

Glue® Information Model

Value-Driven Risk-Adjusted Solution Delivery

NOTE: Lines represent active links that dynamically connect two object instances. Model can be directly traversed from a given object instance by following the chain of links to a desired object instance.

Problem

All business, functional, performance, and related requirements that solution must fulfill to be considered successful.

Solution

Collection of analysis, design, and software that fully defines the technology solution.

What is Value-Driven Risk-Adjusted Solution Delivery?

Value-Driven. Customer is the sole arbiter of quality and value. This business value perspective must be at the center of all priority, sequencing, and implementation decisions.
Risk-Adjusted. Unmanaged risk is the source of all problems. Technology organization must be able to continuously identify, and then aggressively manage solution acquisition risk. Permit customers to take the right risks, while at the same time indemnifying them against unnecessary exposure and uncertainty.
Solution Delivery. Primary goal is to *deliver business solutions*. That is, enabling and enhancing a company's ability to grow and compete in its marketplace through the acquisition (whether built or bought) of complete, fully integrated, and organizationally unified business capabilities. Not simply building and installing software.

Requirement and Solution Models

Two classes of objects, the Requirement Specification and the Construction Specification, permit multiple instances to be organized into a hierarchical structure. These multiple instances are used to define two models—the Requirement Model and the Solution Model. The Requirement Model consists of the hierarchy of Requirement Specifications that represents the totality of requirements that the solution must satisfy. The Solution Model consists of the hierarchy of Construction Specifications that collectively represent that corresponding solution. Together these models define the total business and technology capability that the Project is intending to deliver.

Defect

IsResolved
DefRemovalCost, OpenDate
CloseDate

Identifies the specification to which a given Defect, and the associated fix, applies.

The Four Software Engineering Domains

These four domains, *Problem, Solution, Validation, and Delivery* comprise the complete software engineering practice. All meaningful artifacts surrounding any technology effort are in one of these domains. Moreover, each of these domains has a distinct set of characteristics:

- Principles
- Vocabulary
- Metrics
- Practices
- Tools
- Skills

Finally, the organization of project materials into these four simple domains and the rigorous management of the connections among them greatly enhances the organization's ability to rapidly deliver business value, at low risk, to its customers: *Value-Driven Risk-Adjusted Solution Delivery*.

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 solution 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 simultaneously develop both the Construction Specifications that implement that Requirement, and the Test Cases that validate the chosen implementation. A Requirement is *valid* when all its Test Cases have *passed*.

Requirement 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. In other words, the Supporting Specification indicates which portion of the Solution Model fulfills a given portion of the Requirement Model. Design coverage is a measure of the degree to which the two models are congruent.

Construction Specification

Status
Size
InspectionOverRide

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 (i.e., detailed) the representation. In addition, a modification history is maintained for each abstraction level containing the sequence of modifications that were required to fully develop that abstraction instance. In other words, the most current modification for all abstractions represents the complete solution at that point in time, with the lower abstraction levels representing the overall architecture and design material for that solution, and the higher abstraction levels containing the actual source code implementation of that design.

Cost of Quality

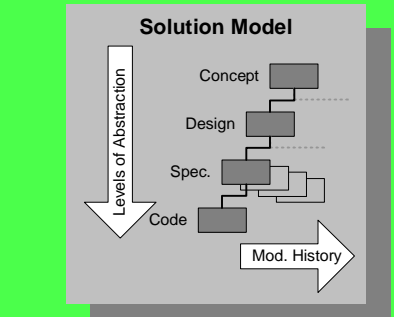
Prevention. Training, process, tools, organization.
Appraisal. Reviews, inspections, testing.
Defect Removal (DefRemovalCost). Fixes.
Internal Failure (IntFailCost). Rework, delays, waste, reduced benefits.
External Failure (ExtFailCost). Lost customers, market share, brand equity, and opportunities.
 Failure costs, especially external, are *unbounded* and high risk, thus warranting active investment in prevention and appraisal components.

Hierarchical representation of Requirements. This is the Requirement Model. Leaf level Requirements (Requirements with no children) must be *testable*.

Change

Identifies the Changes to a given Requirement. Typically a Change Event is documented with a work order, change notice, project request, or similar administrative vehicle.

An alteration to the Requirement Specification. A Change is an acknowledgement that the existing Requirement Model will not satisfy the organization's business goals, thus creating a situation where a modification to the Requirement Specification will be necessary.



Validation Plan

Defines the inspection plans for this Construction Specification.

Specification Scope

Identifies the unique Construction Specifications that comprise this Package's optional Specification Scope. The only way in which software can be validated, accepted, and actually delivered into production is to be explicitly connected to a Package instance in this manner.

Impact

Identifies the set of Impacts to the Work Plan that have been generated by this Change.

Identifies the Requirements a given Test Case will validate. Validation coverage is a measure of the degree to which the Requirements are being validated by the Test Cases. The goal is the minimum number of Test Cases that validate the maximum number of Requirements.

WBS

1 WBS
2 Task
Worker
OriginalStartDate
OriginalTargetDate
OriginalEffortEstimate

Tasks for implementing this Package: *Plan, Construct* (design, build or buy), *Cover* (test planning), *Implement* (all its constituent Packages), and *Validate*. Implementation of a Project is the implementation of its constituent Releases, implementation of a Release is the implementation of its constituent Builds, etc.

WBS for implementing this Package

Work Plan

Task Change

Identifies the specific task affected by this Impact object.

