The target abstraction level controls the degree of refinement detail that is to be carried out by this task for each package. In other words, at the project level the *Construct* task involves the development of the overall design and architecture for the entire proposed solution, while at the module level this same task involves developing detailed component designs and source code.
- Cover.** The elaboration of the validation specifications that comprise the package's validation plan. This is the primary validation planning action for a project and consists of analysis, design, test planning, and test case specification activities.
- Implement (a constituent package).** The implementation of a constituent package. There is one of these tasks for each subordinate package that was defined as a constituent for this package in the associated packaging plan. The dependencies defined in the packaging plan are used to establish the desired relationships among these tasks.
- Validate.** The validation of the package using the validation plan produced by the earlier *Cover* task. This is the primary validation action for a project and consists of dynamically creating the validation environment (the transient workspace for a package that contains the executable software code and its corresponding validation execution results), carrying out the test runs using the predefined test cases, comparing actual results with expected results, analyzing failures, identifying root causes, and recording the associated defects.

The implementation of any package (at any level) is always the same: develop any plans that are relevant for that package, carry out the analysis, design, and coding activities as controlled by the target abstraction level, develop the test plans that will be used later to validate it, implement any constituent packages, and then finally, after all constituent packages have been implemented (and therefore are valid), execute the pre-defined test plans to ensure that this package meets its requirements.

At this point the package is considered complete and valid, in other words, it is considered implemented.

	Pri	Project	Release	Build	Module
Plan	1	Infrastructure, Project Plan, Package, Specification Scope	Project Plan, Package, Specification Scope	Optional	Optional
Construct	2	Requirement Specification, Supporting Specification, Construction Specification, Affected Specification, Root Cause, Defect	Requirement Specification, Supporting Specification, Construction Specification, Affected Specification, Root Cause, Defect	Construction Specification, Affected Specification, Root Cause, Defect	Construction Specification, Source Code, Affected Specification, Root Cause, Defect
Cover	3	Validation Plan, Integration Test Plan, Acceptance Test Plan, Test Case, Target Requirement	Validation Plan, Integration Test Plan, Acceptance Test Plan, Test Case, Target Requirement	Validation Plan, Integration Test Plan, Acceptance Test Plan, Test Case, Target Requirement	Validation Plan, Unit Test Plan, Test Case, Target Requirement
Implement	3	Release	Build	Module	None
Validate	4	Actual results, Failure	Actual results, Failure	Actual results, Failure	Actual results, Failure
Target Abstraction Level		Overall design, Architecture	General design	Detail design	Component design, Source code

Figure 6. Process definition table.

In summary, then, the WBS for any project is developed in the following manner:

1. The five tasks are created for the highest-level package, the *project*.
2. Use the *Pri* column to establish the sequence and degree of simultaneity for each of these tasks in the plan.
3. Establish one *Implement* task for each *release* using the packaging plan to identify the constituent releases and their sequence and dependencies.
4. Develop effort and calendar estimates for each task. Note that the assigned task owner, that is the individual responsible for actually carrying out the work, should create all effort and schedule estimates.

At this point we now have a complete (but, very high level) plan for the entire project. Figure 7 shows a project WBS based on the sample packaging plan illustrated earlier in Figure 2.

Project Management: Turning Concept Into Reality

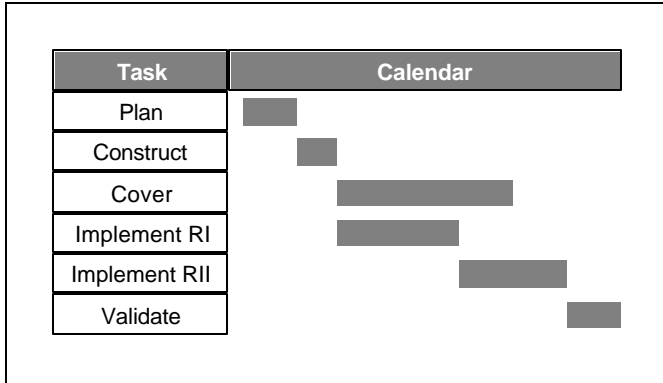


Figure 7. A project WBS.

The top-down process for developing the full WBS continues with the development of the WBS for each release. That is, steps 1-4 are now carried out for each *Implement* release task defined in step 3.

Figure 8 shows the results of carrying out these four steps to produce the WBS for Release I in our example. This subordinate WBS provides the clarifying detail for the associated summary task in the parent WBS. Figure 9 depicts the hierarchical relationship between the *Implement* summary task at one level and the corresponding detail WBS at the level below that defines the implementation of the associated package.

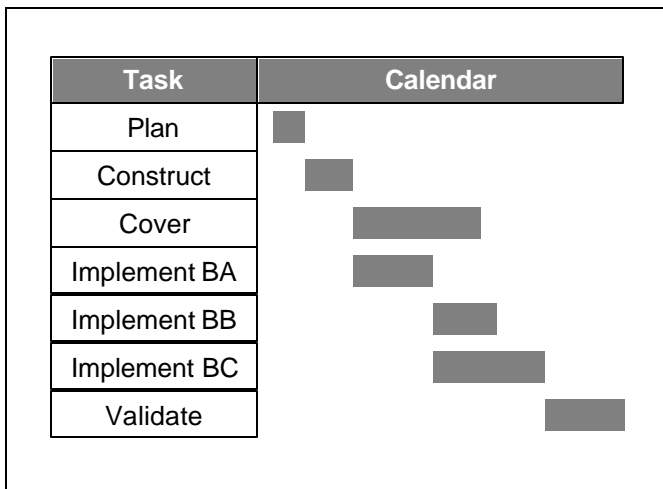


Figure 8. A release WBS for Release I.

