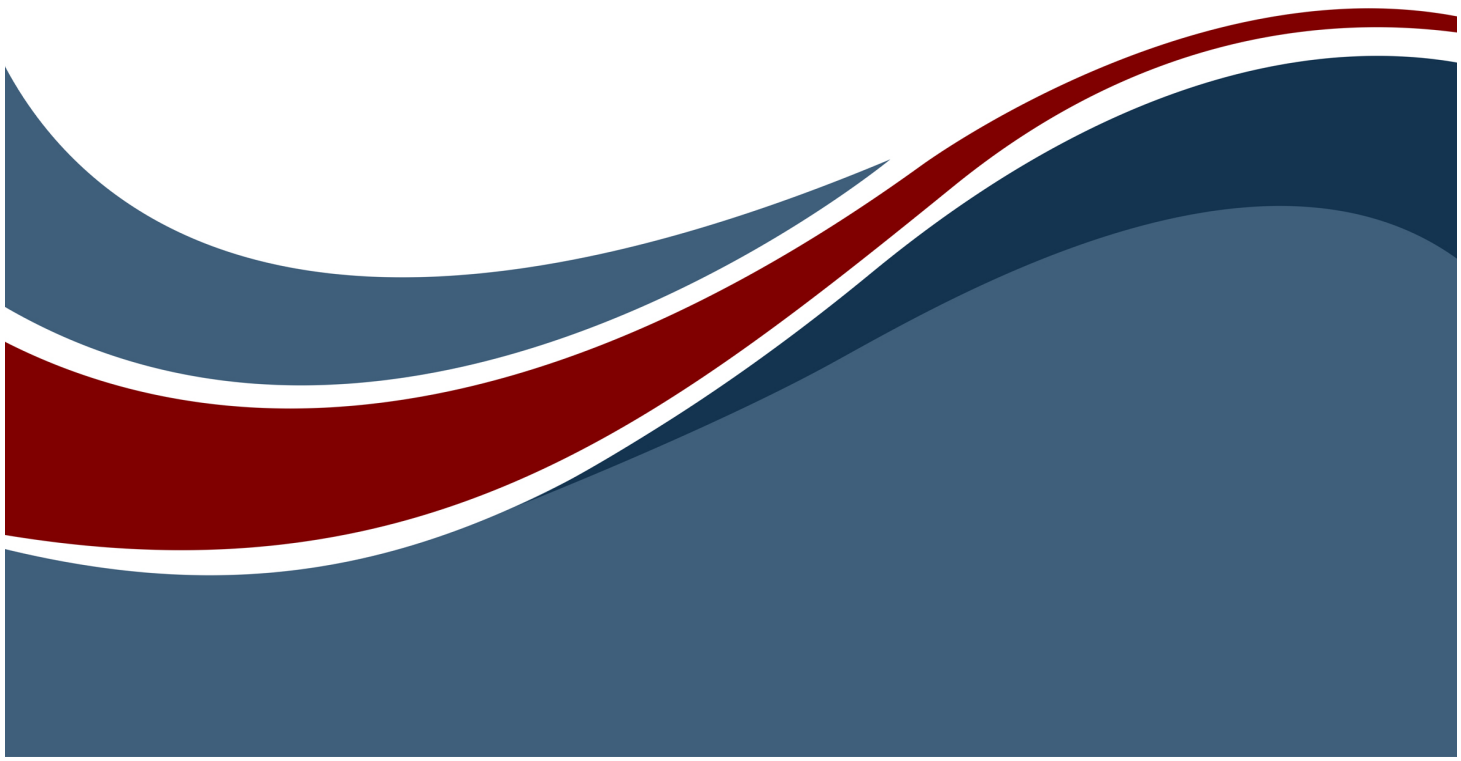
GENESYS™ 2022

System Definition Guide



Zuken Vitech Inc.
2270 Kraft Drive, Suite 1600
Blacksburg, Virginia 24060
540.951.3322 | FAX: 540.951.8222
Customer Support: support@vitechcorp.com
www.vitechcorp.com

Publication Date: August 2022

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CUSTOMER RESOURCE OPTIONS

Supporting users throughout their entire journey of learning model-based systems engineering (MBSE) is central to Vitech's mission. For users looking for additional resources outside of this document, please refer to the links below. Alternatively, all links may be found at www.vitechcorp.com/online-resources/.

Webinars

Immense, on-demand library of webinar recordings, including systems engineering industry and tool-specific content.

Screencasts

Short videos to guide users through installation and usage of GENESYS.

A Primer for Model-Based Systems Engineering

Our free eBook and our most popular resource for new and experienced practitioners alike.

Help Files

Searchable online access to GENESYS help files.

Technical Papers

Library of technical and white papers for download, authored by Vitech systems engineers.

Technical Support

Frequently Asked Questions (FAQ), support-ticket web form, and information regarding email, phone, and chat support options.

Our team has also created resources libraries customized for your experience level:

All Resources	Advanced
Beginner	IT / Sys Admin
Intermediate	Student

PREFACE

This System Definition Guide (SDG) provides a structured approach for populating a GENESYS™ project with systems engineering design information using the GENESYS base schema. It presents topical actions that must be accomplished in the context of the classic systems engineering activities of requirements analysis, behavioral analysis, physical architecture synthesis, and verification and validation as illustrated in Figure 1. Thus, the approach is consistent with commonly used systems engineering handbooks and standards, company-unique Capability Maturity Model® Integration (CMMI) processes, and project-specific Systems Engineering Plans (SEP) and Systems Engineering Management Plans (SEMP).

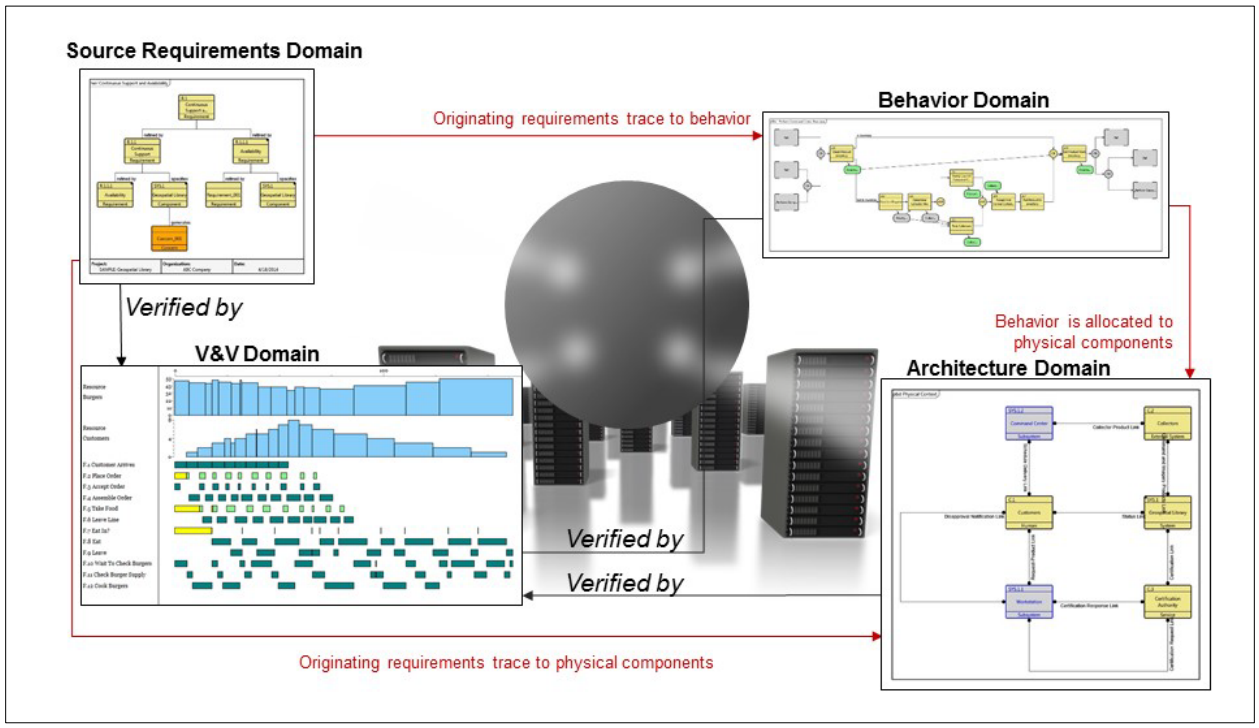
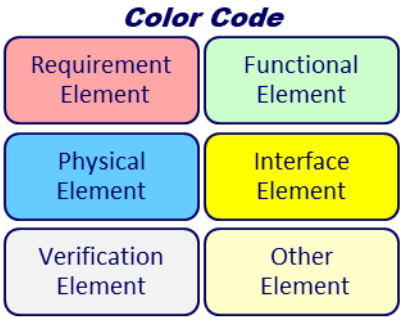


Figure 1 Systems Engineering Activities

This guide describes each activity and addresses the GENESYS schema classes used to capture the associated information. Following the activity discussion is a schema diagram and table identifying the principal schema classes, the classes' attributes and relationships typically used for the described activity. (Note: All the intrinsic relationships and targets are not presented to allow for focus on the key relationships for the topic addressed. The relationships and attributes shown on the diagram are the minimum recommended items that should be established when performing the referenced activity.) In addressing each activity, attention is given to populating the repository in a manner that facilitates the production of reports using the report writer provided with GENESYS. This guide describes the development of a system design that results in standard documentation at minimal additional cost.



The graphics used have the classes color coded so that the user can see at a glance if the class is a requirement element in the problem domain, a functional element or physical element in the solution domain, an interface element characterizing an exchange, a verification element to demonstrate the suitability of the solution architecture, or a broader concept.

This guide is intended to augment the Model-Based Systems Engineering (MBSE) with GENESYS training course, including the training course's reference material, to help the student retain what

was learned during the training class. The ultimate goal is to assist the systems engineer in making the most effective project use of GENESYS. The approach is generic and is not exhaustive of all cases and situations. The approach is written in the context of top-down systems engineering. The activities discussed in this guide can be re-ordered for middle-out or reverse engineering.

The following additional resources are available for use with this guide:

- For descriptions of different views, behavior diagram notation, and the mechanics of entering data into GENESYS, the reader is referred to the GENESYS Help/Documentation folder.
- For the definition of schema terms, the reader is referred to the GENESYS schema, which contains descriptions for each schema entity.

THIS PAGE INTENTIONALLY BLANK

INTRODUCTION

This guide outlines the first layer of a top-down design approach using the STRATA™ process. Reverse engineering or bottom-up and middle-out design approaches are discussed and outlined in the GENESYS training class materials. The approach used in top-down design has elements common to all three design approaches; therefore, it is important to understand the top-down methodology, which is emphasized in the GENESYS training class materials. STRATA satisfies, among others, the following key aspects of a complete systems engineering process:

- A convergent design approach leading to a realizable system solution provided such a system is not precluded by the system requirements.
- A model-based approach, where a system design repository is an embodiment of the design using a system design language.
- A layered and hierarchical approach, where the design is complete within a layer subject to the granularity of the layer. Layer 1 is the most abstract, while the last layer is more fine-grained and represents the top-level of the concrete (top-level physical) design.
- Design complexity is managed with this layered approach. The upper layers focus on a success-oriented design. As the behavioral aspects are allocated to the physical architecture, it is in the lower layers, where the impacts of physicality arise (error conditions, recovery, maintainability, etc.) and add behavior to the design.

Figure 2 represents the STRATA layered approach.

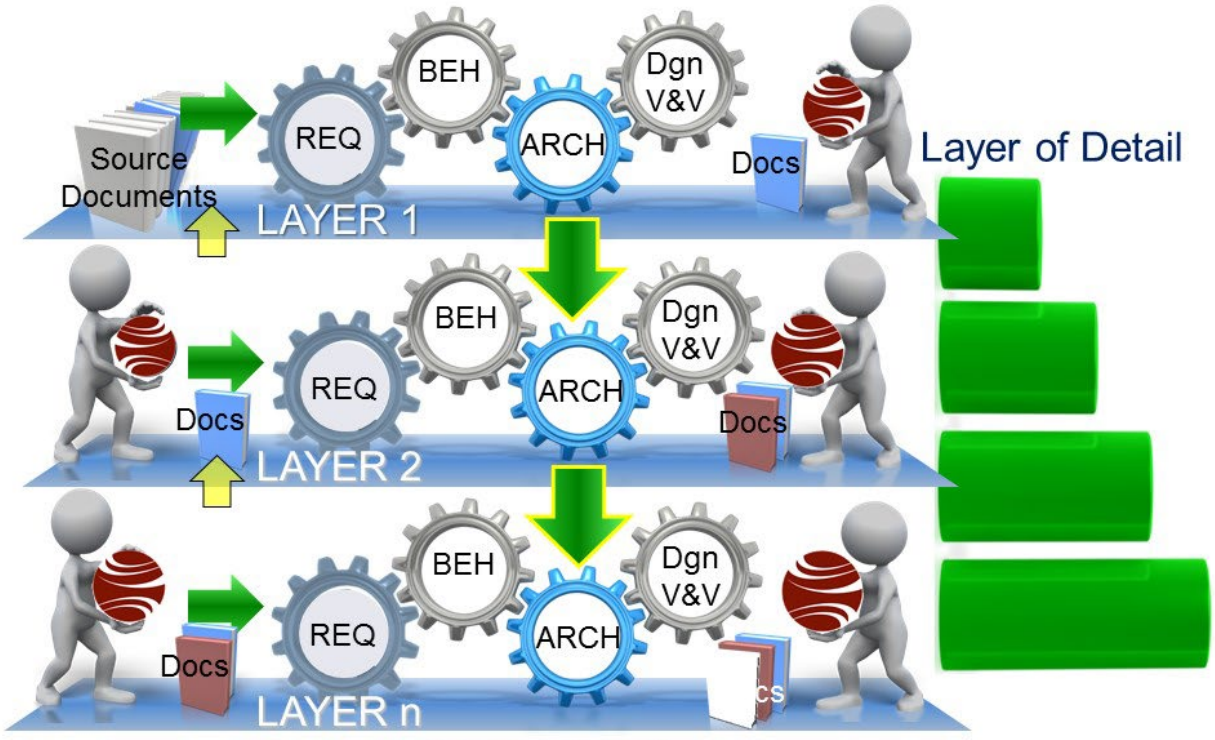


Figure 2 STRATA MBSE Process

The vertical arrows in Figure 2 are of two sizes. The larger downward-pointing arrow indicates the main workflow is focused on advancing the design to the next level of detail. The smaller upward arrow indicates some of the design decisions at the current layer have an impact on the previous layer. The horizontal arrows reflect the injection of source requirements into the current layer. All the source requirements are

not necessarily incorporated at the first layer because misallocation and distraction of engineering resources may occur.

