**Entity**

**General**

**Item** An article in a collection, enumeration, or series.

**Label** A descriptive name used to identify something.

**Meta** A prefix used on a concept to mean beyond or about its own concept, e.g. metadata is data about data.

**Section** A part of (a requirements) document.

**Term** A word or group of words having a particular meaning.

**Context**

**Actor** A human or machine that communicates with a system.

**App** A computer program, or group of programs designed for end users, normally with a graphical user interface. Short for application.

**Component** A composable part of a system. A reusable, interchangeable system unit or functionality.

**Domain** An application area. A product and its surrounding entities.

**Module** A collection of coherent functions and interfaces.

**Product** Something offered to a market.

**Release** A specific version of a system offered at a specific time to end users.

**Resource** A capability of, or support for development.

**Risk** Something negative that may happen.

**Service** Actions performed by systems and/or humans to provide results to stakeholders.

**Stakeholder** Someone with a stake in the system development or usage.

**System** A set of interacting software and/or hardware components.

**User** A human interacting with a system.

**Requirement**

**DataReq**

**Class** An extensible template for creating objects. A set of objects with certain attributes in common. A category.

**Data** Information stored in a system.

**Input** Data consumed by an entity.

**Member** An entity that is part of another entity, e.g. a field in a class.

**Output** Data produced by an entity, e.g. a function or a test.

**Relationship** A specific way that entities are connected.

**DesignReq**

**Design** A specific realization or high-level implementation description (of a system part).

**Screen** A design of (a part of) a user interface.

**MockUp** A prototype with limited functionality used to demonstrate a design idea.

**FunctionalReq**

**Function** A description of how input data is mapped to output data. A capability of a system to do something specific.

**Interface** A defined way to interact with a system.

**State** A mode or condition of something in the domain and/or in the system. A configuration of data.

**Event** Something that can happen in the domain and/or in the system.

**GeneralReq**

**Epic** A large user story or a collection of stories.

**Feature** A releasable characteristic of a product. A (high-level, coherent) bundle of requirements.

**Goal** An intention of a stakeholder or desired system property.

**Idea** A concept or thought (potentially interesting).

**Issue** Something needed to be fixed.

**Req** Something needed or wanted. An abstract term denoting any type of information relevant to the (specification of) intentions behind system development. Short for requirement.

**Ticket** (Development) work awaiting to be completed.

**WorkPackage** A collection of (development) work tasks.

**QualityReq**

**Breakpoint** A point of change. An important aspect of a (non-linear) relation between quality and benefit.

**Barrier** Something that makes it difficult to achieve a goal or a higher quality level.

**Quality** A distinguishing characteristic or degree of goodness.

**Target** A desired quality level or goal.

**ScenarioReq**

**Scenario** A (vivid) description of a (possible) future system usage.

**Task** A piece of work (that users do, maybe supported by a system).

**Test** A procedure to check if requirements are met.

**Story** A short description of what a user does or needs. Short for user story.

**UseCase** A list of steps defining interactions between actors and a system to achieve a goal.

**VariabilityReq**

**Variant** An object or system property that can be chosen from a set of options.

**RelationType**

**binds** Ties a value to an option. A configuration binds a variation point.

**deprecates** Makes outdated. An entity deprecates (supersedes) another entity.

**excludes** Prevents a combination. An entity excludes another entity.

**has** Expresses containment, substructure. An entity contains another entity.

**helps** Positive influence. A goal helps to fulfill another goal.

**hurts** Negative influence. A goal hinders another goal.

**impacts** Some influence. A new feature impacts an existing component.

**implements** Realisation of. A module implements a feature.

**interactsWith** Communication. A user interacts with an interface.

**is** Sub-typing, specialization, part of another, more general entity.

**precedes** Temporal ordering. A feature precedes (is implemented before) another feature.

**requires** Requested combination. An entity is required (or wished) by another entity.

**relatesTo** General relation. An entity is related to another entity.

**superOf** Super-typing, generalization, includes another, more specific entity.

**verifies** Gives evidence of correctness. A test verifies the implementation of a feature.
**Attribute**

**StringAttribute**

**Code** A collection of (textual) computer instructions in some programming language, e.g. Scala. Short for source code.

**Comment** A note that explains or discusses some entity.

**Deprecated** A description of why an entity should be avoided, often because it is superseded by another entity, as indicated by a 'deprecates' relation.

**Example** A note that illustrates some entity by a typical instance.

**Expectation** The required output of a test in order to be counted as passed.

**FileName** The name of a storage of serialized, persistent data.

**Gist** A short and simple description of an entity, e.g. a function or a test.

**Image** The name of a picture of an entity.

**Spec** A (detailed) definition of an entity. Short for specification

**Text** A sequence of words (in natural language).

**Title** A general or descriptive heading.

**Why** A description of intention. Rationale.

**IntAttribute**

**Benefit** A characterisation of a good or helpful result or effect (e.g. of a feature).

**Capacity** The largest amount that can be held or contained (e.g. by a resource).

**Cost** The expenditure of something, such as time or effort, necessary for the implementation of an entity.

**Damage** A characterisation of the negative consequences if some entity (e.g. a risk) occurs.

**Frequency** The rate of occurrence of some entity.

**Model Scripting**

**Model Construction**
A Model has a body within parentheses with a comma-separated sequence of zero or more Elms. A relation links an Entity with a submodel body including a sequence of zero or more Elms.

```scala
val m = Model(
  Title("example"),
  Feature("helloWorld") has Spec("Print hello msg."),
  Stakeholder("x") requires (Req("nice") has (
    Prio(10),
    Gist("gimme this")),
  Req("cool") has (Prio(5),
    Gist("better have it"))
)
```

**Model Operations**
Add element to a Model m:
```
val m = Model(
  Prio(1),
  Feature("1") has Benefit(4),
  Feature("2") has Benefit(2),
  Feature("3") has Benefit(1)),
Stakeholder("Y") has (Prio(2),
  Feature("1") has Benefit(2),
  Feature("2") has Benefit(1),
  Feature("3") has Benefit(1)),
Release("A") precedes Release("B"),
Resource("dev") has (Feature("1") has Cost(10),
  Feature("2") has Cost(10),
  Feature("3") has Cost(40),
  Release("A") has Capacity(100),
  Release("B") has Capacity(100)),
Resource("test") has (Feature("1") has Cost(40),
  Feature("2") has Cost(40),
  Feature("3") has Cost(70),
  Release("A") has Capacity(100),
  Release("B") has Capacity(100)),
Feature("1") has Constraint("x"),
Feature("2") has Constraint("y"),
Feature("3") has Constraint("z"))
```

Available exporters:
```
toGraphVizNested
toGraphVizFlat
toPathTable
tohtml
toText
tolatex
toQuperSpec
```

**TreeView Attribute**

**Constraints** A collection of propositions that restrict the possible values of a set of variables.

**Node**

```
Elem
Model
def toVector: Vector[Elem]
Relation
val entity: Entity
val link: RelationType
val tail: Model
Node
Entity
val id: String
val value: T
Attribute[T]

Attr(value: T)
val id: String
val value: T

Def

Elem
Model
def toVector: Vector[Elem]
Relation
val entity: Entity
val link: RelationType
val tail: Model
Node
Entity
val id: String
val value: T
Attribute[T]

Attr(value: T)
```