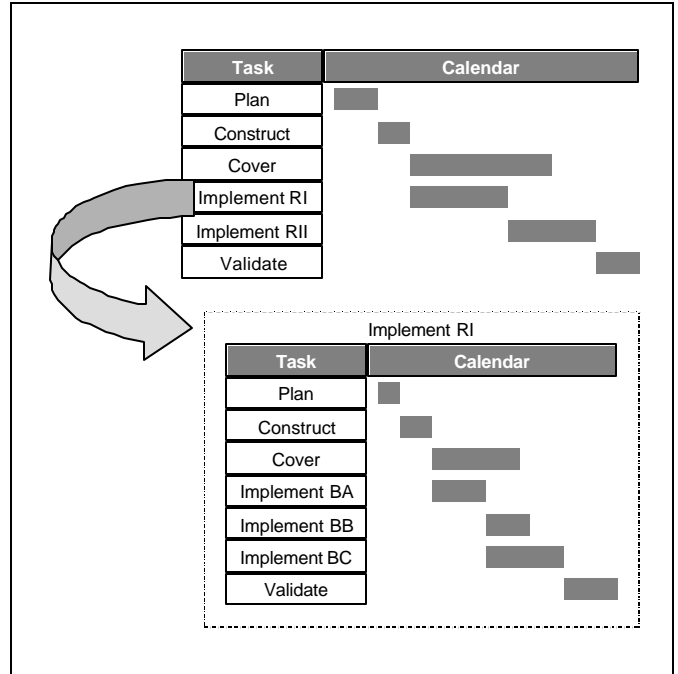


Figure 9. Each *Implement* task summarizes its constituent WBS.

Steps 1-4 are then repeated for each build (Figure 10), and finally down to the module level (Figure 11), where, as the process definition table indicates, the process stops since a module has no constituent packages. At this point a complete detailed WBS for the entire project has been created.

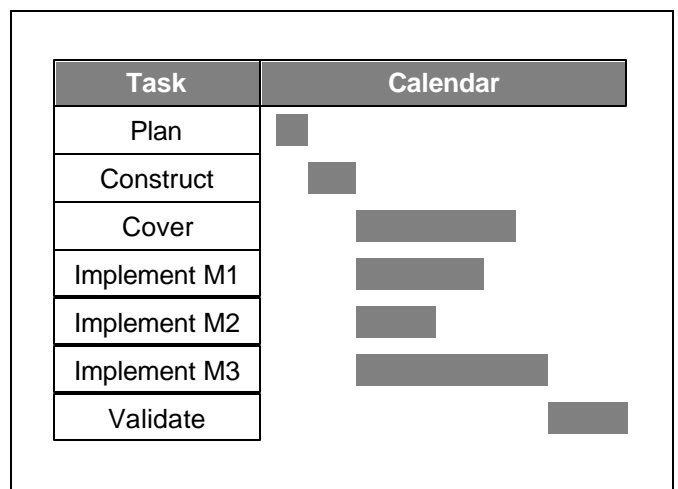


Figure 10. A build WBS for Build A.

To improve reliability and to simplify the process, the project manager should focus on developing the WBS down to the module level only for those releases and builds that are scheduled for

Project Management: Turning Concept Into Reality

implementation in the near term (typically, over the next two to three months), while leaving the WBS at the higher levels for the packages targeted for periods beyond that. This allows a moving window of planning detail to be used that always keeps the team focused on the work at hand, while still providing an overall perspective on the total effort. Furthermore, this makes it easier to reflect any learning derived from implementing earlier packages into the sizing and schedule estimates of future packages, thus increasing the plan's reliability.

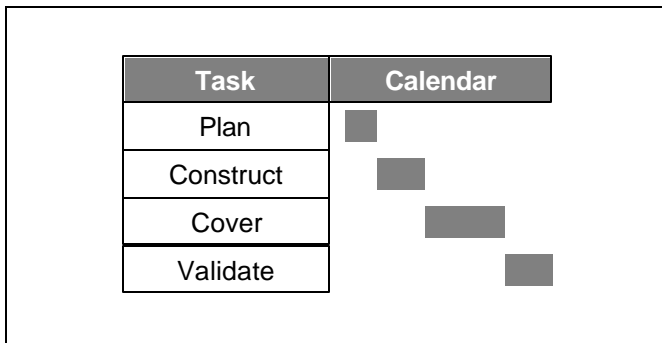


Figure 11. A module WBS.