There are several key benefits to note; however, these are not all the benefits available through the STRATA process. The key benefits are:

- Avoiding propagation of layer incompleteness errors into subsequent layers. A layer should be completed before moving onto the next layer. Using this approach does not guarantee that there will be no errors; only that the assumptions/decisions taken at the lower, subsequent layer are not dependent upon decisions which have not been made in the higher, previous layer. This removes a source of procedural errors, which is quite common with other approaches. A completed layer represents the engineering baseline for configuration control purposes. There are clear criteria for deciding when a layer is complete. Any iteration between layers only occurs between the current layer and the previously completed layer. This operational constraint ensures convergence of the process provided that the requirements lead to a feasible system.
- Where flaws exist, correction of them may occur without dire programmatic consequences. Resolving an error is done once and it is unnecessary to search for possible error artifacts. By framing and developing the solution in layers, STRATA ensures that assumptions, boundaries, interfaces, functions, and architectures are convergent, consistent, and complete.
- The four engineering domains are worked concurrently. These domains are the Requirements Domain, Behavior Architecture Domain, Physical Architecture Domain, and the Design Verification and Validation Domain. The Design Verification and Validation Domain is contextually broader than just validation and verification. This domain also includes all engineering studies, analyses, experiments, prototypes, and the like. The starting domain varies with the problem type. Top-down usually begins within the requirements domain, middle-out with the behavior domain, and bottom-up (reverse engineering) with the architecture domain.
- Engineering documentation is available at the completion of a layer, in particular, deliverable documentation. Formal specifications have different degrees of completion at each layer and each draft has a degree of usability consistent with the granularity of the layer.
- Documentation is tolerant of rapid program changes. Consequently, the process is fail-safe. If the systems engineering effort is stopped because of cost or schedule, consistent and usable documentation is available from the last completed layer, which is not the usual case with other approaches.
- Management has a clear idea of how complete the design is, the design's direction, and its real likelihood of completion.

REQUIREMENTS CAPTURE

This section is written assuming that the customer or end-user has been provided with a system requirements specification. If that is not the case, it is assumed that systems engineering will start with the task of collecting all stakeholder needs and transforming them into required functionality, and performance and design constraints. The end result of this effort will be a collection of requirements that are treated as originating (source) requirements (See Section 1.2).

1.1 Define Need and System Concept

Identify the system context and its mission. Physical entities, including the system of interest, external systems, and other entities affecting the system (excluding links) need to be identified and are represented in GENESYS as entities in the **Component** class. When creating a **Component** that represents the system, the attributes listed in the Table 1 should be assigned values. A **Component**'s Type attribute designates what the entity represents, in this case a system. The relationships and target classes associated with any entity created here will be established in subsequent sections.

Table 1 System Definition

Entity Class	Attributes	Relations	Target Classes
Component	Abbreviation Description Doc. PUID Mission Number Operations Purpose Receptions Type: System Values		

1.2 Capture Source (Originating) Requirements

Capturing requirements from source documents involves the creation of entities in the GENESYS repository in the following classes:

- **Document:** create an entity for each source document
- **RequirementGroup:** create an entity for each requirement group (often maps directly to a section heading in a requirements document)
- **Requirement:** create an entity for each source requirement
- **ExternalFile:** create an entity for each source requirement-related table or graphic
- **DefinedTerm:** create an entity for each pertinent acronym or special term in the source requirement documentation

As part of extracting and populating the **Requirement** class, the following should be performed:

- Parse compound requirements into single, verifiable **Requirement** statements.[1] These should be linked to their parent **Requirement** using the *refines/refined by* relation.
- Place any requirement's tables and graphics in separate files and reference them in the project repository using **ExternalFile** entities where each entity *augments* the subject **Requirement** to which it is related. When reports are utilized later in the project, links are

[1] Capture each requirement statement in the Description attribute of the corresponding entity in the **Requirement** class.

used to include these external tables and graphics in the output immediately following the entity Description and make entries in the List of Figures and List of Tables, as appropriate. In order to properly number and label the tables and graphics for inclusion in the output, only a single graphic or table should appear in each external file entity.

- Optionally, relate any requirements that share common categorization to **RequirementGroup** entities using the *grouped by/groups* relation. Often, **RequirementGroup** entities map to section headings in traditional textual requirements specifications. This is especially helpful in avoiding the unadvised practice of using a **Requirement** entity as a container for grouping other requirements, as requirements are not containers and should not be used in that way.[2]
- Acronyms and/or special terms appearing in the source document should be captured in the repository as **DefinedTerms**. For an acronym or abbreviation, the acronym is entered into the Acronym attribute and what it stands for is entered as the name of the entity. For a special term, the term is the name of the entity and its definition is entered into the Description attribute. By filling in both the Acronym and Description attributes, appropriate entries will appear in both the acronym and glossary sections of documents generated using the GENESYS reports once the **DefinedTerm** is linked to the output **Document** using the *used by* relation.

The following paragraphs contain information on special topics concerning the entry of source requirements.

Extracting requirements from source documents. The entry of source requirements into a GENESYS repository can be accomplished by copying and pasting the information from the originating document into the entities in the repository. To do this the user should copy the text for a requirement from the originating document and use the “Paste Unformatted” command in the Description attribute box for the requirement. Alternatively, the document parser feature can be used to facilitate creation of the requirement entities. Also, there is an option to import requirements from IBM® DOORS®. The process to accomplish this is covered in the DOORS Connector Guide.

Setting the Origin attribute. It is important to determine the customer’s acceptable requirements traceability. A customer may require traceability to the exact wording in a source document or may allow traceability to parsed statements. Once this decision has been made; set the Origin attribute to “Originating” for each **Requirement** in the document requirements hierarchy down through the lowest-level traceable **Requirements** (i.e., those deemed originating by the customer). For all other **Requirements**, set the Origin attribute to “Derived” (except those arising from the resolution of a **Concern**, for these, set the Origin attribute to “Design Decision”). [3]This will record which **Requirements** are originating and ensure that the traceability matrix produced by the GENESYS reports trace to the correct source **Requirement** entities.

Establishing Doc. PUIDs. If the source documents have assigned Project-Unique Identifiers (PUIDs) to the requirements, these should be captured in the repository in the Doc. PUID attribute of the **Requirement**. If PUIDs have not been pre-assigned, it is advisable to assign one to each originating **Requirement**. This can be done manually or by utilizing the Assign Documentation PUID wizard under the Project Tab.

Note: If available, Doc. PUIDs are automatically output by all of the included reports. To take advantage of this feature, Doc. PUIDs should be assigned to entities in the following classes: **Component**, **Function**, **Item**, **Link**, **Mode**, **Requirement, State, UseCase**, and **VerificationRequirement**.

---

[2] Often, organizations have rules that state that all requirements must be formally verified. If one uses requirements as containers, that increases the number of formal requirements that need formal verification unnecessarily and leads to confusion as to why there are often requirements without any text or definition. Using RequirementGroup to associate requirements into like groups or categories avoids this problem.

[3] The terms Originating, Derived, and Design Decision distinguish customer source requirements, directly attributable requirements, and requirements stemming from engineering decisions respectively.

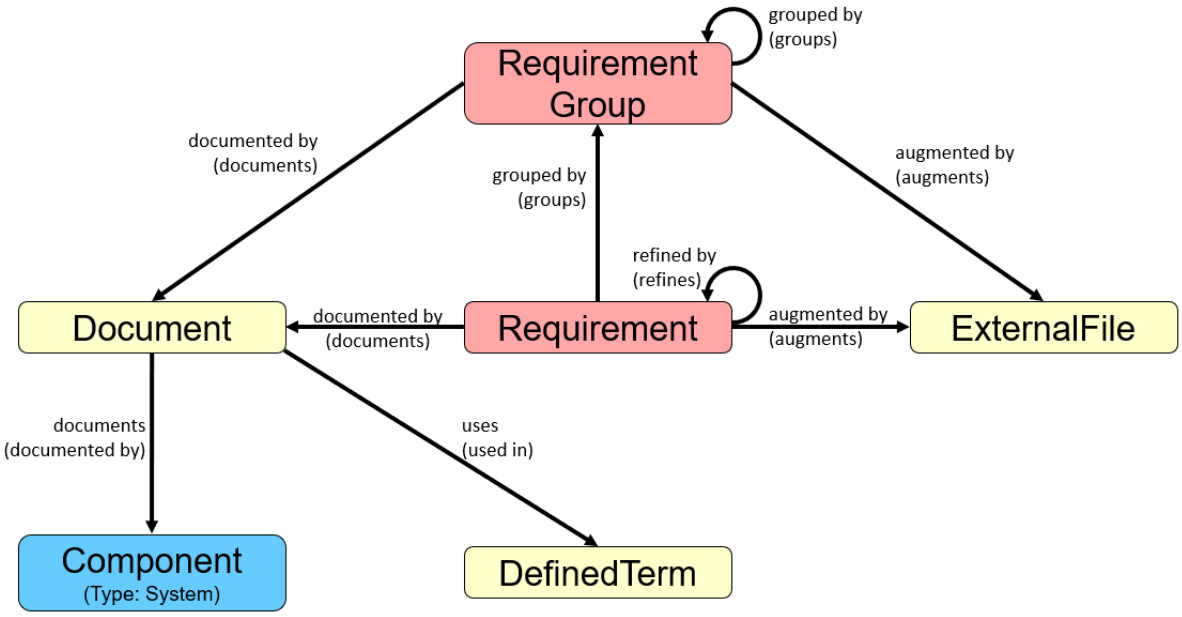


Figure 3 Source Requirements

Table 2 Source Requirements

Entity Class	Attributes	Relations	Target Classes
Component (Type: System)	See Section 1.1	*documented by (documents)*	**Document**
DefinedTerm	Acronym Description	*used in (uses)*	**Document**
Document	CDRL Number Contract Line Item Number Description Document Date Document Number External File Path either[4] Govt. Category or Non-Govt. Category Number Revision Number Type	*documents (documented by)*[5]	**Component** (Type: System) **Requirement** **RequirementGroup** **UseCase**
		uses (used in)	**DefinedTerm**

[4] These attributes are used when the source document is to be listed as an applicable document in a report generated from the repository. See Section 1.4 for an explanation.

[5] Only the top-level **Requirements** need to be *documented by* the source **Document**. The included reports search up the requirements hierarchy to locate the source **Document**.

Table 2 Source Requirements

Entity Class	Attributes	Relations	Target Classes
ExternalFile	Description External File Path Number Page Orientation Title Type	*augments* *(augmented by)*[6]	**Requirement** **RequirementGroup**
Requirement (Origin: Originating)	Description Doc. PUID Incentive Performance Parameter Key Performance Parameter Number Paragraph Number[7] Paragraph Title Rationale Type Weight Factor	*augmented by (augments)*	**ExternalFile**
		documented by (documents)	**Document**
		refined by (refines)	**Requirement**
		grouped by (groups)	**RequirementGroup**
RequirementGroup (Origin: Originating)	Description Name Number Origin Paragraph Number Error! Bookmark not defined. Paragraph Title Doc. PUID Title Weight Factor	*augmented by (augments)*	**ExternalFile**
		documented by (documents)	**Document**
		grouped by (groups)	**RequirementGroup**
		groups	**Requirement**

Warning: Note that text attributes do not support embedded tables and graphics. Therefore, tables and graphics should be captured as ***ExternalFile*** entities.

1.3 Define System Boundary

Based on an examination of the originating requirements or related source documents, identify the system boundary and context. To define the boundary, identify each external with which the system must interface. An external is represented as a **Component** and may identify the system's environment, an actual external system, or a human. Create a **Component** entity representing the context and decompose it into the system and its externals using the *built from* relation. Set the Type attribute as appropriate for each **Component**. Note that humans may be considered as part of the system or as external to the system depending on the

[6] The Position attribute of this relationship should be set to control the order in which multiple external files are appended to the **Requirement**'s Description attribute when it is output in reports generated from the repository.

[7] Used to record the source document paragraph number and title for an originating **Requirement** or **RequirementGroup**.

actions they take or the roles they play in performing the system functions. In many cases, there are humans who are part of the system and humans who are external to the system.

To complete the system boundary definition, identify all interfaces between the system and each external by creating entities in the **Link** class. Defining a **Link** entity establishes that the system interacts with an external. Typically, there will be only one interface between the system and each external. The details of the interface are characterized by child **Link** entities (See Section 1.16).

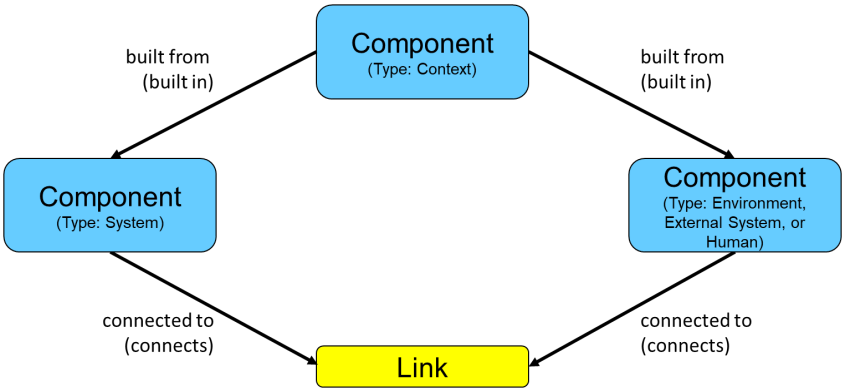


Figure 4 System Boundary

Table 3 System Boundary

Entity Class	Attributes	Relations	Target Classes
Component (Type: Context)	Description Number	*built from (built in)*	**Component** (Type: System, Environment, External System, or Human)
Component (Type: Environment, External System, or Human)	Abbreviation Description Number	*built in (built from)*	**Component** (Type: Context)
		connected to (connects)	**Link**
Component (Type: System)	See Section 1.1	*built in (built from)*	**Component** (Type: Context)
		connected to (connects)	**Link**
Link	Description Doc. PUID Number	*connects (connected to)*	**Component** (Type: System and Environment, External System, or Human)

Suggestion: Create a folder for the context and externals to separate them from the evolving system component hierarchy. Typically, the context and externals are given a different numbering scheme than the entities in the system component hierarchy to differentiate them in GENESYS views such as the Physical Block Diagram and Hierarchy diagrams.

1.4 Collect Additional Applicable Documents

Identify any other applicable or reference documents such as standards, regulatory documents, and Interface Control Documents for interfaces to existing external systems. These or specific portions of these documents may be referenced in the source requirements. If needed, extract additional **Requirements** and **RequirementGroups** from the applicable or reference documents.

