abstract	**abstrakt[ne]**	Something which cannot be directly instantiated; the opposite of *concrete*.
abstract class	**abstraktklass**	A *class* that cannot be directly instantiated. Contrast: *concrete class*.
abstraction	**abstraktsioon**	The creation of a *view* or *model* that suppresses unnecessary details to focus on a specific set of details of interest The essential characteristics of an entity that distinguish it from all other kinds of entities. An abstraction defines a boundary relative to the perspective of the viewer.
acceptance	**vastuvõtmine**	An action by which the customer accepts ownership of software products as a partial or complete performance of a contract.
action	**toiming**	The specification of an executable statement that forms an abstraction of a computational procedure. An action typically results in a change in the state of the system, and can be realized by sending a message to an object or modifying a link or a value of an attribute.
action sequence	**toimingujada**	An expression that resolves to a sequence of actions.
action state	**toimingu olek**	A state that represents the execution of an atomic action, typically the invocation of an *operation.*
activation	**aktiveerimine**	The execution of an *action.*
active class	**aktiivne klass**	A *class* representing a thread of control in the system.A class whose instances are active objects. See: active object.
activity	**tegevus**	A unit of work a *worker* may be asked to perform
active object	**aktiivne objekt**	An *object* that owns a *thread* and can initiate control activity. An instance of *active class.* See: *active class, thread.*
activity graph	**tegevusskeem**	A special case of a *state machine* that is used to model processes involving one or more classifiers. Contrast: *statechart diagram.* Synonym: *activity diagram.*
actor (instance)	**tegija(isend)**	Someone or something, outside the system or business that interacts with the system or business.
actor class	**tegijaklass**	Defines a set of actor instances, in which each actor instance plays the same role in relation to the system or business.A coherent set of roles that users of use cases play when interacting with these use cases. An actor has one role for each *use case* with which it communicates.
actor-generalization	**tegija üldistus**	An actor-generalization from an actor class (descendant) to another actor class (ancestor) indicates that the descendant inherits the role the ancestor can play in a use case.
actual parameter	**tegelik parameeter**	Synonym: *argument*.
aggregate [class]	**agregaat**	A class that represents the "whole" in an aggregation (whole-part) relationship. See: *aggregation.*
aggregation	**agregatsioon**	An association that models a whole-part relationship between an aggregate (the whole) and its parts.A special form of association that specifies a whole-part relationship between the aggregate (whole) and a component part. See: *composition.*
analysis	**analüüs**	The part of the software development process

		whose primary purpose is to formulate a model of the problem *domain*. Analysis focuses on what to do, design focuses on how to do it. See *design*.
analysis class	**analüüsiklass**	An abstraction of a *role* played by a design element in the system, typically within the context of a *use-case realization*. Analysis classes may provide an abstraction for several role, representing the common behavior of those roles. Analysis classes typically evolve into one or more design elements (e.g. design *class*es and/or *capsule*s, or design *subsystem*s).
analysis & design	**analüüs ja projekteerimine**	A *core workflow* in the Unified Process, whose purpose is to show how the system's *use case*s will be realized in implementation; (general) activities during which strategic and tactical decisions are made to meet the required functional and quality *requirements* of a system. For the result of analysis and design activities, see "*Design Model*."
analysis mechanism	**analüüsimehhanism**	An architectural mechanism used early in the *design* process, during the period of discovery when key *classes* and *subsystems* are being identified. Typically analysis mechanisms capture the key aspects of a solution in a way that is implementation independent. Analysis mechanisms are usually unrelated to the problem domain, but instead are "computer science" concepts. They provide specific behaviors to a domain-related *class* or *component*, or correspond to the implementation of cooperation between classes and/or components. They may be implemented as a *framework*. Examples include mechanisms to handle persistence, inter-process communication, error or fault handling, notification, and messaging, to name a few.
analysis time	**analüüsiaeg**	Refers to something that occurs during an analysis phase of the software development process. See: *design time, modeling time.*
architectural baseline	**arhitektuuri arendusalus**	The *baseline* at the end of the *Elaboration* phase, at which time the foundation structure and behavior of the system is stabilized.
architectural mechanism	**arhitektuurimehhanism**	An architectural mechanism represents a common solution to a frequently encountered problem. They may be patterns of structure, patterns of behavior, or both.
architectural pattern	**arhitektuurimall**	A description of an archetypal solution to a recurrent design problem that reflects well-proven design experience. Presented in the Software Architecture Document.
architectural view	**arhitektuurivaade**	A view of the system *architecture* from a given perspective; focuses primarily on structure, modularity, essential components, and the main control flows.
architecture	**arhitektuur**	The highest level concept of a system in its environment [*IEEE*]. The architecture of a software system (at a given point in time) is its organization or structure of significant *components* interacting through *interface*s, those components being composed of