This top-down development of the total WBS for a project by following the structure and relationships specified in the process definition table and the packaging plan significantly contributes to reducing project risk:

- Allows project manager to always be developing a plan that comprises the entire project scope
- A top-down approach reduces the risk that tasks are left out, which is one of the primary reasons for under-estimating the cost and schedule
- Once the WBS is refined down to the module level (which, by definition, are small), the bottom-up estimating technique becomes much more reliable since it is always easier to estimate small chunks of work rather than larger, more complex pieces
- The subordinate WBS for each constituent package becomes a natural mechanism for allocating these to package managers who can then, as needed, provide estimates and resources for their package

The life-cycle approach introduced here and characterized by the repository information model and the values indicated in the process definition table defines a **preferred method** for implementing all technology projects whether they are small or

large, custom or package installation, or via a central team or through a disparate set of dispersed talent. This preferred method has been field-proven over literally hundreds of technology efforts and has been shown to result in the fastest, lowest cost, highest quality, and lowest risk technology delivery.

Managing Expectations

The ability to quickly capture, at the required level of detail and specificity, the business problem with its associated requirements in terms the customer can understand, and then to continuously validate that understanding is absolutely vital for success.

In carrying this out it is important to recognize that in addition to the written requirements, there are also the often un-captured, but nevertheless very real, expectations of the actual users and sponsors of the system. For an implementation to be considered successful, these expectations must be flushed out and managed just as rigorously as the requirements. History is replete with examples of implementations that have satisfied the written requirements but have failed nonetheless because the resultant system has not met the expectations of the end users or customers.

In fact, in many cases, the primary risk item for any given project will not lie in the technical areas at all, or even in the applications, but rather in the misunderstandings on the part of the various constituencies as to the real scope of the effort, what is intended to be solved and when, and on the implications of the implemented solutions to their current activities.

The approaches outlined in this discussion have been expressly designed to address both of these crucial drivers of project success. Indeed, all the processes described here have been designed in such a way that they not only generate the appropriate work products in a simple, efficient, and organized manner, but also attempt to uncover missing or poorly-understood requirements, as well as close those critical gaps between what customers expect and what they actually get.

In particular, the process of linking each requirement to the corresponding portions of the solution that are intended to fully or partially support that requirement (see the *supporting specification* in Figure 4 and the *design coverage* metric in the section on Sizing) helps identify design gaps and ambiguous or illogical business rules before they result in costly rework and delays downstream.

Project Management: Turning Concept Into Reality

Moreover, the package implementation approach with its emphasis on the preparation of a given package's validation plan (both for integration and acceptance testing) while its lower level constituent packages are being implemented, tends to quickly flush out expectation gaps and missing requirements before the corresponding software is actually developed. In other words, this validation planning approach (with its requirement for pre-defined test cases and expected results) is actually validating the requirements and their supporting design as a by-product of developing the test plans.

For example, as shown in Figure 7, the preparation of the integration test plan and the acceptance test plan for the entire project is carried out in parallel with the implementation of the project's two releases. This means that while the team responsible for implementing these releases and their constituent builds and modules is carrying out its work, the independent testing role and the users are preparing the plans for validating those packages. This parallel set of activities from two completely different perspectives, one focused on building the solution, and the other focused on finding defects in that solution, has been shown to be very efficient in not only finding bugs early (i.e., in many cases before there is even any software to test), but in rapidly exposing those critical gaps in expectations.

Finally, these ideas and approaches add value regardless of the type of project. Even in a package installation (where very little new development or modification is being planned) this approach can be a very cost effective way to ensure that the purchased products will, in fact, support the business needs before the product is installed and major implementation costs and license fees are incurred.

Requirements Driven Validation

Since validation activities typically consume 50% of the total project effort, it is important to have a carefully engineered validation approach. The unit test plans, integration test plans, and acceptance test plans comprise the full set of test runs, test cases, and expected results that form the primary validation plans of the project. These test plans are derived from both the requirements material and the packaging plans, as well as details emerging from the design efforts.

The goal of this planning is to ensure that all requirements will be completely validated.

Validation coverage, i.e., the process of linking each requirement to its corresponding test cases and expected results (see *target requirement* in Figure 4

and the *validation coverage* metric in the section on Sizing), is an important part of this process. This technique examines each requirement individually and asks the following simple, but very powerful question: Can the test planner visualize a specific test case and expected result that when executed will, in fact, validate this requirement? In other words, when the test case executes and the actual results match the expected results, can this requirement be considered implemented?

If the answer is no, this usually indicates that the requirement is too high-level or ambiguous and will require further elaboration and clarification before proceeding. And, of course, if the testers are unable to develop a test case from this requirement, then it probably implies that the requirement would have been insufficient to effectively guide the developers as well.

By using this method, test planners typically uncover missing requirements, illogical requirements, and other gaps in understanding that substantially impact the cost and schedule of the implementation.

An effective test planning approach comprises three levels of testing. Each level is carried out by a different project role, and applies to specific package classes:

- Unit testing, carried out by the developers as part of module implementation, is used to validate the specific functionality and performance of a single module
- Integration testing, carried out ideally by a separate independent test team or, at least, by someone other than the developer of its constituent modules, is used to validate each build, release, and the entire project
- Acceptance testing, carried out by the user or knowledge worker, is used to verify customer approval of each build, release, and the project as a whole

Unit test plans are developed in conjunction with their associated modules and, consequently, their test cases are primarily designed based on detailed knowledge of the underlying logic and structure of the module's constituent functionality. This is known as *white box testing*, since it relies heavily on internal knowledge of the module.