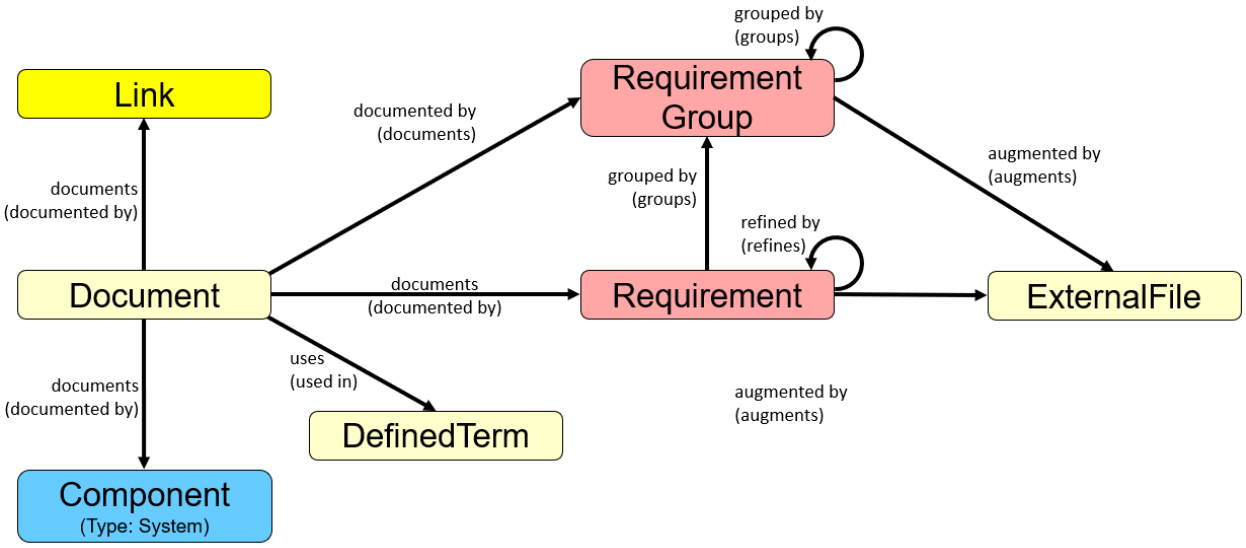


Figure 5 Applicable Documents and Requirements

Table 4 Applicable Documents and Requirements

Entity Class	Attributes	Relations	Target Classes
Component (Type: System)	See Section 1.1	*documented by (documents)*	**Document**
DefinedTerm	See Section 1.2	*uses (used in)*	**Document**
Document	See Section 1.2	*documents (documented by)*	**Component** **Link** **Requirement** **RequirementGroup**
		uses (used in)	**DefinedTerm**
ExternalFile	See Section 1.2	*augments (augmented by)*[8]	**Requirement** **RequirementGroup**
Link	See Section 1.3	*documented by (documents)*	**Document**
Requirement	See Section 1.2	*augmented by (augments)*[5]	**ExternalFile**
		documented by (documents)	**Document**
		refined by (refines)	**Requirement**
		grouped by (groups)	**RequirementGroup**
RequirementGroup	See Section 1.2	*augmented by (augments)*	**ExternalFile**
		documented by (documents)	**Document**
		grouped by (groups)	**RequirementGroup**
		groups	**Requirement**

Suggestion: Create folders to group source documents and applicable documents.

[8] The Position attribute of this relationship should be set to control the order in which multiple external files are appended to the requirement description when it is output in reports.

REQUIREMENTS ANALYSIS

Requirements analysis involves a collection of concurrent, inter-related activities. These are addressed in the following subsections.

1.5 Parse Originating Requirements

If not previously done when capturing source **Requirements** (See Section 1.2), parse the originating **Requirements** into single, verifiable **Requirements** statements. Related requirements can be grouped into a **RequirementGroup**. This parsing can result in **Concerns** to be resolved. These should be identified as described in Section 1.6 below. See Section 1.2 for a discussion of Originating vs. Derived or Design Decision **Requirements**.

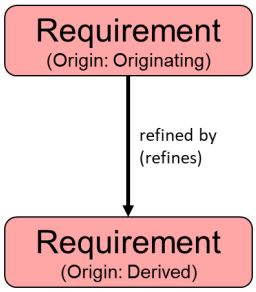


Figure 6 Derived Requirements

Table 5 Derived Requirements

Entity Class	Attributes	Relations	Target Classes
Requirement (Origin: Originating)	Description Doc. PUID Number Rationale Type Weight Factor	*refined by (refines)*	**Requirement** (Origin: Derived)

1.6 Identify Requirement Concerns and Risks

Requirement Concerns. Examine each parsed source **Requirement**, capturing any questions or problems identified by creating **Concern** entities. The assignment of resolution responsibility to an individual or organization is captured by the *assigned to* relation between the **Concern** entity and an **Organization** entity. The resolution of a **Concern** may require customer involvement and/or trade studies. These should be captured in the repository using the **Document** entity and linked to the **Concern** using the *documented by* relation. The resolution of a **Concern** is not a requirement in itself but generally either *results in* a design decision **Requirement,** or the addition or clarification of one or more other **Requirements**. Any resultant **Requirement** (Origin attribute set to “Design Decision”) should be linked to both the **Concern** (*result of* relation) and the **Requirement(s)** (*refines* relation) that generated the **Concern**.

Requirement Risks. Risks are possible problems that are significant enough to potentially affect the achievement of a major program objective or milestone. Because the information needed is different than that of a **Concern**, **Risk** is a separate entity class in GENESYS. **Requirements** are among the many sources of program risk. Therefore, examine each leaf-level source **Requirement** and identify any **Risks**. Systems engineers or risk management personnel, depending on the project organization, may enter or manage **Risks** in GENESYS. Generally, **ProgramActivity** or **ProgramElement** risks are addressed in the Program Management Facility. Details of program management are not addressed in this guide. Any risk status graphs should be identified as **ExternalFiles** and linked to the **Risk** using the *augments* relation. Any risk mitigations should be identified as **MitigationActivity** elements and related to the risk using the

mitigates relation. Similarly, any risk status reports should be identified as **Documents** and linked to the **Risk** using the *documents* relation. For more information on Risk Management refer to section 1.7.

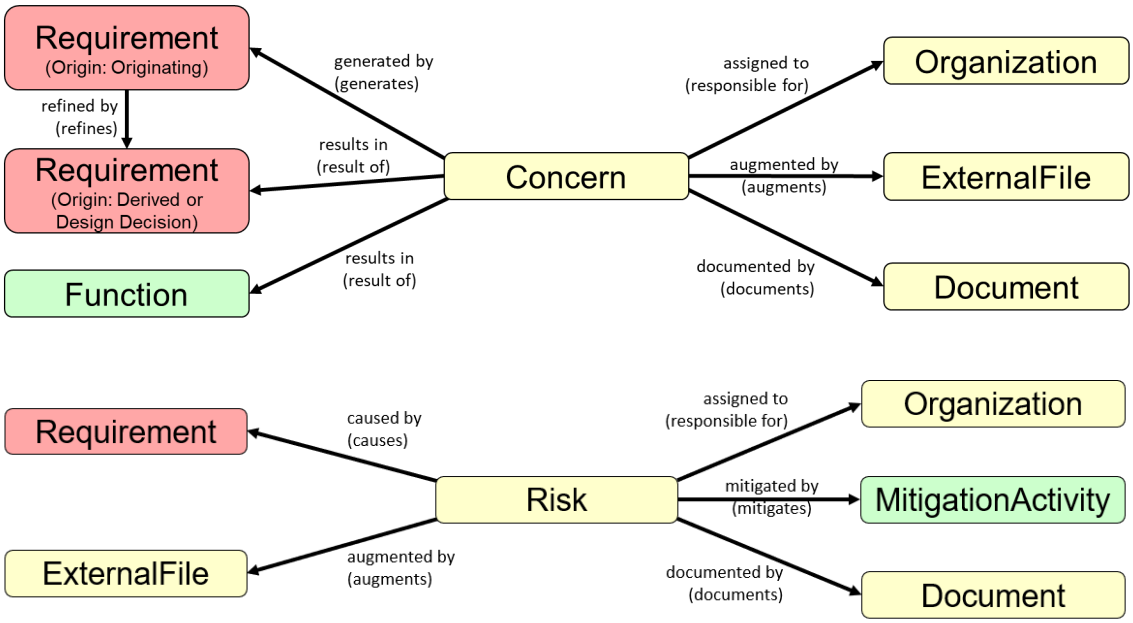


Figure 7 Requirement Concerns and Risks

Table 6 Requirement Concerns and Risks

Entity Class	Attributes	Relations	Target Classes
Concern	Alternatives Assumptions Decision Description Date Closed Due Date Importance Number Originator Rationale Status	*assigned to (responsible for)*	**Organization**
		augmented by (augments)	**ExternalFile**
		documented by (documents)	**Document**
		generated by (generates)	**Requirement**
		results in (result of)	**Requirement**
Document	See Section 1.2	*documents (documented by)*	**Concern** **Risk**
ExternalFile	See Section 1.2	*augments (augmented by)*	**Concern** **Risk**
Function	Description Doc. PUID Duration	*result of (results in)*	**Concern**
MitigationActivity	Name Number	*mitigates (mitigated by)*	**Risk**

Table 6 Requirement Concerns and Risks

Entity Class	Attributes	Relations	Target Classes
	Description StartDatePlanned StartDate EndDatePlanned EndDate Duration Status Result		
Organization	Abbreviation Description Number Role	*responsible for (assigned to)*	**Concern** **Risk**
Requirement	See Sections 1.2 and 1.5	*causes (caused by)*	**Risk**
		generates (generated by)	**Concern**
		refined by (refines)	**Requirement**
		result of (results in)	**Concern**
Risk	Consequence Description Handling Approach Likelihood Number Risk Rating Risk Score Scoring Rationale Significance Status Trigger Date Type	*assigned to (responsible for)*	**Organization**
		augmented by (augments)	**ExternalFile**
		caused by (causes)	**Requirement**
		documented by (documents)	**Document**
			MitigationActivity
		mitigated by (mitigates)	

1.6.1 Informal Comments using the Note Class

The **Note** class may be used to provide informal comments (additional information or queries) regarding the characteristics of a particular entity in the design model. A **Note** may be related to any other entity in the model. The **Note** properties allow the design team to status a **Note**, provide a decision on incorporating the **Note**, and elevate the importance of the **Note** by relating the **Note** to a **ChangeRequestPackage, Concern**, or **Risk**.

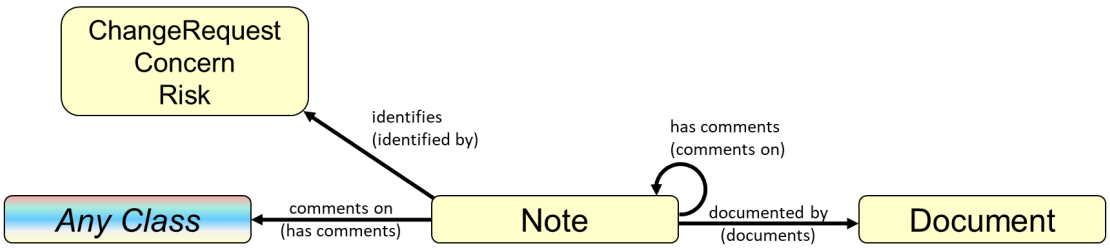


Figure 8 Note Class

Table 7 Note Class

Entity Class	Attributes	Relations	Target Classes
Note	Decision Description Status Type	*comments on (has comments)*	**Entity**
		documented by (documents)	**Document**
		identifies (identified by)	**Nexus** **Risk**

1.7 Risk Management

Risks are possible problems that are significant enough to potentially affect the achievement of a contractual requirement, major program objective or milestone. Risks can be defined as the degree of exposure to an event that might happen to the detriment of a program, project, or other activity. Risks are described by a combination of the probability that the risk event will occur and the consequence of the extent of loss from the occurrence, or impact. Risk is an inherent part of all activities, whether the activity is simple and small, or large and complex.

1.7.1 Typing Risks

Risks are typed into four main categories, as defined by the INCOSE Systems Engineering Handbook v4 (pg. 220):

- **Technical Risk** – The possibility that a technical requirement of the system may not be achieved in the system life cycle. Technical risk exists if the system may fail to achieve performance requirements; to meet operability, producibility, testability, or integration requirements; or to meet environmental protection requirements. A potential failure to meet any requirement that can be expressed in technical terms is a source of technical risk.
- **Cost Risk** – The possibility that available budget will be exceeded. Cost risk exists if a) the project must devote more resources than planned to achieve technical requirements, b) the project must add resources to support slipped schedules due to any reason, c) if changes must be made to the number of items to be produced, or d) if changes occur in the organization or national economy. Cost risk can be predicted at the total project level or for a system element. The collective effects of element-level cost risk can produce cost risk for the total project.
- **Schedule Risk** – The possibility that the project will fail to meet scheduled milestones. Schedule risk exists if there is inadequate allowance for acquisition delays. Schedule risk exists if difficulty is experienced in achieving schedule technical accomplishments, such as the development of software. Schedule risk can be incurred at the total project level for milestones such as deployment of the first system element. The cascading effects of element-level schedule risks can produce schedule risk for the total project.
- **Programmatic Risk** – Produced by events that are beyond the control of the project manager. These events often are produced by decisions made by personnel at higher levels

> of authority, such as reductions in project priority, delays in receiving authorization to proceed with a project, reduced or delayed funding, changes in organization or national objectives, etc. Programmatic risk can be a source of risk in any of the other three risk categories. (INCOSE, pg. 220).

If a program requires additional categories, these may be added via the schema editor. Typical additional categories may include “security” or “safety.” Safety may be a useful category if the user or user’s organization wishes to utilize risk management as a lightweight means of conducting safety hazards analysis.

1.7.2 Risk Process

A number of standards define similar steps for holistic risk management, including the INCOSE Systems Engineering Handbook (v4), the Risk Management Guide for DOD Acquisition (sixth edition), and the NASA Risk Management Handbook (NASA/SP-2011-3422 version 1.0). A consolidated continuous risk management process consists of the following steps:

- Risk Identification
- Risk Analysis
- Risk Handling Planning
- Risk Handling Plan Implementation
- Risk Tracking

1.7.2.1 Risk Identification

Risk identification is usually a group brainstorming activity, assessing the requirements, state of the design, external forces, and conditions on the project, etc. This is where risks are generated and documented. This should be a recurring step. At this point, a description, the type, and status should be set at a minimum. In a program that used multi-tiered risk management identification processes (say for example at the component level and the system level), lower-level **Risks** could *lead to* high level **Risks**. In this way, one can show the traceability between **Risks**.

1.7.2.2 Risk Analysis

Risks are typically analyzed in at two dimensions: likelihood and impact/consequence. GENESYS supports a standard 1 to 5 rating in each of these dimensions, allowing for risks to be plotted on a standard 5x5 risk cube.

Likelihood is the probability of occurrence of the risk. The 1 to 5 ratings are defined in probability bands as:

1 - (0% < p < 20%)
2 - (20% <= p < 40%)
3 - (40% <= p < 60%)
4 - (60% <= p < 80%)
5 - (80% <= p < 100%)

Consequence is the severity of adverse effects stemming from the risk. The 1 to 5 ratings are defined in probability bands as:

1 - Minimal or no consequence to technical performance.
2 - Minor reduction in technical performance or supportability, can be tolerated with little or no impact on program.
3 - Moderate reduction in technical performance or supportability with limited impact on program objectives.
4 - Significant degradation in technical performance or major shortfall in supportability; may jeopardize program success.
5 - Severe degradation in technical performance; cannot meet KPP or key technical/supportability threshold; will jeopardize program success.