Continuous Integration and Incremental Delivery

This is a powerful technique for high performance, high quality delivery of business value in which the entire solution is constructed, validated, and accepted Package by Package in a series of iterations, where each iteration delivers an operational and functional subset of the total evolving solution. When the last Package has been integrated and delivered in this fashion the application is considered complete.

Progress

Period
EffortThisPeriod
EffortRemToDo
NewTargetDate
ExplanationOfVariance

Identifies the specific Reporting Period associated with this Progress instance.

Packaging Plan

Prerequisite

Functional Scope

- #### The Journey
1. Rigorous management control of all the artifacts in each domain.
 2. Simple tight linkage among domains.
 3. Each domain is a Center of Excellence.

Validation Specification

The collection of Validation Specifications that are used to validate a Package. The focus of validation (or acceptance) for any given Validation Specification (i.e., a test plan) is completely defined by that Package's Functional Scope. All Validation Specifications for a given Package are intended to validate that the collection of Packages within its Functional Scope—as a cohesive, integrated whole—deliver on their allocated Requirements.

Package

Status
Manager

Specifies implementation dependencies among peer Packages.

Packages are containers. They contain software as well as other Packages. The implementation of a Package consists of the implementation of all its constituent Packages. The collection of Package instances and their relationships as defined by the Package's Functional Scope is called the Packaging Plan. The complete work breakdown structure (WBS) for the implementation of a Package is defined by the aggregate WBS of the Packages in its Packaging Plan.

Logical grouping of related Test Cases organized into individual test runs, scenarios, sessions, etc. Each Test Suite is a separate automation point.

Test Suite

Mode
TestCaseFile

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. A Test Case has *passed* when it has no *unresolved* Failures.

Test Case

IsPass

Identifies the constituent Packages of a given Iteration. Iterations are dynamically generated based on the status of the Packages in the Packaging Plan and the type of Iteration desired:

- Integration Iteration.** Include a Package and its Functional Scope if the Package's status is either *Accepted* or *Valid*, or, if all the Packages in its Functional Scope are either *Accepted* or *Valid* (or, it has no Functional Scope), and all its Prerequisite Packages are either *Accepted* or *Valid* (or, it has no Prerequisites).
- Acceptance Iteration.** Include a Package and its Functional Scope if its status is *Accepted*, or, if it is *Valid* and all the Packages in its Functional Scope are *Accepted* (or, it has no Functional Scope), and all its Prerequisite Packages are *Accepted* (or, it has no Prerequisites).
- Production Iteration.** Include only Packages and their Functional Scope if Package status is *Accepted*.

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, or *proximate cause*, of the Failure. The underlying Defect can be in any Construction Specification in the Solution Model.

Test Data

Data required to execute a Test Case.

Status

At any point in time, a Package is in one of three states: *Unvalidated*, *Valid*, or *Accepted*. The diagram specifies the allowable state transitions. A Package is considered *Accepted* when all the Packages in its Functional Scope and all its Prerequisite Packages are *Accepted*, and all its unit, integration, and acceptance test plans have *passed*. A Package is considered *Valid* when all the Packages in its Functional Scope and all its Prerequisite Packages are either *Valid* or *Accepted*, and all its unit and integration test plans have *passed*. Otherwise, the Package is *Unvalidated*.

Failure

IsResolved
IntFailCost, ExtFailCost, OpenDate
Severity, CloseDate

An instance of actual results not matching expected results in a Validation Plan (or, in production). The execution of a Validation Plan can give rise to any number of Failures. Each Failure typically results from one or more 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. A Failure is considered *resolved*, if all its associated Defects are *resolved* and its Incident Test Case has *passed*.

Basic Principles

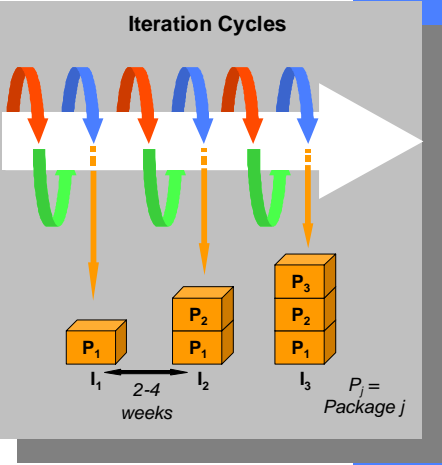
1. Risk is a measure of uncertainty.
2. Customer usage increases certainty, transfers ownership, and closes expectation gaps. Everything else is a weak approximation. The higher the delivery frequency, the greater the reduction in uncertainty, and the greater the energy utilization. Key question: "How many delivery cycles (iterations) are necessary to reduce risk commensurate with the energy expenditure?"
3. Reuse increases certainty. Assemble rather than build.
4. Process knowledge increases certainty. Success = continuous improvement + learning + facts + metrics.
5. Bugs decrease certainty. Never ship defects downstream.

Validation

Array of inspection plans, and unit, integration, and acceptance test plans focused on ensuring that solution fully addresses the problem.

Delivery

Small chunks of solution (feature sets) suitable for continuous integration and incremental delivery.



The Construction Specification uses a single stage validation: Status = (Unvalidated, Accepted). A Construction Specification is *Accepted* if all inspection plans have *passed* and all its Defects are *resolved*.

1 ● Cardinality n



www.itestqp.com
(630) 365 - 1606

Primary Metric Collection Point