Integration and acceptance test plans, however, are requirements driven. That is, their test cases and expected results are developed not based on the internal logic and structure of the package, but rather on the requirements that these packages must deliver. This technique is known as *black box testing*.

Project Management: Turning Concept Into Reality

This validation approach requires that there be at least one test case for each defined requirement, and, further, that the goal of any given test case is to uncover as many defects as possible, not to prove something works. Also, these test runs, test cases, and expected results are catalogued in a test library to facilitate regression testing during implementation and to support the ongoing maintenance activities after the solution is in full production operation.

Note, that the goal of the acceptance test plan is not so much to uncover problems in the technology—the integration test plan is designed to rigorously flush out those types of problems—but, rather, to uncover gaps in expectations and misunderstandings regarding assumptions, vocabulary, etc. This is why it is important that this be a user-developed deliverable.

This failure driven mind-set, together with the parallel development of the integration and acceptance test plans with the construction of the corresponding constituent packages, has been demonstrated to force out defects, misunderstandings, and expectation gaps very early in the process where, as we have seen, they are exponentially cheaper to fix.

Consequently, this entire test planning process tends to not only generate the needed test plans, but also to actually validate the solution requirements well before any software is actually developed.

Continuous Integration

One of the central features of the preferred method described here is its reliance on continuous integration. Continuous integration is a powerful iterative process of *incremental delivery* in which the entire solution is constructed, validated, and accepted package by package:

- Each module is specified, constructed, and unit tested independently by the developers against its individual specifications, defects are isolated and corrected, and the module retested until defect free
- The defect free modules are promoted to the integration testing role (an independent test team) which assembles the modules into their predefined builds (proscribed in the packaging plan) and then integrates the newly promoted build (say, build n) into the evolving solution consisting of all prior defect-free builds (builds 1 through n-1)
- The independent test team executes the integration test plan for that build against the newly integrated system (now consisting of builds 1 through n)

- This integration testing consists of executing the catalogued test runs and individual test cases, and then comparing the actual results against the pre-defined expected results in order to identify any failures (i.e., instances of a discrepancy between actual and expected results)
- For each such discrepancy instance, a failure report is prepared and forwarded to the development or support team for root cause analysis, defect identification, and correction
- Any defects are isolated (usually to the current build), the build is demoted, the constituent modules are corrected, re-unit tested, re-promoted, and re-integration tested until all test cases have run failure free
- Validated builds are then subjected to their acceptance test plans in a similar manner
- A release is considered complete when the last build in that release has been validated and accepted
- A project is considered complete when all releases are completed

Regression testing can be an important element of the validation process, especially for large complex applications. *Regression* is the condition that occurs when a package that was previously validated has become invalid, thus necessitating regressing (or, logically dropping back) to an earlier point in the project's implementation in order to revalidate the package after any defects that gave rise to this condition have been corrected and validated. The continuous integration approach is designed to effectively manage all test libraries and package repositories to ensure that the balance between resource utilization (both technical and user), quality, schedule, and cost is always optimized.

The continuous integration approach as outlined above is designed to provide a unifying and seamless connection among the four vital components of any technology project: the **requirements** of the system, the **construction specifications** that describe the functionality that will support those requirements, the **packages** that define how and when that functionality will be delivered, and the **test plans** that serve to verify the overall quality, reliability, and integrity of the entire implementation.

This approach has the following benefits:

Project Management: Turning Concept Into Reality

- Allows progress to be defined in terms of actual customer capabilities that can be visualized and experienced, rather than by technical or project oriented events or phases, and thus serves as a practical expectation and progress management tool
- Ensures an operational subset of the solution is always available, i.e., that subset consisting of all those builds that have been integrated so far
- Allows the proposed benefit stream to be realized as early as is feasible
- Creates a momentum of success and true progress since new tangible value is available (typically) every three to six weeks
- Facilitates the transfer of ownership from the developers to the customer: a critical step for success
- Identifies defects (especially gaps in customer expectations) very early in the project life cycle which substantially reduces project costs while improving customer satisfaction
- Increases scheduling flexibility by permitting the build sequence to be altered as conditions warrant
- Isolates any problems to the current build, thus dramatically reducing correction and reintegration costs and schedule delays
- Promotes a high degree of parallelism in scheduling that dramatically reduces overall project calendar time

Project Roles

Regardless of project size, there are several key roles (Figure 12) that are central to success. On smaller efforts, for example, all these roles may be the responsibility of only one or two people, while on larger efforts, full-time resources may be required to effectively discharge their responsibilities.

The **project manager** is the individual fully responsible for meeting all project goals. There must be only one project manager for a given project. It is not unusual for projects to have several individuals who either formally or informally has been assigned this role. Or, sometimes there is no one assigned. This is a recipe for disaster, and must be avoided. Lack of clarity regarding the identity and responsibilities of the project manager only creates unnecessary problems and confusion.

On large efforts, it can be helpful to establish a staff function reporting to the project manager, called the project office, which provides administrative, analytical, and problem-solving support to the team in managing the plans, schedules, status, contingencies, sizing estimates, variances, and changes.