Additionally, the Trigger Date should be determined. The Trigger Date is when the risk owner/estimator believes that the risk event will occur, at which time it will become known if the risk occurred and is now an issue, or if it has not occurred and the risk can be retired. Trigger Date is a useful field, which allows the team to ensure that risks with upcoming trigger dates are reviewed more frequently.

1.7.2.3 Risk Handling Planning

While we typically think of mitigating risks, there are additional options. The four standard risk handling approaches, as defined by the INCOSE Systems Engineering Handbook, v4 (pg. 120) are:

- Avoid the risk through change of requirements or redesign
- Accept the risk and do no more
- Mitigate the risk by expending budget and other resources to reduce likelihood and/or occurrence
    - These should have Mitigation Activities developed and related to the risk
- Transfer the risk by agreement with another party that it is in their scope to mitigate
    - These risks should be associated with an **Organization** entity using the *assigned to* relationship

GENESYS supports definition of the handling approach via the Handling Approach attribute. Handling approaches should at least be established for the moderate and high-risk items, at a minimum. One may choose to modify the handling approach as time progresses.

If the handling approach results in a needed change to the requirements, functional architecture, physical architecture, V&V, or program structure, a **ChangeRequestPackage** can be generated if a formal record of the change is needed. Otherwise, the impacted entities can be identified via the *impacts* relationship. A combination of the Handling Approach attribute, the Status attribute, and the entity at the target of the *impacts* relationship defines the configuration of the change.

1.7.2.4 Risk Tracking

The status of **Risks** and their Handling Approaches should be assessed periodically. This may result in the **Risk's** Likelihood, Consequence, and the resultant Risk Rating and Risk Score being updated based on changing program circumstances or progress of the handling activities. The Status will also change as the **Risk** moves though its lifecycle.

A **Risk** could impact a number of other entities if it is realized. The impacted entities in the model can be identified by setting the Status attribute to "realized" and created an *impacts* relationship to the impacted entity.

Any risk status graphs should be captured as **ExternalFile** entities and linked to the **Risk** using the *augments* relation. Similarly, any risk status reports should be captured as Document entities and linked to the **Risk** using the *documents* relation.

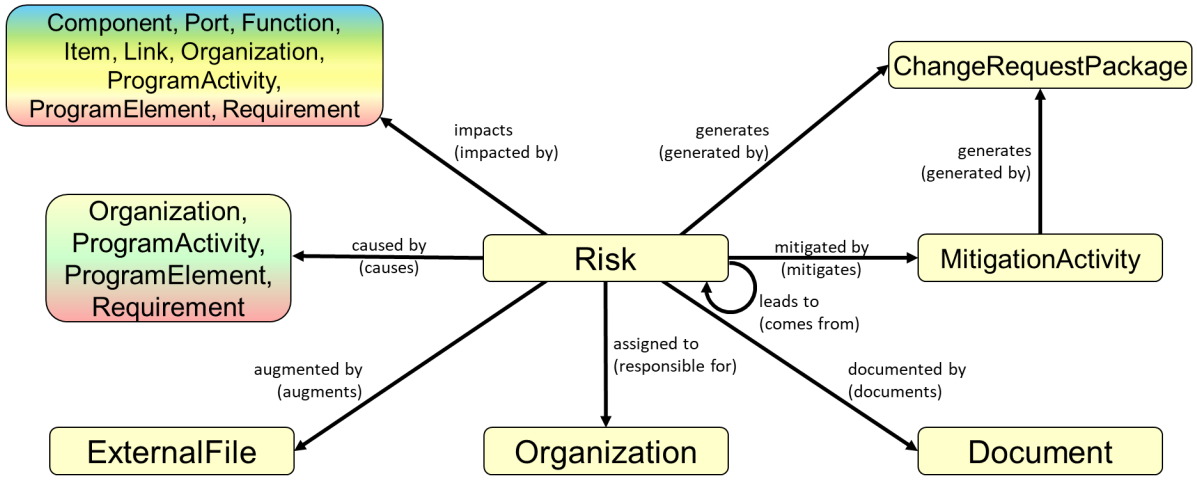


Figure 9 Risk Management

Table 8 Risk Management

Entity Class	Attributes	Relations	Target Classes
ChangeRequestPackage	Alternatives Assumptions Change Request Number Date Closed Decision Description Due Date Importance Originator Rationale Status	*generated by (generates)*	**MitigationActivity** **Risk**
Component	See Section 1.1	*impacted by (impacts)*	**Risk**
Document	See Section 1.2	*documents (documented by)*	**Risk**
ExternalFile	See Section 1.2	*augments (augmented by)*	**Risk**
Port	Abbreviation Description Direction Doc. PUID	*impacted by (impacts)*	**Risk**
Function	See Section 1.6	*impacted by (impacts)*	**Risk**

Table 8 Risk Management

Entity Class	Attributes	Relations	Target Classes
Item	Accuracy Description Doc. PUID Fields Precision Priority Range Size Size Units Type Units	*impacted by (impacts)*	**Risk**
Link	See Section 1.3	*impacted by (impacts)*	**Risk**
Organization	See Section 1.6	*assigned to (responsible for)*	**Risk**
		impacted by (impacts)	**Risk**
MitigationActivity	Description Duration End Date Planned End Date Planned Start Date Result Start Date Status	*generated (generated by)*	**ChangeRequestPackage**
		mitigates (mitigated by)	**Risk**
ProgramActivity	Description Duration End Date Number Planned End Date Planned Start Date Start Date	*causes (caused by)*	**Risk**
		impacted by (impacts)	**Risk**
ProgramElement	Contract Number Cost Description End Date Labor Hours Non-recurring Cost Number Start Date Type	*causes (caused by)*	**Risk**
		impacted by (impacts)	**Risk**
Requirement	See Section 1.2	*impacted by (impacts)*	**Risk**

Table 8 Risk Management

Entity Class	Attributes	Relations	Target Classes
Risk	See Section 1.6	*assigned to (responsible for)*	**Organization**
		augmented by (augments)	**ExternalFile**
		caused by (causes)	**Organization** **ProgramActivity** **ProgramElement** **Requirement**
		documented by (documents)	**Document**
		generates (generated by)	**ChangeRequestPackage**
		impacts (impacted by)	**Component** **Port** **Function** **Item** **Link** **Organization** **ProgramActivity** **ProgramElement** **Requirement**
		leads to (comes from)	**Risk**
		mitigated by (mitigates)	**MitigationActivity**

1.8 Generate Mitigation Activities

For those **Risks** that are to be handled via mitigation, GENESYS provides a class called **MitigationActivity**. A **MitigationActivity** is an action performed to reduce either the probability of occurrence or consequence/impact of an uncertainty element. A **MitigationActivity** may mitigate one or more risks. It may *impact* a **Component, Requirement**, **Function**, **Link**, **ProgramActivity**, **ProgramElement** or **Organization**. It may *result in* a new or updated **Requirement** or **Function**.

Often, programs require that mitigation activities be captured as tasks or activities in the program's Integrated Master Schedule. In GENESYS, **MitigationActivities** can be exported to MS Project via the GENESYS MS Project Connector.

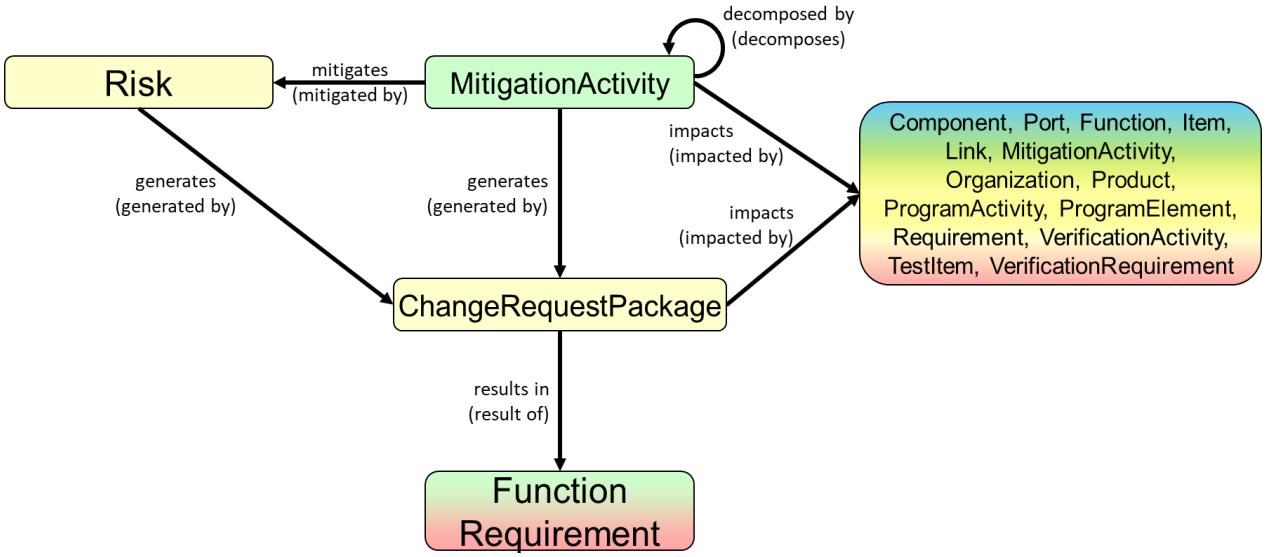


Figure 10 Mitigation Activities

Table 9 Mitigation Activities

Entity Class	Attributes	Relations	Target Classes
ChangeRequestPackage	See Section 1.7	*generated by (generates)*	**MitigationActivity** **Risk**
		impacts (impacted by)	**Component** **Port** **Function** **Item** **MitigationActivity** **Organization** **Product** **ProgramActivity** **ProgramElement** **Requirement** **VerificationActivity** **TestItem** **VerificationRequirement**
		results in (result of)	**Function** **Requirement**
Component	See Section 1.1	*impacted by (impacts)*	**MitigationActivity**
Port	See Section 1.7	*impacted by (impacts)*	**MitigationActivity**
Function	See Section 1.6	*impacted by (impacts)*	**MitigationActivity**
		result of (result in)	**ChangeRequestPackage**

Table 9 Mitigation Activities

Entity Class	Attributes	Relations	Target Classes
Item	See Section 1.7	*impacted by (impacts)*	**MitigationActivity**
Link	See Section 1.3	*impacted by (impacts)*	**MitigationActivity**
MitigationActivity	See Section 1.7	*decomposed by (decomposes)*	**MitigationActivity**
		generated (generated by)	**ChangeRequestPackage**
		impacted by (impacts)	**MitigationActivity**
		mitigates (mitigated by)	**Risk**
Organization	See Section 1.6	*impacted by (impacts)*	**MitigationActivity**
Product	Accuracy Description Doc. PUID Fields Precision Priority Range Size Size Units Type Units	*impacted by (impacts)*	**MitigationActivity**
ProgramActivity	See Section 1.7	*impacted by (impacts)*	**MitigationActivity**
ProgramElement	See Section 1.7	*impacted by (impacts)*	**MitigationActivity**
Requirement	See Sections 1.2 and 1.5	*impacted by (impacts)*	**MitigationActivity**
		result of (results in)	**ChangeRequestPackage**
Risk	See Section 1.6	*mitigated by (mitigates)*	**MitigationActivity**
VerificationActivity	Completion Criteria Description Duration End Date Prerequisite Special Comments Start Date Timeout Type	*impacted by (impacts)*	**MitigationActivity**

Table 9 Mitigation Activities

Entity Class	Attributes	Relations	Target Classes
TestItem	Description Priority Size Size Units Type	*impacted by* *(impacts)*	**MitigationActivity**
VerificationRequirement	Description Doc. PUID Level Method Number Status	*impacted by* *(impacts)*	**MitigationActivity**

1.9 Characterize Requirements and Categorize Constraints

Requirements may also be characterized as one of the following:

- Constraint (i.e., limitation on the design or construction of the system)
- Functional (i.e., what the system must do)
- Incentive Award Fee Criterion (i.e., programmatic, or other requirements affecting a contractor's fees for meeting or exceeding the requirement)
- Performance (i.e., how well the system or function must perform)
- Programmatic (i.e., program management constraints)
- Test (i.e., test constraints)
- Verification (i.e., acceptance criteria).

This aspect of a requirement is captured in the repository by setting the **Requirements** Type attribute to the appropriate value. If a determination cannot be made, parse the **Requirement** into a set of **Requirements** where each **Requirement** is only one of the seven types.

Link the system-level physical constraint **Requirements** to the system **Component** using the *specifies* relation. As the system component hierarchy evolves, a constraint **Requirement** should be linked to all of the **Components** to which it applies (i.e., a constraint **Requirement** may apply to the descendants of a **Component** as well as the **Component**). See Section 0 for a discussion of constraints on lower-level **Components** and constraint hierarchies.

Constraint Categorization. For each **Requirement** that is a constraint, categorize it by a **Category** that represents the appropriate requirements domain, such as Reliability, Transportability, Electromagnetic Radiation, etc. These domains correspond to the non-functional leaf-level **Requirements** sections typically found in a System/Segment Specification or other specification.

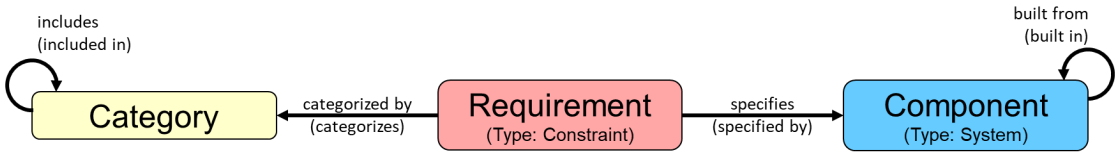


Figure 11 Constraint Requirements

Table 10 Constraint Requirements

Entity Class	Attributes	Relations	Target Classes
Category	Description	*categorizes (categorized by)*	**Requirement** (Type: Constraint)
		included in (includes)	**Category**
Component (Type: System)	See Section 1.1	*specified by (specifies)*	**Requirement** (Type: Constraint)
Requirement (Type: Constraint)	See Section 1.2	*categorized by (categorizes)*	**Category**
		specifies (specified by)	**Component** (Type: System)

BEHAVIORAL ANALYSIS

1.10 Identify States (if needed)

For some projects, the customer expects to represent behavior using **States** and **Modes** rather than using Functions. A **State** identifies a non-overlapping (i.e., one **State** does not share its behavior with another **State**) behavioral and possibly repetitive condition occurring during a component's operating lifetime. In other words, the set of **States** *exhibited by* a **Component** are complete for expressing a **Component**'s behavior including its timing. Alternative **State** representations are possible, but each set definition must be complete and non-overlapping (i.e., the **State** universe may be partitioned in more than way, but each partition needs to be complete and unique).

A **State** may exist either because it is *documented by* a **Document** or *specified by* a **Requirement**. An **ExternalFile** or **Text** entity may also *augment* a **State** for the purpose of further enhancing the meaning or representation of the **State**.