The **system architect** owns the design and its adherence to the requirements. In addition, the system architect is responsible for ensuring that the quality and conceptual integrity of the solution is managed and controlled. To that end, the system architect manages the optional independent test team, which develops the integration test plans, executes the associated test runs, and generates the appropriate failure reports for subsequent correction and repair by the relevant developers.

It should be pointed out that these two roles are very different. The project manager is the single responsible authority for delivering the solution as defined in the contract, i.e., as specified in the approved project plan. The system architect, on the other hand, is the chief designer, and is responsible for the solution design and its adherence to the requirements. Consequently, and because these roles require very different skills, it is extremely difficult to fill these roles with the same individual. Very few people have both the necessary architectural and design skills, as well as the management skills. One or the other inevitably is compromised, to the detriment of both the project and the customer.

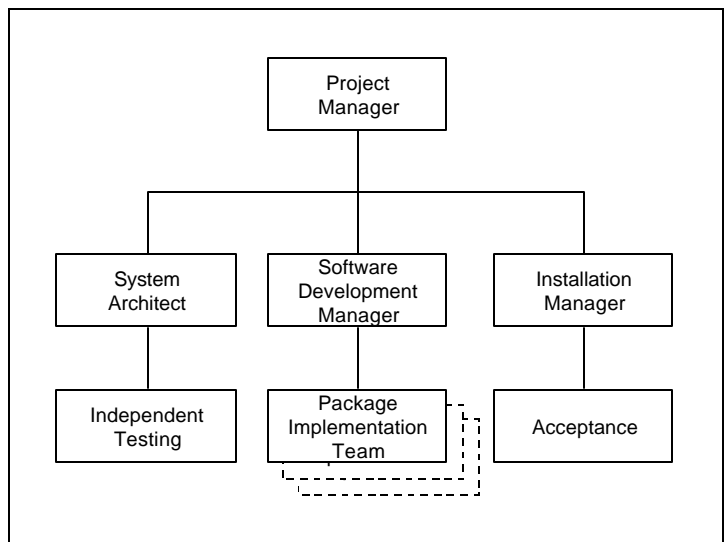


Figure 12. Key project roles.

Project Management: Turning Concept Into Reality

The **software development manager** is responsible for implementing the software deliverables that are defined in the packaging plan. This is the technical software management role, and it is vital that the individual performing this role has deep knowledge of the relevant solution technologies as well as the development environments and tool sets being used to design, construct, and validate the software.

The software development manager structures the delivery teams around the packaging plan. There will be a package (or, deliverable) manager for each package. That individual is fully responsible for the plan and delivery of that package.

	PM	SA	SDM	IM
Project Plan	X			
Requirements Specification		X		
Supporting Specification		Project, Release	Build, Module	
Construction Specification		Project, Release	Build, Module	
Affected Specification			X	
Defect			X	
Root Cause			X	
Failure		ITP	UTP	ATP
Specification Scope		Project, Release	Build, Module	
Target Requirement		ITP	UTP	ATP
Test Case		ITP	UTP	ATP
Validation Specification		ITP	UTP	ATP
Validation Plan		ITP	UTP	ATP
Package	Project		Release, Build, Module	

Figure 13. Ownership of project objects.

The **installation manager** is responsible for the training of the user personnel, the conversion (if needed) of any appropriate databases and files, and the general transition of workflows from the existing operational environment to the one defined by the project solution. In addition, this role is responsible for the production of the acceptance test plan by the user community (in conjunction with the system architect), and for the subsequent package validation and acceptance of all builds and releases, and for the project as a whole.

It is desirable (but not required) that customer personnel carry out this role. It requires an individual with strong logistical management and administrative skills, since much of this responsibility is focused on

the rapid and efficient movement of the various software packages through the acceptance process and into operational use, and making certain that people are being trained and provided the necessary infrastructure to succeed at each stage of the implementation.

Figure 13 summarizes these roles and the project objects that they are responsible for delivering.

Progress

The project manager prepares a status report every period (biweekly or semi-monthly is the preferred frequency), based on the aggregation of the individual status reports provided by the package managers. The status report should contain the following elements:

- **Accomplishments This Period.** Summarizes the packages (or, deliverables, in a non-software project) completed and any issues that have been resolved this period.
- **Planned Accomplishments for Next Period.** Identifies those packages and issues that are planned to be completed during the next reporting period.
- **Issues.** All open issues should be numbered so they can be tracked and measured. The project manager should attempt to quantify each issue with an assessment of the potential impact (hours, costs, and date) so that they can be prioritized. As issues get closed they drop off the list.
- **Progress against Plan.** A simple table (Figure 14) provides a complete quantitative picture of the project's progress against the current estimates and targets. There is typically a need for separate tables showing effort (person hours, person days), target dates, and costs.
 - *Original Estimate* is the effort agreed to in the original plan, budget, contract, etc., and serves as the base against which variances are reported. As changes are approved, this estimate can be updated accordingly, or a separate column can be added to track approved changes.
 - *This Period* is the effort incurred during the current reporting period.
 - *To Date* is the total effort actually incurred so far. A value of zero indicates that work has not commenced.

Project Management: Turning Concept Into Reality

- *Rem To Do* is the amount that the individual doing the work estimates is required to complete the assignment. This must be re-estimated every period. It is not simply an automatic subtraction of the *To Date* value from the *Original Estimate*. In other words, *Rem To Do* should always represent what the worker currently thinks is the effort required to complete that assignment based on the experience so far. A value of zero indicates that work is completed.
- *Var This Period* is simply the difference between the *Var Total* computed this period and the *Var Total* computed for last period. This value represents the variance actually incurred this reporting period and is a simple mechanism to track and manage slippages and over-runs before they become unmanageable. The volatility of this number (see Sizing) is an excellent indicator of project risk. Also, the trend of this value over time is a window into whether the project is improving or worsening.
- *Var Total* is the total variance and is the difference between the *Original Estimate* (plus any approved changes) and the sum of the *To Date* and *Rem To Do* values.
- Finally, it is useful to have a place where the relevant managers can include any written explanations for any current period variances. This tends to simplify status reporting and ensures that any material changes in progress are documented for learning and root cause analysis.

	Original Estimate	This Period	To Date	Rem To Do	Var This Period	Var Total
...
Module 8	125	25	100	25		
Module 9	85	20	30	60	-5	-5
Build D	210	45	140	75	-5	-5
...
Total Project	5200	350	800	4000	200	400

Figure 14. Example progress against plan (Effort Table).

The progress reporting structure should map directly to the WBS so that the relevant aggregations and summaries can be produced by package. This also provides the ability to drill down to finer details to

better understand trends and risks. For example, we see in Figure 14 that Build D is projecting an over-run of five hours for the implementation of Module 9. Figure 15 provides more detail on this status by revealing the WBS for this module. This shows that during the planning activities (*Plan*) the developer was able to complete that work ahead of estimate by five person-hours. However, the construction task (*Construct*), which is still in progress, is now projecting an over-run of ten person-hours. The sum results in the total five-hour projected over-run for the module.

	Original Estimate	This Period	To Date	Rem To Do	Var This Period	Var Total
Plan	10	0	5	0	5	5
Construct	35	20	25	20	-10	-10
Cover	10	0	0	10		
Validate	30	0	0	30		
Module 9	85	20	30	60	-5	-5

Figure 15. Example drill-down for progress of Module 9.

And, further, since neither the unit test planning (*Cover* task), nor the unit testing itself (*Validate* task) has started for this module, it is important for the project manager to ensure that the developers have reflected their current understanding of the risks and complexity that resulted in increasing the effort for programming is properly reflected in those two tasks. This is vital for gaining reliable plans and estimates. There is typically a great deal of learning that is gained early in the project life cycle that must be reflected in downstream activities, or it will be lost.

Change Control

Change is a part of life. Effective project managers understand that to be successful they must embrace and manage change, not let change manage them. There are two sources of change that demand attention: change in the original problem itself (i.e., a change in requirements or scope) and change in the solution.

A change in the problem is equivalent to a new project. In other words, all the issues related to defining the project initially apply equally to changes in that original problem. Changes in the problem arise chiefly from changes in the business environment, market place, and industry, as well as from a better understanding of the original problem, whether that understanding is on the part of the customer or the project team.

Project Management: Turning Concept Into Reality

A change in the solution is driven by either a change in the problem (in which case, the associated solution change represents a portion of the total cost of that problem change) or by an improved solution (i.e., through learning, better tools, improved technologies, etc.).

Any change that has a material effect on the five sizing variables (thresholds can be set to guide the team) should be documented in a formal manner and require customer, system architect, and project manager approval before any significant work is done. It is important for the project manager to formalize this process and to aggressively communicate to the team that it is not in the customer's interests to have team members unilaterally incorporating changes into the solution, no matter how seemingly small or isolated they appear to be.

Finally, one should not confuse changes with estimating variances. Change affects the delivered content (i.e., mass) of the solution while an estimating variance does not affect content. Estimating variances occur as the project team engages in carrying out their assigned tasks and are a function of the natural variability in the delivery environment and the fit of the skills and experience of the worker to the work. Estimating variances are addressed as part of the routine task management activities of the project.

Risk Management

In many ways, this is the essential discipline of an effective project manager. As was mentioned earlier (see Sizing) technology project risk is an aggregate measure of four risk factor categories:

- Complexity of the solution (the greater the complexity, the greater the risk)
- Organizational and business scope of the implementation (the larger the scope, the greater the risk)
- Stability of the environment (the more flux in the environment, the greater the risk)
- Reliability, relevance, and fit of the delivery process with the solution being contemplated (the less suited the process, the greater the risk)

Project managers use this risk value in two ways:

- To scale the derived variables (effort, time, and cost) when sizing the project since risk is a measure of their variability

- To identify and deploy specific risk mitigation action items to both reduce the likelihood of a given risk factor and to reduce the impact of that factor should it occur

The material found at the end of this discussion (Attachment: Project Risk Factors) is a guide to help project managers understand their project's risk profile.

Each risk factor is assessed a value from 1-5 (low to high). To help calibrate the responses and improve consistency of evaluation, a definition has been provided for three evaluation levels—the targeted level (3), and the two endpoints (1 and 5). Intermediate values can then be deduced from these definitions. Note that as with our discussion on sizing, the most useful risk assessments are those that have been calibrated based on actual experience in the environment the project is being conducted in.