A given **State** may be a member of a particular subset of **States**. The collection of such **States** is represented as a **Mode**; this is shown as the **State** *encompassed by* a **Mode**. A **Component** *exhibits* a **State** and also *contains* a **Mode**. Each **State** *incorporates* one or more **Functions**. Associated with the *incorporates* relation are two attributes – Entry and Exit. The relationship attribute "Entry" set to True indicates the behavior is performed upon entry into the State. The relationship attribute "Exit" set to True indicates the behavior is performed immediately before exiting the State. A target of the relationship where the Behavior Type attribute value of "Integrated (Root)" indicates behavior that is performed once the "Entry" behavior completes and continues until it finishes or the State exits.

One or more subordinate **States** may *decompose* a parent **State**, which delineates the progression from a composite **State** to an atomic **State** (an atomic **State** is identified by the absence of targets for the *decomposed by* relation). The movement from one **State** to another **State** occurs through a **Transition**. A **State** is *exited by* a **Transition** and correspondingly, the **Transition** *enters* a new **State** or may re-enter the same **State**. However, the timing of the **Transition**'s effect is governed by a Guard Condition attribute. The Guard Condition attribute is a rule, which may be empty, simple, or complex. It evaluates to a Boolean value. (An empty Guard Condition is not evaluated, so that the new state is entered without impediment). If the Guard Condition is true, the transition occurs; otherwise, the transition waits for the Guard Condition to change from false to true at which point the new **State** is entered.

Events serve to communicate to external **State** machines at the time point of a **Transition**. A **Transition** is *triggered by* an **Event** and an **Event** is *responsible for* an **Item**, which conveys the message governed by the **Event**.

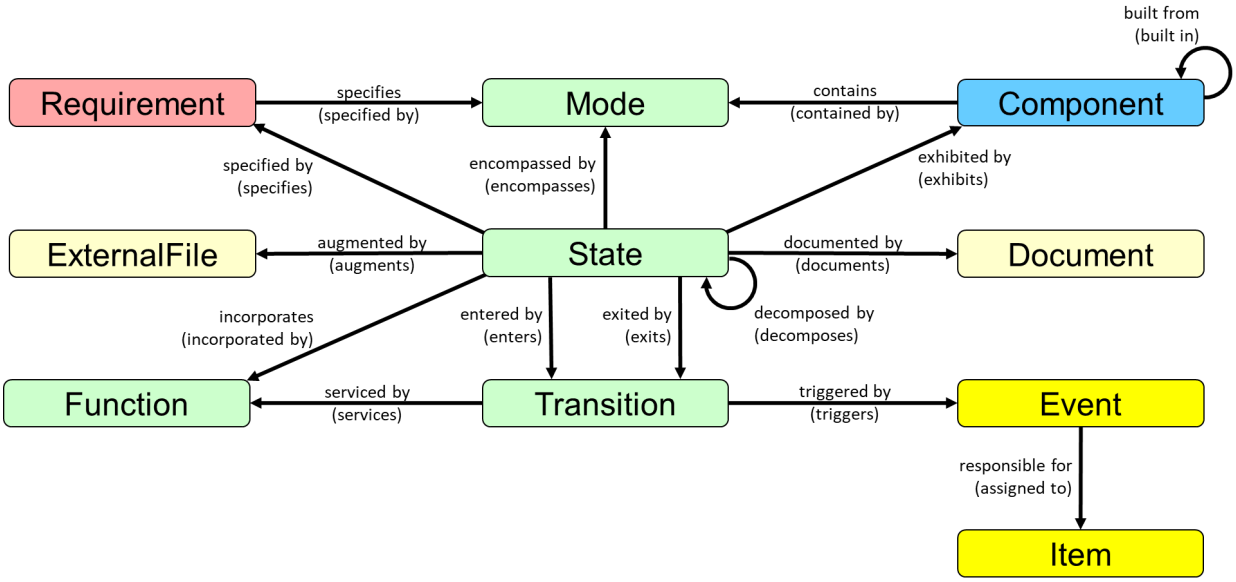


Figure 12 States View

Table 11 States View

Entity Class	Attributes	Relations	Target Classes
Component (Type: System)	See Section 1.1	*built from (built in)*	**Component**
		contains (contained by)	**Mode**
		exhibits (exhibited by)	**State**
Document	See Section 1.2	*documents (documented by)*	**Mode** **State**
Event	Description	*documented by (documents)*	**Document**
		responsible for (assigned to)	**Item**
		triggers (triggered by)	**Transition**
ExternalFile	See Section 1.2	*augments (augmented by)*[9]	**State**
Function	See Section 1.6	*incorporated by (incorporates)*	**State**
		services (serviced by)	**Transition**
Item	See Section 1.7	*assigned to (responsible for)*	**Event**
Mode	Description Number	*contained by (contains)*	**Component**
		documented by (documents)	**Document**

[9] The Position attribute of this relationship should be set to control the order in which multiple external files are appended to the requirement description when it is output in formal documentation generated from the repository.

Table 11 States View

Entity Class	Attributes	Relations	Target Classes
		encompasses (encompassed by)	**State**
		specified by (specifies)	**Requirement**
Requirement	See Sections 1.2, 1.5, and 1.9	*specifies (specified by)*	**Mode** **State**
State	Description Doc. PUID Number Title	*augmented by (augments)*	**ExternalFile**
		decomposed by (decomposes)	**State**
		documented by (documents)	**Document**
		encompassed by (encompasses)	**Mode**
		entered by (enters)	**Transition**
		exhibited by (exhibits)	**Component**
		exited by (exits)	**Transition**
		incorporates (incorporated by)	**Function**
		specified by (specifies)	**Requirement**
Transition	Delay Delay Units Description Guard Number	*documented by (documents)*	**Document**
		enters (entered by)	**State**
		exits (exited by)	**State**
		triggered by (triggers)	**Event**

1.11 Use Cases (if needed)

On some projects, use cases are used instead of threads or scenarios. In other instances, use cases are precursors to the development of system requirements, which lead to the development of threads or scenarios. Identify any system use cases and instantiate them by creating **UseCase** entities. A **UseCase** entity *describes* a **Component** to which the use case is applicable. A **UseCase** entity *involves* a **Component** fulfilling the role of an actor in the use case. A **UseCase** entity is *elaborated by* either a **Function** entity, a **ProgramActivity** entity or a **VerificationActivity** entity depending upon whether the use case affects a system behavior, program management behavior, or test behavior.[10] A **UseCase** entity may be *extended by* a **UseCase** to add additional specificity to the use case. A **UseCase** entity *elicits* a **Requirement**. An external use case diagram may be referenced in the repository by creating an **ExternalFile** entity that *augments* a **UseCase** entity.

[10] In this instance, the resulting behaviorType of a Function, ProgramActivity, or VerificationActivity would be set to "Thread."

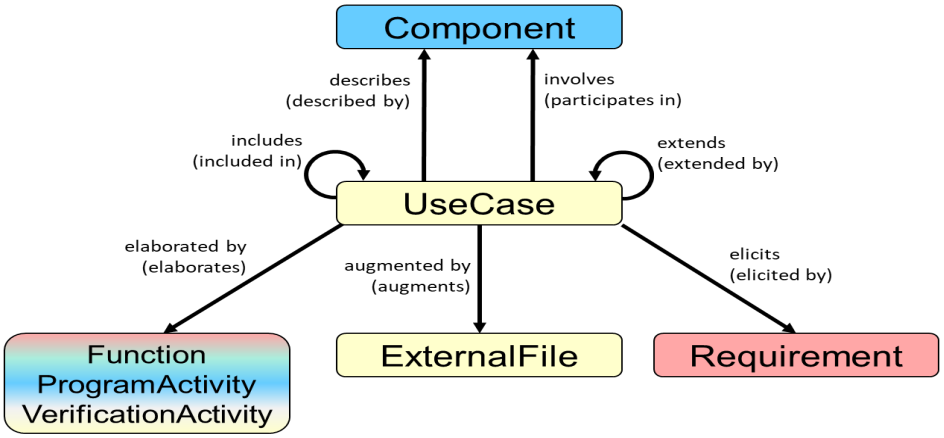


Figure 13 Use Case Application

Table 12 Use Case Application

Entity Class	Attributes	Relations	Target Classes
Component	See Section 1.1	*described by (describes)*	**UseCase**
		participates in (involves)	**UseCase**
ExternalFile	See Section 1.2	*augments (augmented by)*[11]	**UseCase**

[11] The Position attribute of this relationship should be set to control the order in which multiple external files are appended to the requirement description when it is output in formal documentation generated from the repository.

Table 12 Use Case Application

Entity Class	Attributes	Relations	Target Classes
Function	See Section 1.6	*elaborates (elaborated by)*	**UseCase**
ProgramActivity	See Section 1.7	*elaborates (elaborated by)*	**UseCase**
Requirement	See Sections 1.2 and 1.5	*elicited by (elicits)*	**UseCase**
VerificationActivity	See Section 1.8	*elaborates (elaborated by)*	**UseCase**
UseCase	Alternate Flow Description Number Postconditions Preconditions Primary Flow	*augmented by (augments)*[7]	**ExternalFile**
		describes (described by)	**Component**
		elaborated by (elaborates)	**Function** **ProgramActivity** **VerificationActivity**
		extends (extended by)	**UseCase**
		includes (included in)	**UseCase**
		involves (participates in)	**Component**

1.12 Develop the System Behavioral Hierarchy

Behavioral analysis in GENESYS begins with defining the major threads through the system and culminates in an integrated behavior view of the system and actions *allocated to* subcomponents. For the system, a top-level **Function** should be defined and *allocated to* the **Component** of type System. The attribute behaviorType should be set to “Integrated (Root)” to identify that this top-level **Function** represents the totality of functionality performed by the system and is decomposed (hierarchically) into all of the functions performed by the system. The root **Function** is decomposed into the primary **Functions** of the system.

Function Traceability. If a **Function** is identified in direct response to an originating **Requirement** or to a **Concern** decision, the **Function** should be linked to the causal originating entities, using either the *based on (basis of)* or *result of (results in)* relations. Thus, establishing requirements traceability beyond the **Function** hierarchy. This also supports the use of **Functions** as requirements, i.e., the inclusion of “shall” in the **Function** Description.

State Mapping. If a **State** entity has been defined (See Section 1.10), it should be linked to the first-level (i.e., non-root level) **Functions** using the *incorporates* relation to identify which **Functions** and their descendants are included in each **State**. Some of these lower–level **Functions** *services* **Transition** entities.

Function Allocation. In conjunction with Physical Architecture Synthesis (See Section 1.15), for each layer of **Components**, **Functions** are decomposed until they can be uniquely allocated to the next level of **Component** in the component hierarchy. The allocation of these functions are considered standard, that is, non-root. When generating written reports, this functional hierarchy and allocation provides the organizational foundation for the assignment of performance **Requirements** in a **Component** specification and the **Functions** performed (i.e., **Function** *specified by* **Requirement**).

Function Inputs and Outputs. For each **Function** in the evolving functional hierarchy, input and output **Items** are identified and associated using the relations: *input to (inputs)* and *output from (outputs)*. When **Functions** are allocated in conjunction with Physical Architecture Synthesis (See Section 1.15), these **Items** form part of the definition of the component interfaces (See Sections 1.16 and 1.17). As with **Functions**, **Items** should be aggregated to simplify presentation.

Note: When doing behavior development, a root **Function** can be established for any **Component** and the behavior diagram built using the allocated **Functions** to define the full behavior of the **Component** from the **Component**'s perspective rather than from the system's perspective. These lower-level root **Functions** do not appear in the system functional hierarchy, but act as access points into the hierarchy. Reports in GENESYS use either root or atomic **Functions**, whichever allocation is present.

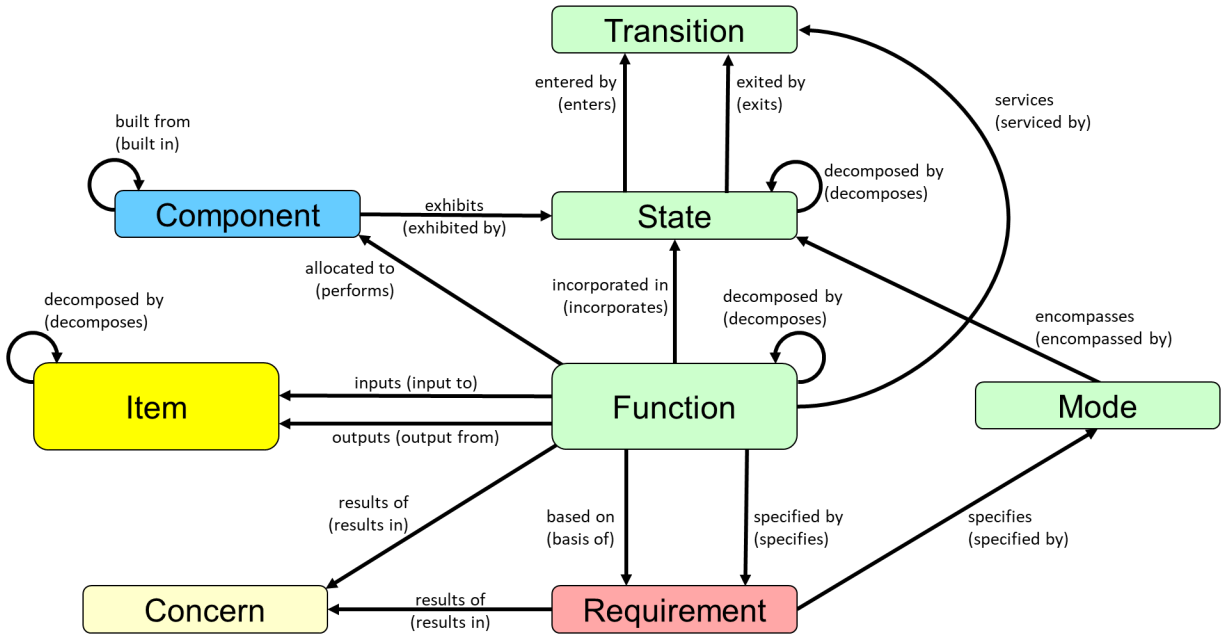


Figure 14 Behavioral Decomposition

Table 13 Behavioral Decomposition

Entity Class	Attributes	Relations	Target Classes
Component (Type: System)	See Section 1.1	*built from (built in)*	**Component**
		exhibits (exhibited by)	**State**
		performs (allocated to)	**Function**
Function	See Section 1.10 (Behavior Type: Integrated (Root))[12]	*allocated to (performs)*	**Component**
		based on (basis of)	**Requirement**
		decomposed by (decomposes)	**Function**
		inputs (input to)	**Item**

[12] A **Component** should have only one root **Function**.