To use this tool, add the risk assessment values and divide the total by twenty to compute the average risk value for the project. If this value is three or less, then the project is considered to have a manageable risk, and no intervention or special action is suggested. Regardless of this average value however, any risk factor with a value greater than three should be investigated and corrective action taken as part of normal project management. In other words, high risk factors should be aggressively pursued, regardless of the total average project risk.

These action items can then be monitored and a new risk profile derived to ensure that all project risks are being effectively identified and managed.

Quality

Quality means meets requirements. It is not a subjective assessment of relative "goodness". High quality solutions meet the requirements that the customers and the environment impose on it. Low quality solutions do not. Moreover, successful solutions must be both high quality and meet or exceed the customers' expectations (see Managing Expectations section).

The most effective quality programs are not really "programs" at all, but rather imbed into the solution delivery process itself the tools, information, and techniques that allow the workers to build and deliver solutions right the first time, and as a natural byproduct of doing the work. That is, the workers, not a separate quality assurance function, must be the ones who are fully responsible for the quality of their work products.

Project Management: Turning Concept Into Reality

The project manager must ensure that such a quality oriented solution delivery process (the preferred method described here is one example) is not only in place, but that it is also a stable process. In other words, the capability of the process must be predictable and repeatable. Only when this process is stable can it be improved in a measurable and sustainable way.

Project Feasibility

The decision to actually provide the resources to initiate a project is typically determined based on a cost benefit analysis or a business case. Clearly, this is a step that precedes the project work itself, which has been the primary focus of our discussion. Nevertheless, since the feasibility of any proposed effort must be based on the organization's best knowledge of the costs (including development costs) and benefits of such a project, it is not unusual for such a non-software deliverable such as a CBA, feasibility study, or business case to be prepared as part of the early stages of the work, and then serve as a checkpoint to further progress.

The exact structure of these deliverables is not critical so long as they provide the following information:

- Correlation table that demonstrates how the project's outcomes will support the organization's goals
- Net present value of each of the project's costs by category (including one-time and ongoing) such as hardware, software, direct labor, etc.
- Net present value of each of the project's benefits by category (including one-time and ongoing) such as revenue, earnings, etc.
- Itemization of the project's risks including probability and severity

Alternative implementation plans and approaches should be arrayed so that the appropriate decision makers can easily converge on the most attractive option. The do-nothing choice can also be helpful to depict, as well. The most attractive option will typically be the selection that has the maximum value with the lowest risk, where value equals the arithmetic sum of the NPVs of all costs and benefits.

It is absolutely vital to the integrity of this process that the senior managers who have the relevant responsibility for the various cost or benefit areas be the ones who take complete ownership for their particular portions of the analysis. For example, the CIO typically takes ownership for the development

costs, while a business manager will be required to take ownership for any increased revenue or reduced operating cost projections.

The goal correlation table portion of the analysis (this can be viewed as a characterization of the intangible benefits) is used to calibrate choices with similar value and risk projections.

Finally, when the project has been approved, the various components of the business case should be migrated into the project plan (see Project Plan) where they become a depiction of the project's goals.

Summary

Since project managers forge that essential link between strategy and execution, they are, almost by definition, strategic assets. This discussion has focused on a set of principles and approaches for assisting organizations and project managers in strengthening that link and thus increasing the strategic value of the enterprise.

Mr. Smith has been assisting businesses and technology organizations for over thirty years and is currently CEO of iTest Quality Partners, a consultancy, headquartered in Chicago, IL., specializing in helping firms maximize their strategic value. He can be reached at wsmith@itestqp.com.

Project Management: Turning Concept Into Reality

Attachment: Project Risk Factors

Factor	Description
Complexity	In general, project risks are directly proportional to technology, plan, and solution complexity.
Technology	<p>The more leading edge the technology that is being used, the higher the risk.</p> <ul style="list-style-type: none"> 1=technology is completely field-proven with every component having a significant installed base 3=technology is field-proven and has been utilized in a number of similar situations 5=technology is less than 3 years old or has few if any installations
Plan	<p>The more constraints (especially those related to calendar time) imposed on the project plan, the greater the risk.</p> <ul style="list-style-type: none"> 1=no meaningful constraints 3=project schedule and cost is driven primarily by project team estimates and work plans 5=project has <u>externally imposed</u> target date (cost) which is earlier (lower) than project team developed plans
Solution	<p>The greater the complexity of the constructed solution, the greater the risk.</p> <ul style="list-style-type: none"> 1=very simple solution with very few pieces 3=number of constructed components is well within team's experience, and number and type of interactions is relatively small and well-behaved, or minimal to less than 10% of package functionality is being modified 5=very large number of components with substantial interaction and interdependencies, or more than 30-40% of a package is being modified <p>Alternatively, the cyclomatic complexity (also known as McCabe's essential complexity) can be used (1=average cyclomatic number for all modules < 7 with no module > 10, 3= average cyclomatic number < 15, 5=average cyclomatic number > 50).</p>
Scope	In general, the greater the organizational and business scope of the proposed solution, the greater the risk.
Number of departments and people affected	<p>The greater the number of organizational units involved, the greater the risk.</p> <ul style="list-style-type: none"> 1=only one area affected with no dependencies 3=one area affected primarily, with minimal dependencies in other areas 5=50% or more of the business unit is directly involved or significantly affected
Degree of user training	<p>The greater the need for user training, the greater the risk.</p> <ul style="list-style-type: none"> 1=no user training required 3=minimal training required or small number of users need moderate to extensive training 5=most users require extensive training and support
Size and number of teams	<p>The more people involved, the greater the risk.</p> <ul style="list-style-type: none"> 1=project headcount is 5 or less 3=no more than 2-3 teams (each with 5 or less members) or a total project headcount not exceeding 15 5=project headcount exceeds 25
Stability	In general, the greater the stability, robustness, and resistance to change, the lower the risk.
Requirements	<p>The more stable the business and technical requirements, the lower the risk.</p> <ul style="list-style-type: none"> 1=very stable, virtually no changes in requirements 3=requirements and scope change averages no more than 15% of total cost/duration 5=requirements change exceeds 30% of total cost/duration