Table 13 Behavioral Decomposition

Entity Class	Attributes	Relations	Target Classes
		outputs (output from)	**Item**
		result of (results in)	**Concern**
Concern	See Section 1.6	*results in (result of)*	**Function** **Requirement**
Item	See Section 1.7	*decomposed by (decomposes)*	**Item**
		input to (inputs)	**Function**
		output from (outputs)	**Function**
Mode	See Section 1.10	*encompasses (encompassed by)*	**State**
		specified by (specifies)	**Requirement**
Requirement	See Sections 1.2 and 1.5	*basis of (based on)*	**Function**
State	See Section 1.10	*decomposed by (decomposes)*	**State**
		encompassed by (encompasses)	**Mode**
		entered by (enters)	**Transition**
		exhibited by (exhibits)	**Component**
		exited by (exits)	**Transition**
Transition	See Section 1.10	*enters (entered by)*	**State**
		exits (exited by)	**State**
		serviced by (services)	**Function**

1.13 Refine and Allocate Functional Performance Requirements

As the functional hierarchy is developed, decompose, and allocate performance **Requirements** to **Functions**. This may be a complex process, particularly if it involves a domain change or trade studies that assimilate multiple performance **Requirements** to reach a design decision. The result of the design decision, captured as a **Requirement** whose Origin attribute is set to Design Decision, may result in multiple **Functions** and performance **Requirements** as well as constraint **Requirements**. If this is a major design decision, it should be augmented with a **Concern** to capture **Concern**-type information that is not normally captured by a **Requirement**.

Since **Functions** may be aggregated to enhance understanding, not every **Function** will have performance **Requirements**; however, **Functions** allocated to a **Component** should have performance **Requirements** to clearly define how well the functions must be performed in terms of such characteristics as timing and accuracy. Performance **Requirements** are inseparable from their associated **Functions**. Thus, only the **Function** is *allocated to* a **Component** (i.e., the performance **Requirement** for an allocated **Function** should not be linked with the *specifies* relation to the performing **Component**).

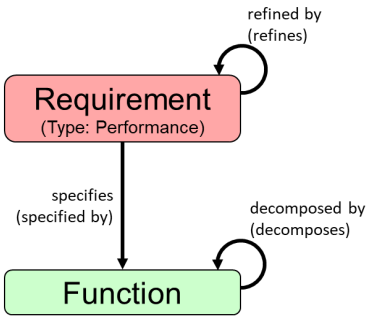


Figure 15 Performance Requirements

Table 14 Performance Requirements

Entity Class	Attributes	Relations	Target Classes
Function	See Section 1.6	*decomposed by (decomposes)*	**Function**
		specified by (specifies)	**Requirement**
Requirement	See Sections 1.2 and 1.5	*refined by (refines)*	**Requirement**
		specifies (specified by)	**Function**

1.14 Capture Behavioral and Performance Concerns and Risks

While developing the system's functional hierarchy and deriving the associated performance **Requirements**, additional **Concerns** and **Risks** may be identified. They should be captured in the repository in a manner analogous to **Concerns** and **Risks** resulting from the analysis of originating **Requirements** (See Section 1.6).

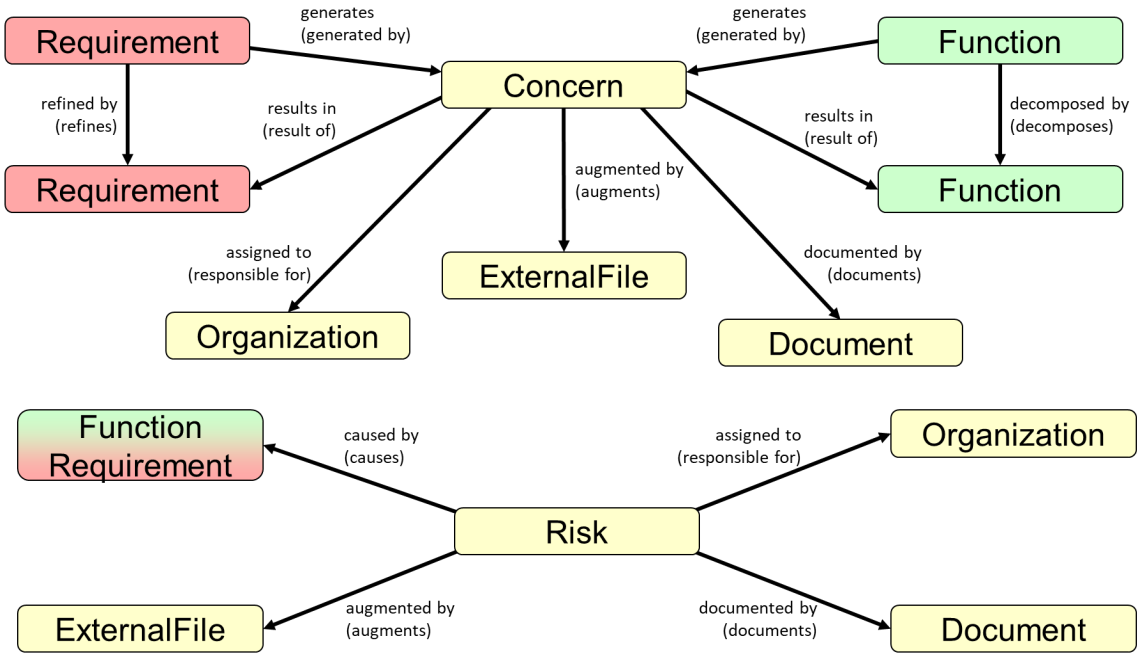


Figure 16 Functional or Performance Requirement Concern and Risk

Table 15 Functional or Performance Requirement Concern and Risk

Entity Class	Attributes	Relations	Target Classes
Document	See Section 1.2	*documents (documented by)*	**Concern Risk**
ExternalFile	See Section 1.2	*augments (augmented by)*	**Concern Risk**
Function	See Section 1.6	*decomposed by (decomposes)*	**Function**
		causes (caused by)	**Risk**
		generates (generated by)	**Concern**
		result of (results in)	**Concern**
Concern	See Section 1.6	*assigned to (responsible for)*	**Organization**
		documented by (documents)	**Document**
		generated by (generates)	**Function Requirement**
		results in (result of)	**Function Requirement**
Organization	See Section 1.6	*responsible for (assigned to)*	**Concern Risk**
Requirement	See Sections 1.2 and 1.5	*causes (caused by)*	**Risk**
		generates (generated by)	**Concern**
		refined by (refines)	**Requirement**
		result of (results in)	**Concern**
Risk	See Section 1.6	*assigned to (responsible for)*	**Organization**
		augmented by (augments)	**ExternalFile**
		caused by (causes)	**Function Requirement**

PHYSICAL ARCHITECTURE SYNTHESIS

1.15 Allocate Functions to Next Level of Components

In conjunction with the analysis of requirements, behavioral decomposition, and assessment of component technology, the process activity identifies the next layer of **Components** in the system component hierarchy.

As the component hierarchy evolves, **Functions** are *allocated to* **Components**. This allocation is performed in layers. When a decomposed **Function** is allocated to a **Component**, all lower-level **Functions** in its decomposition become part of the **Component**'s behavior. The **Component** may be further decomposed, in which case even lower-level **Functions** are allocated to the lower-level **Components**. At the leaf-level, these allocations are termed Standard. Since **Functions** can be

aggregated to enhance understanding, there is not a one-to-one correspondence between levels in the **Function** hierarchy and levels in the **Component** hierarchy.

Note: As stated in Section 1.11, when doing behavior development, a root **Function** may be established for any **Component** and the behavior diagram built using the allocated atomic **Functions.** This defines the full behavior of the **Component** from the **Component's** perspective rather than from the system perspective. These lower-level root **Functions** for lower-level **Components** do not appear in the system functional hierarchy, but act as access points into the hierarchy.

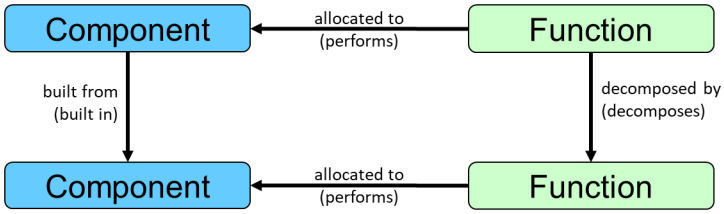


Figure 17 Component Hierarchy and Function Allocation

Table 16 Component Hierarchy and Function Allocation

Entity Class	Attributes	Relations	Target Classes
Component	See Section 1.1	*built from (built in)*	**Component**
		performs (allocated to)	**Function**
Function	See Section 1.6	*allocated to (performs)*	**Component**
		decomposed by (decomposes)	**Function**

1.16 Refine External Interface Definitions

An external interface entity identifies the fact that the system communicates in some manner with an external **Component** (See Section 1.3). Other details of the interface are captured in **Link** entity definitions. The **Link** class is decomposable using the *includes* (*included in)* relation pair. As the system component hierarchy evolves, the terminus point for **Links** must change. One may use a child **Link** to connect to the correct lower-level **Components.** Thus, using the child connections simplifies the maintenance of the connections as the interface design proceeds.[13] Otherwise, the systems engineer must disconnect and reconnect the **Link** to the lower-level component. This allows **Links** to retain their conceptual identity even though their child connection end points change as the component hierarchy grows in depth. The components that provide the **Items** for the **Links** are determined by the functional allocation.

Links may be *specified by* performance and constraint **Requirements**. Only the lowest layer of **Items** should be *transferred by* a **Link**.

[13] If the remaining terminus of the **Link** does not change, which may occur with an external **Component**, the child **Link** will coincidently terminate with the parent **Link**.

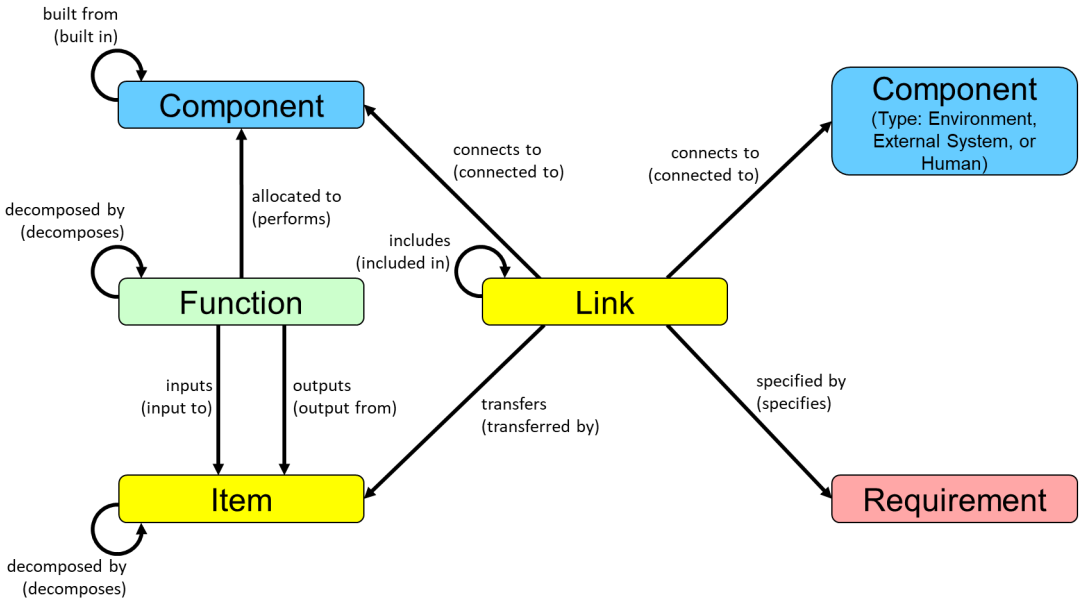


Figure 18 External Interface Definition

Table 17 External Interface Definition

Entity Class	Attributes	Relations	Target Classes
Component	See Section 1.1	*built from (built in)*	**Component**
		connected to (connects to)	**Link**
		performs (allocated to)	**Function**
Function	See Section 1.6	*allocated to (performs)*	**Component**
		decomposed by (decomposes)	**Function**
		inputs (input to)	**Item**
		outputs (output from)	**Item**
Item	See Section 1.7	*decomposed by (decomposes)*	**Item**
		input to (inputs)	**Function**
		output from (outputs)	**Function**
		transferred by (transfers)	**Link**
Link	See Section 1.3	*connects to (connected to)*	**Component**
		includes (included in)	**Link**
		specified by (specifies)	**Requirement**
		transfers (transferred by)	**Item**
Requirement (Type: Performance or Constraint)	See Sections 1.2 and 1.5	*specifies (specified by)*	**Link**

1.17 Derive or Refine Internal Interfaces

Within the system hierarchy, the allocation of **Functions** to **Components** establishes the internal interfaces of the system based on the **Items** that flow between the allocated **Functions**. The internal interfaces are also formalized in the repository using the **Link** entity class.

As described in Section 1.14, the terminus point for **Links** must change as lower-level **Components** are introduced. Since the **Link** class is decomposable, one may use, as appropriate, a child **Link** to connect to the correct lower-level **Components.** Thus, using the child connections simplifies the maintenance of the connections as the interface design proceeds.[14] Otherwise, the systems engineer must disconnect and reconnect the **Link** to the lower-level component. This allows **Links** to retain their conceptual identity even though their child connection end points change as the component hierarchy grows in depth. The components that provide the **Items** for the **Links** are determined by the functional allocation. This also allows the content of formal higher-level specifications to remain unchanged as the **Link** connection points move deeper into the system component hierarchy.

Links may be *specified by* performance and constraint **Requirements**. Only the lowest layer of **Items** should be *transferred by* a **Link**.

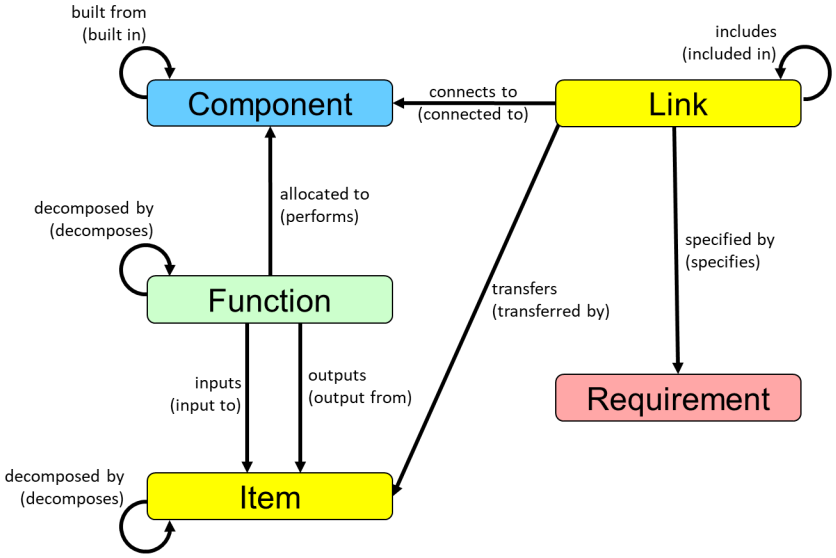


Figure 19 Internal Interface Definition

Table 18 Internal Interface Definition

Entity Class	Attributes	Relations	Target Classes
Component	See Section 1.1	*built from (built in)*	**Component**
		connected to (connects to)	**Link**
Function	See Section 1.6	*allocated to (performs)*	**Component**
		decomposed by (decomposes)	**Function**
		inputs (input to)	**Item**

[14] When dealing with Internal Interfaces, the remaining terminus of the **Link** does change with internal connections. Therefore, the child **Link** will terminate with a corresponding child **Component**.