Project Management: Turning Concept Into Reality

Factor	Description
Design	<p>The more stable the implemented application and technical design, the lower the risk.</p> <ul style="list-style-type: none"> • 1=very stable, virtually no change in design • 3=design changes average no more than 15% of total cost/duration • 5=design change exceeds 30% of total cost/duration
Sponsors/ Users	<p>The more stable the customer team and the associated relationships, the lower the risk.</p> <ul style="list-style-type: none"> • 1=no changes at all • 3=no change in sponsors and no more than 15% change in user personnel that are directly associated with effort • 5=sponsor change or more than 30% change in user team
Degree of organizational change	<p>The greater the scope of structural, cultural, governance, and management changes that are required for implementation, the greater the risk.</p> <ul style="list-style-type: none"> • 1=no meaningful change in the organization • 3=very limited change • 5=high degree of organizational change
Number of new or heavily affected business processes	<p>The greater the change to the business or operating model, the greater the risk.</p> <ul style="list-style-type: none"> • 1=no changes to business processes • 3=no significant change to critical business processes and less than 15% change to remaining business processes • 5=substantial change to critical processes or greater than 30% change in general to existing business processes
Project team	<p>The more stable the project team, the lower the risk.</p> <ul style="list-style-type: none"> • 1=no changes at all • 3=no change in key players (project manager, system architect, software development manager, etc.) and no more than 15% change in personnel • 5=one or more key player changes or more than 30% change in project team
Fit	In general, the better the actual <u>applied fit</u> of the environment to the solution being delivered, the lower the risk. There are four major categories of fit: user, process, tools, and competency.
Degree of sponsor involvement	<p>The greater the sponsor involvement, the less risk.</p> <ul style="list-style-type: none"> • 1=sponsor is project manager, or equivalent • 3=sponsor is involved in all key decisions, participates in reviews and work sessions as planned and assumes ownership for results • 5=no sponsor or sponsor has limited involvement and does not assume ownership of results
Degree of user involvement	<p>The more the users stay involved with the effort, the less risk.</p> <ul style="list-style-type: none"> • 1=users are the developers and implementers • 3=users involved as planned and assume appropriate level of ownership of the result • 5=users uninvolved
Project plan effectiveness	<p>The more effective and complete the project plan, the lower the risk.</p> <ul style="list-style-type: none"> • 1=plan is completely in accordance with content, approach, and structure defined in text • 3=plan is generally consistent with overall ideas and concepts defined in text • 5=no plan

Project Management: Turning Concept Into Reality

Factor	Description
Quality of methodology	<p>The more effective and appropriate the solution delivery methodology, the lower the risk.</p> <ul style="list-style-type: none"> • 1=solution delivery methodology completely implements the preferred method defined in text • 3=solution delivery methodology is generally consistent with overall ideas and concepts defined in text • 5=no documented (or very incomplete) methodology <p>Alternatively, the SEI maturity model can be used as a quality guide for this assessment.</p>
Quality of tools	<p>The more effective and relevant the deployed tool and language set, the lower the risk.</p> <ul style="list-style-type: none"> • 1=tool set considered state of the art and extremely appropriate for achieving project goals • 3=tool and language set generally appropriate for project goals • 5=limited or poor quality tools and languages
Project management	<p>The more skilled and experienced the project manager, the lower the risk.</p> <ul style="list-style-type: none"> • 1=project manager experienced in a wide variety of environments and has deep knowledge of all tools • 3=project manager has successfully delivered similar projects • 5=new (or, no) project manager
Methods, languages, technologies, and tools being used	<p>The more experience the team has with the actual deployed project environment, the lower the risk.</p> <ul style="list-style-type: none"> • 1=extensive experience with every aspect of environment, many are considered tool experts and teachers • 3=team has successfully used substantially all elements of this environment before • 5=limited (or, no) experience with the environment
Business problem	<p>The more knowledge and understanding that the project team has of the business problem being addressed, the lower the risk.</p> <ul style="list-style-type: none"> • 1=team has extensive knowledge and understanding of all aspects of the business problem • 3=team has successfully dealt with this or a similar business problem in the past • 5=new area for most (or, all) members

For more information, please contact us at (630) 365-1606, or visit www.itestqp.com.