Table 18 Internal Interface Definition

Entity Class	Attributes	Relations	Target Classes
		outputs (output from)	**Item**
Item	See Section 1.7	*decomposed by (decomposes)*	**Item**
		input to (inputs)	**Function**
		output from (outputs)	**Function**
		transferred by (transfers)	**Link**
Link	See Section 1.3	*connects to (connected to)*	**Component**
		includes (included in)	**Link**
		specified by (specifies)	**Requirement**
		transfers (transferred by)	**Item**
Requirement (Type: Performance or Constraint)	See Sections 1.2 and 1.5	*specifies (specified by)*	**Link**

1.17.1 Ports

Ports provide identification of the place where entities can connect to and interact with a specified component block. At the upper, abstract levels of an architecture, we generally do not specify the nature of a port, and the port is simply called a “port”. However, at lower levels of the physical architecture we may need to refine and specify distinct behavior and requirements for individual connection points to a configuration item.

Components may have several connection points (or ports) available for interfacing to other components in the architecture. Ports are places where other components can connect to and interact with a component. In GENESYS we have the ability to refine the definition of ports and their properties using the definitions of SysML Version 1.4.

A required interface on a port specifies an operation required by the component to realize its behavior. A provided interface on a port specifies the operation that the component provides on the interface. Required and provided interfaces are shown on an internal block diagram using the ball-and-socket notation. The ball notation indicates a provided interface and the socket notation indicates a required interface This notation used the *provided by/provides* and *required by/requires* relations. A **Port** that *provides* a **PortDefinition** will be shown on the Flow Internal Block as the ball notation connected to the port; a **Port** that *requires* a **PortDefinition** will be shown as socket notation connected to the port.

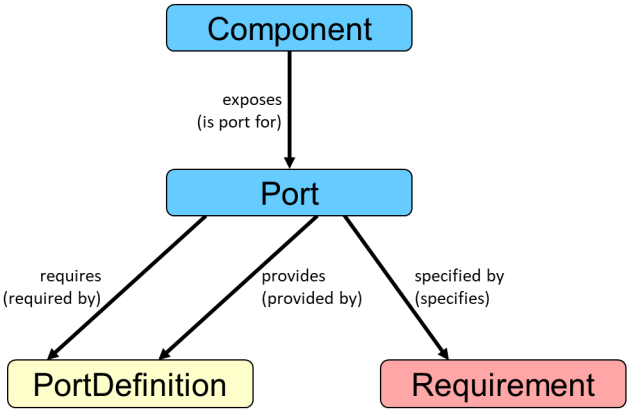


Figure 20 Port Definitions

Table 19 Port Definitions

Entity Class	Attributes	Relations	Target Classes
Port	See Section 1.7	*augmented by (augments)*	**ExternalFile** **Text**
		categorized by (categories)	**Category**
		causes (caused by)	**Risk**
		documented by (documents)	**Document**
		generalization of (kind of)	**Port**
		generates (generated by)	**Nexus**
		exposes (is port for)	**Port**
		impacted by (impacts)	**Nexus** **Risk**
		is port for (exposes)	**Component** **Port**
		kind of (generalization of)	**Port**
		packaged by (packages)	**Package**
		provides provided by)	**PortDefinition**
		requires (required by)	**PortDefinition**
		specified by (specifies)	**Requirement**
PortDefinition	Description Operations	*augmented by (augments)*	**External File** **Text**
		categorized by (categorizes)	**Category**

Table 19 Port Definitions

Entity Class	Attributes	Relations	Target Classes
		documented by (documents)	**Document**
		packaged by (packages)	**Package**
		provided by (provides)	**Port**
		required by (requires)	**Port**

1.18 Assign/Derive Constraints for Components

Based on the constraint **Requirements** allocated to a parent **Component**, constraint **Requirements** are derived for the subcomponents. This can be a simple flow-down of the same requirement or may be a budgeting of a limiting constraint, such as weight, between subcomponents.

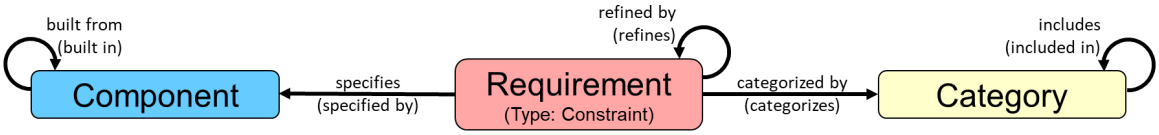


Figure 21 Component Constraint Requirements

Table 20 Component Constraint Requirements

Entity Class	Attributes	Relations	Target Classes
Category	See Section 1.9	*categorizes (categorized by)*	**Requirement** (Type: Constraint)
		includes (included in)	**Category**
Component (Type: System)	See Section 1.1	*built from (built in)*	**Component**
		specified by (specifies)	**Requirement (Type: Constraint)**
Requirement (Type: Constraint)	See Sections 1.2 and 1.5	*categorized by (categorizes)*	**Category**
		refined by (refines)	**Requirement** (Type: Constraint)
		specifies (specified by)	**Component** (Type: System)

1.19 Capture Physical Architecture Concerns and Risks

While developing the physical architecture and deriving interfaces and performance/constraint **Requirements**, additional concerns and risks may be identified. These should be captured in the repository in a manner analogous to **Concerns** and **Risks** resulting from the analysis of originating **Requirements** (See Section 1.6 and 1.7).

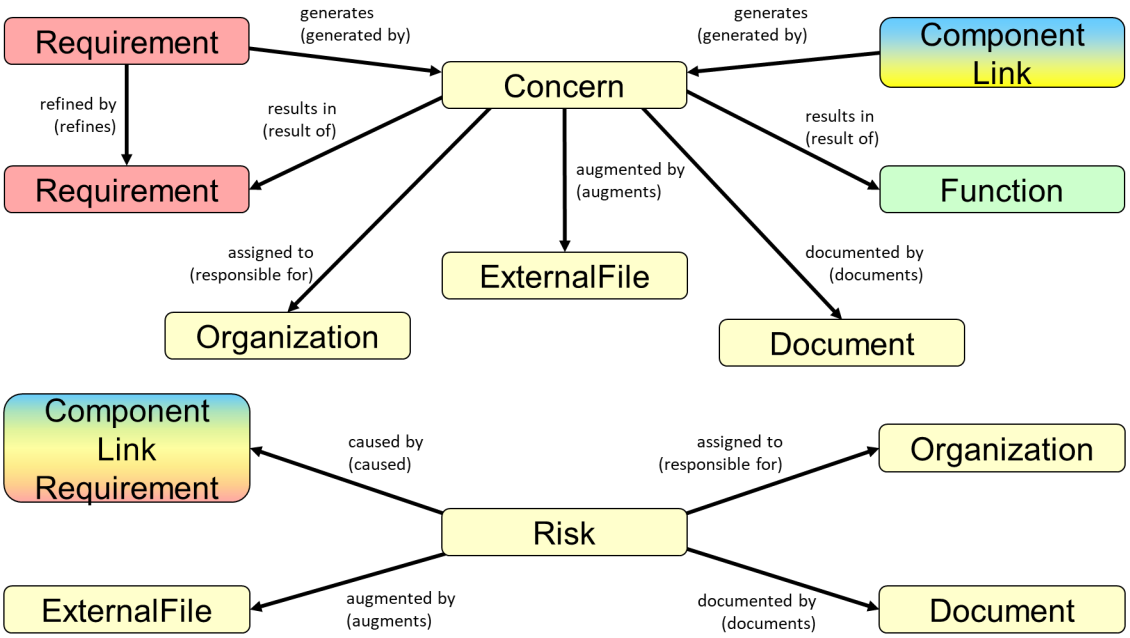


Figure 22 Physical Architecture Concern and Risk

Table 21 Physical Architecture Concern or Risk

Entity Class	Attributes	Relations	Target Classes
Component	See Section 1.1	*causes (caused by)*	**Risk**
		generates (generated by)	**Concern**
Concern	See Section 1.6	*assigned to (responsible for)*	**Organization**
		documented by (documents)	**Document**
		generated by (generates)	**Component** **Link** **Requirement**
		results in (result of)	**Function** **Requirement**
Document	See Section 1.2	*documents (documented by)*	**Concern** **Risk**
ExternalFile	See Section 1.2	*augments (augmented by)*	**Concern** **Risk**
Function	See Section 1.6	*result of (results in)*	**Concern**
Link	See Section 1.3	*causes (caused by)*	**Risk**
		generates (generated by)	**Concern**
Organization	See Section 1.6	*responsible for (assigned to)*	**Concern** **Risk**
Requirement		*causes (caused by)*	**Risk**

Table 21 Physical Architecture Concern or Risk

Entity Class	Attributes	Relations	Target Classes
	See Sections 1.2 and 1.5	*generates (generated by)*	**Concern**
		refined by (refines)	**Requirement**
		refines (refined by)	**Requirement**
		result of (results in)	**Concern**
Risk	See Section 1.6	*assigned to (responsible for)*	**Organization**
		augmented by (augments)	**ExternalFile**
		caused by (causes)	**Component** **Link** **Requirement**
		documented by (documents)	**Document**

VERIFICATION/VALIDATION

1.20 GENESYS Simulator

The simulator is a discrete event simulator that executes the behavioral and link viewpoints to provide an assessment of system performance, resource levels, and to verify the dynamic integrity of the conceptual view. The simulator dynamically interprets a behavior viewpoint (i.e., the Enhanced Functional Flow Block Diagram [EFFBD] or Activity Diagram [AD]) in conjunction with the **Component**'s link view. It also identifies and displays timing, resource utilization, link flow, and viewpoint inconsistencies. The correction of any inconsistencies usually results in a re-expression of derived **Requirements**, which in turn affects verification and validation of the system. The simulator usage should be an integral part of behavioral analysis and physical architecture synthesis to assure dynamic consistency of the system's requirements.

1.21 Establish Verification Requirements

For each specified **Component**, including the system, establish how each leaf-level **Function** and **Requirement** is to be verified. This information is captured in the repository using **VerificationRequirements**. **VerificationRequirements** can range from requirements on acceptance testing such as a qualification test to verification of individual **Requirements** and **Functions**. Related VerificationRequirements can be grouped into a VerificationGroup. A single **VerificationRequirement** may verify multiple leaf-level **Requirements** and **Functions**. Conversely, a single **Requirement** or **Function** may be verified by multiple **VerificationRequirements.** A **VerificationRequirement** should be established for each **Verification** method that is to be used to verify a **Function** or **Requirement**. For example, if a test and subsequent analysis must be performed to verify a **Requirement**, a **VerificationRequirement** should be written for the test and a **VerificationRequirement** should be written for the analysis. This allows each to be assigned to the proper **VerificationActivity** and **VerificationEvent** (see Section 1.22) and assigned to the proper executing **Organization**.

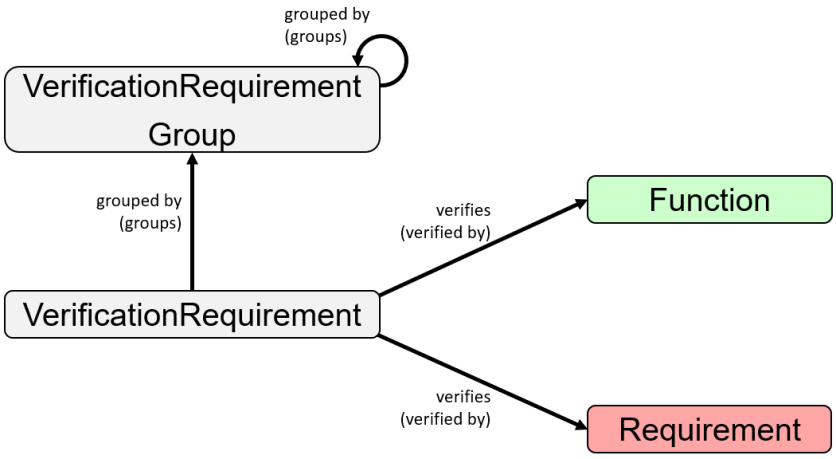


Figure 23 Verification Requirements

Table 22 Verification Requirements

Entity Class	Attributes	Relations	Target Classes
Function	See Section 1.6	*verified by (verifies)*	**VerificationRequirement**
Requirement	See Sections 1.2 and 1.5	*verified by (verifies)*	**VerificationRequirement**
VerificationRequirement	See Section 1.8	*verifies (verified by)*	**Function** **Requirement**
VerificationGroup	Description Name Number Doc. PUID Status Title	*grouped by (groups)*	**VerificationGroup**
		groups	**VerificationRequirement**

1.22 Establish Verification Events

The programmatic entities that capture the verification effort are summarized in the repository as **VerificationEvents**. **VerificationActivities** represent the executable verification steps and expected results used by a particular VerificationEvent. **TestConfigurations** identify the equipment and facilities needed for particular **VerificationActivities**. A **TestConfiguration** identifies **Components** of the system and **Links** to the **Components** under test, support **Components**, as well as test equipment and test support software. As **VerificationActivities** are planned and conducted, the **VerificationRequirement's** Status attribute is updated in the repository. A **VerificationActivity** may cause **Risks** or generate **Concerns** if the activity is anticipated or doesn't go as planned.

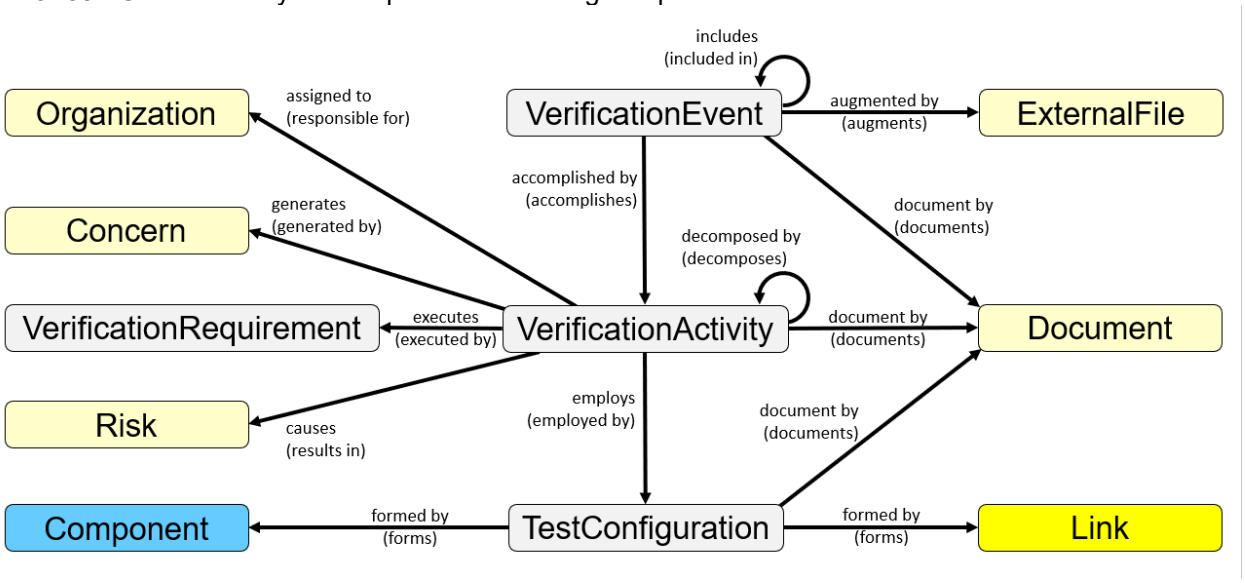


Figure 24 Verification Planning

Table 23 Verification Planning

Entity Class	Attributes	Relations	Target Classes
Component	See Section 1.1	*forms (formed by)*	**TestConfiguration**
Concern		*generated by (generates)*	**VerificationActivity**

Table 23 Verification Planning

Entity Class	Attributes	Relations	Target Classes
Document	See Section 1.2	*documents (documented by)*	**VerificationActivity TestConfiguration VerificationEvent**
ExternalFile	See Section 1.2	*augments (augmented by)*	**TestConfiguration VerificationEvent**
Link	See Section 1.3	*forms (formed by)*	**TestConfiguration**
Organization	See Section 1.6	*responsible for (assigned to)*	**VerificationActivity**
Risk		*caused by (causes) results in (result of)*	**VerificationActivity**
TestConfiguration	Description Number	*documented by (documents)*	**Document**
		employed by (employs)	**VerificationActivity**
		formed by (forms)	**Component Link**
VerificationActivity	See Section 1.8	*accomplishes (accomplished by)*	**Verification Event**
		assigned to (responsible for)	**Organization**
		causes (caused by)	**Risk**
		based on (basis of)	**VerificationRequirement**
		decomposed by (decomposes)	**VerificationActivity**
		documented by (documents)	**Document**
		employs (employed by)	**TestConfiguration**
		executes (executed by)	**VerificationRequirement**
		generates (generated by)	**Concern**
		result of (results in)	**Risk**
VerificationEvent	Description Duration Duration Units End Date Labor Hours	*accomplished by (accomplishes)*	**VerificationActivity**
		augmented by (augments)	**ExternalFile**
		documented by (documents)	**Document**

Table 23 Verification Planning

Entity Class	Attributes	Relations	Target Classes
	Non-Recurring Cost Number Start Date Title Type	*includes* *(included in)*	**VerificationEvent**
VerificationRequirement	See Section 1.8	*basis of (based on)*	**VerificationActivity**
		specifies (specified by)	**VerificationActivity**

1.23 Verification Planning

Test support and planning are captured in the GENESYS repository using the **VerificationActivity** and **TestItem** classes. These classes are analogous to the **Function** and **Item** classes. From a behavioral perspective, there is no difference among these classes. There are some attribute and relational differences, however. Overall and individual test planning in GENESYS begins with defining major test threads for the system and culminates in an integrated behavior view of the test activities for the system, subsystem, etc. For the program or project, a top-level **VerificationActivity** should be defined and associated with a **VerificationEvent** using the *accomplishes (accomplished by)* relation pair. The **VerificationActivity** attribute behaviorType should be set to “Integrated (Root)” in order to identify that this top-level **VerificationActivity** represents the totality of testing needed to satisfy the test objectives of the program/system represented by the associated **VerificationEvent**. The root **VerificationActivity** may be decomposed (hierarchically) into all of the activities needed to satisfactorily define the test plan for the program/system. A **TestItem** is an *input to* or an *output from* a **VerificationActivity**. **TestItems** are the control indicators or measurables associated with a **VerificationActivity**, i.e., test data. A **VerificationActivity** is *established by* a **Requirement**.

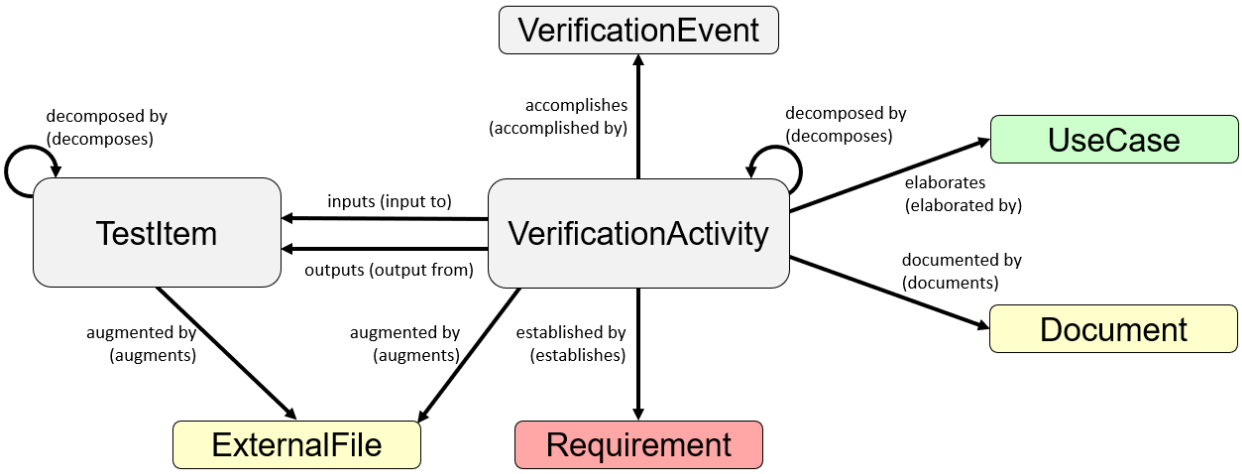


Figure 25 Test Planning

Table 24 Test Planning

Entity Class	Attributes	Relations	Target Classes
Document	See Section 1.2	*documents (documented by)*	**VerificationActivity**
ExternalFile	See Section 1.2	*augments (augmented by)*	**VerificationActivity** **TestItem**
VerificationActivity	See Section 1.8 and Table 23 Verification Planning	*accomplishes (accomplished by)*	**VerificationEvent**
		augmented by (augments)	**ExternalFile**
		decomposed by (decomposes)	**VerificationActivity**
		elaborates (elaborates by)	**UseCase**
		established by (establishes)	**Requirement**
		inputs (input to)	**TestItem**
		outputs (output from)	**TestItem**
TestItem	See Section 1.8	*augmented by (augments)*	**ExternalFile**
		decomposed by (decomposes)	**TestItem**
		input to (inputs)	**VerificationActivity**
		output from (outputs)	**VerificationActivity**
Requirement	See Sections 1.2 and 1.5	*establishes (established by)*	**VerificationActivity**

VERIFICATION COMPLIANCE TRACKING

As **VerificationActivities** are executed, results can be captured and tracked in the Actual Result attribute. If a **VerificationActivity** fails and does not meet its documented Expected Result criteria, the stakeholders must decide what action to take. The failure can be tracked using a **Concern**. The concern can be used to discuss the corrective action plan, which may include redesign, reverification, or the processing of a deviation or waiver request. Deviations and waivers are typically processed through a Change Management process and can be tracked in GENESYS using a **ChangeRequestPackage** entity related to the **VerificationActivities**' associated **VerificationRequirement**.

CHANGE MANAGEMENT SUPPORT

The GENESYS repository provides Change Management Support through the use of the **ChangeRequestPackage** class. The **ChangeRequestPackage** class allows the capture of system design changes, verification compliance issues, and their impacts upon the model held in the GENESYS repository. A **ChangeRequestPackage** entity contains the basic characterization of the change proposal, which will be submitted to the system's change approval agent and may be *augmented b*y one or more **ExternalFile** entities. The need for a formal change request occurs whenever there is a change to the

source requirements or other changes affecting the system's baseline.[15] These system changes may arise from internal and external organizations. Capturing the source of the change proposal is through the *originated by (originates)* relationship pair associating the **Organization** entity with the proposed **ChangeRequestPackage** entity. Also, the **ChangeRequestPackage** entity is *assigned to* one or more organizational entities for review. One or more of the **Component**, **Function**, **Link**, **Organization**, **Requirement**, **Resource**, **UseCase**, or **VerificationRequirement** classes may *generate* the **ChangeRequestPackage** entity. As an outcome of the analysis and review by the various **Organizations,** the impacts upon the system design will be established and the benefits and deficiencies of the proposed change are uncovered and presented to the system's change approval agent.

Once the proposed changes are approved, the system's current baseline is updated and resaved to serve as the program or project's current system baseline. The resulting saved change file (captures the set of approved changes in the model) is used to propagate the changes to the current working layer of the design repository.[16] This allows the design team to proceed with the design, while minimizing the effects of the changes on the current layer's design.

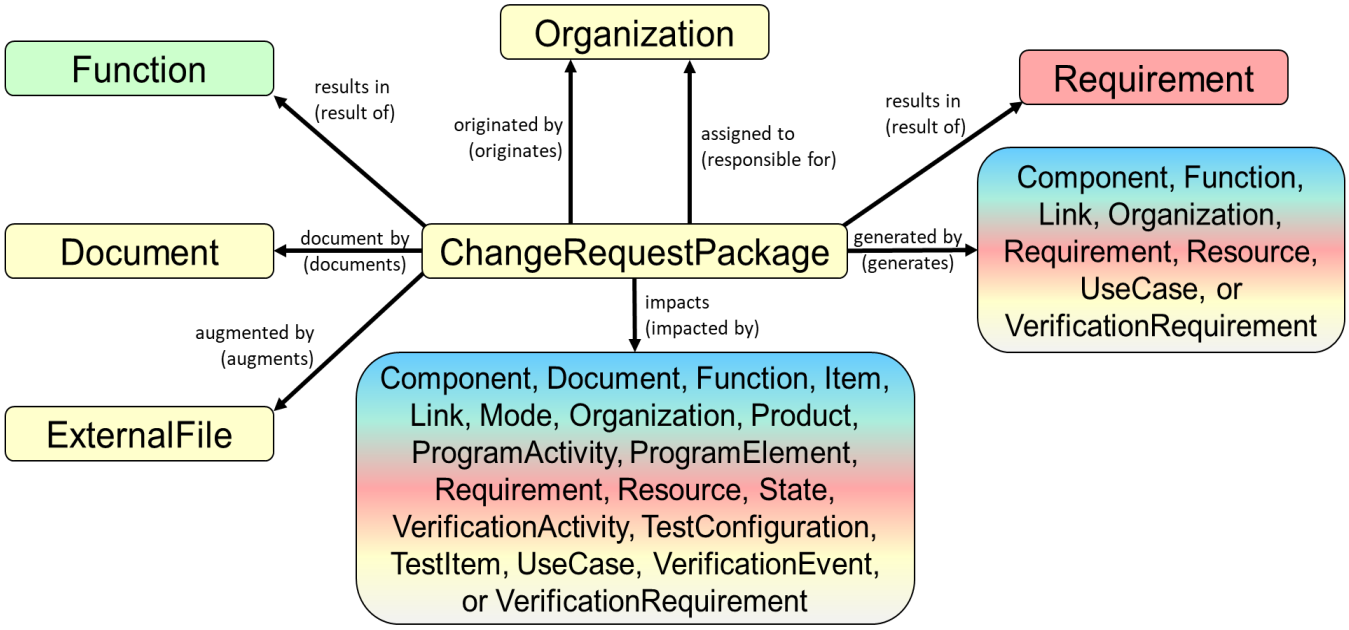


Figure 26 Change Management Support

Table 25 Change Management Support

Entity Class	Attributes	Relations	Target Classes
ChangeRequestPackage	See Section 1.7	*assigned to (responsible for)*	**Organization**

[15] A system design baseline is defined as a completed layer of the STRATA process, which has been placed under configuration control. The baseline of interest in this discussion is the baseline, which is the immediate predecessor to the current working layer.

[16] A properly executed system design process will only have changes affecting the current baseline (the current completed STRATA layer). If the change must go back to an even earlier baseline, it points to a flawed system design process and increases the likelihood of not having the design process converge to a realizable system, much less one within cost and schedule limits.

Table 25 Change Management Support

Entity Class	Attributes	Relations	Target Classes
		augmented by (augments)	**ExternalFile**
		documented by (documents)	**Document**
		generated by (generates)	**Component** **Document** **Function** **Link** **Organization** **Requirement** **Resource** **State** **UseCase** **VerificationRequirement**
		impacts (impacted by)	**Component** **Document** **Function** **Item** **Link** **Mode** **Organization** **Product** **ProgramActivity** **ProgramElement** **Requirement** **Resource** **State** **VerificationActivity** **TestConfiguration** **TestItem** **UseCase** **VerificationEvent** **VerificationRequirement**
		originated by (originates)	**Organization**
		results in (result of)	**Function** **Requirement**
Component	See Section 1.1	*generates (generated by)*	**ChangeRequestPackage**
		impacted by (impacts)	**ChangeRequestPackage**
Document	See Section 1.2	*generates (generated by)*	**ChangeRequestPackage**

Table 25 Change Management Support

Entity Class	Attributes	Relations	Target Classes
		impacted by (impacts)	**ChangeRequestPackage**
ExternalFile	See Section 1.2	*augments (augmented by)*	**ChangeRequestPackage**
Function	See Section 1.1	*generates (generated by)*	**ChangeRequestPackage**
		impacted by (impacts)	**ChangeRequestPackage**
		result of (results in)	**ChangeRequestPackage**
Item	See Section 1.7	*impacted by (impacts)*	**ChangeRequestPackage**
Link	See Section 1.3	*generates (generated by)*	**ChangeRequestPackage**
		impacted by (impacts)	**ChangeRequestPackage**
Organization	See Section 1.6	*generates (generated by)*	**ChangeRequestPackage**
		impacted by (impacts)	**ChangeRequestPackage**
		originates (originated by)	**ChangeRequestPackage**
		responsible for (assigned to)	**ChangeRequestPackage**
Requirement	See Sections 1.2 and 1.5	*generates (generated by)*	**ChangeRequestPackage**
		impacted by (impacts)	**ChangeRequestPackage**
		result of (results in)	**ChangeRequestPackage**
State	See Section 1.10	*generates (generated by)*	**ChangeRequestPackage**
		impacted by (impacts)	**ChangeRequestPackage**
VerificationActivity	See Section 1.8	*impacted by (impacts)*	**ChangeRequestPackage**
TestItem	See Section 1.8	*impacted by (impacts)*	**ChangeRequestPackage**
Requirement	See Sections 1.2 and 1.5	*generates (generated by)*	**ChangeRequestPackage**
		impacted by (impacts)	**ChangeRequestPackage**
		results of (results in)	**ChangeRequestPackage**

Table 25 Change Management Support

Entity Class	Attributes	Relations	Target Classes
VerificationRequirement	See Section 1.8	*generates (generated by)*	**ChangeRequestPackage**
		impacted by (impacts)	**ChangeRequestPackage**

2270 Kraft Drive, Suite 1600
Blacksburg, Virginia 24060
540.951.3322 | FAX: 540.951.8222
Customer Support: support@vitechcorp.com
www.vitechcorp.com