		successively smaller components and interfaces.The organizational structure of a system. An architecture can be recursively decomposed into parts that interact through interfaces, relationships that connect parts, and constraints for assembling parts. Parts that interact through interfaces include *classes*, *components* and *subsystems*.
argument	**argument**	A binding for a parameter that resolves to a run-time instance. Synonym: *actual parameter*. Contrast: *parameter*.
artifact	**tehis**	(1) A piece of information that (1) is produced, modified, or used by a process, (2) defines an area of responsibility, and (3) is subject to version control. An artifact can be a *model*, a *model element*, or a *document*. A document can enclose other documents. A piece of information that is used or produced by a software development process. An artifact can be a model, a description, or software. Synonym: *product.*
artifact guidelines	**tehisejuhised**	A description of how to work with a particular *artifact*, including how to create and revise the artifact.
artifact set	**tehistik**	A set of related artifacts which presents one aspects of the system. Artifact sets cut across *core workflows*, as several artifacts are used in a number of core workflows (e.g. the Risk List, the Software Architecture Document, and the Iteration Plan).
association	**side**	A relationship that models a bi-directional semantic connection among instances.The semantic relationship between two or more classifiers that specifies connections among their instances.
association class	**sidemeklass**	A model element that has both *association* and *class* properties. An association class can be seen as an association that also has class properties, or as a class that also has association properties.
association end	**sidemeots**	The endpoint of an association, which connects the association to a *classifier*.
asynchronous action	**asünkroonne toiming**	A request where the sending object does not pause to wait for results. Contrast: *synchronous action*.
attribute	**atribuut**	An attribute defined by a *class* represents a named property of the class or its objects. An attribute has a *type* that defines the type of its instances.A feature within a classifier that describes a range of values that instances of the classifier may hold.
baseline	**arendusalus**	A reviewed and approved release of *artifacts* that constitutes an agreed basis for further evolution or development and that can be changed only through a formal procedure, such as *change management* and *configuration control.*
behavior	**käitumine**	The observable effects of an operation or event, including its results.
behavioral feature	**käitumisjoon**	A dynamic feature of a *model element*, such as an *operation* or *method.*
behavioral model aspect	**mudeli käitumisaspekt**	A *model aspect* that emphasizes the behavior of the *instances* in a system, including their

		methods, *collaborations*, and *state* histories.
binary association	**kahendasssotsiatsioon**	An association between two *classes*. A special case of an *n-ary association*.
binding	**sidumine**	The creation of a *model element* from a *template* by supplying arguments for the parameters of the template.
boolean	**Boole'i muutuja**	An enumeration whose values are true and false.
boolean expression	**Boole'i avaldis**	An expression that evaluates to a *boolean* value.
boundary class	**piiriklass**	A class used to model communication between the system's environments and its inner workings.
build	**redaktsioon**	An operational version of a system or part of a system that demonstrates a subset of the capabilities to be provided in the final product.
call	**kutse**	An *action state* that invokes an *operation* on a *classifier*.
capsule	**kapsel**	A specific *design pattern* which represents an encapsulated *thread* of control in the system. A capsule is a *stereotyped class* with a specific set of required and restricted *associations* and *properties*.
cardinality	**võimsus**	The number of elements in a set. Contrast: *multiplicity*.
change control board (CCB)	**muutmisnõukogu**	The role of the CCB is to provide a central control mechanism to ensure that every *change request* is properly considered, authorized and coordinated.
child	**tütar**	In a *generalization* relationship, the specialization of another element, the parent. See: *subclass*, *subtype*. Contrast: *parent*.
change management	**muutusehaldus**	The activity of controlling and tracking changes to *artifacts*. See also: *scope management*.
change request (CR)	**muutmistaotlus**	A general term for any request from a *stakeholder* to change an *artifact* or *process*. Documented in the Change Request is information on the origin and impact of the current problem, the proposed solution, and its cost. See also: *enhancement request*, *defect*.
checkpoints	**kontrolltingimused**	A set of conditions that well-formed *artifacts* of a particular type should exhibit. May also be stated in the form of questions which should be answered in the affirmative.
class	**klass**	A description of a set of objects that share the same *attributes*, *operations*, *methods*, *relationships*, and semantics. A class may use a set of interfaces to specify collections of operations it provides to its environment. See: *interface*.
class diagram	**klassiskeem**	A diagram that shows a collection of declarative (static) *model elements*, such as *classes*, *types*, and their contents and *relationships*.
client	**klient**	A *classifier* that requests a service from another classifier. Contrast: *supplier*.
classifier	**klassifikaator**	A mechanism that describes behavioral and structural features. Classifiers include *interfaces*, *classes*, *datatypes*, and *components*.
collaboration	**koostöö**	(1) Is a description of a collection of objects that interact to implement some behavior

		within a context. It describes a society of cooperating objects assembled to carry out some purpose. (2) It captures a more holistic view of behavior in the exchange of messages within a network of objects. (3) Collaborations show the unity of the three major structures underlying computation: data structure, control flow, and data flow. (4) A collaboration has a static and a dynamic part. The static part describes the roles that objects and links play in an instantiation of the collaboration. The dynamic part consists of one or more dynamic interactions that show message flow over time in the collaboration to perform computations. A collaboration may have a set of *messages* to describe its dynamic behavior. (5) A collaboration with messages is an interaction. The specification of how an *operation* or *classifier*, such as a *use case*, is realized by a set of classifiers and *associations* playing specific roles used in a specific way. The collaboration defines an interaction. See: *interaction.*
collaboration diagram	**koostööskeem**	(1) A collaboration diagram describes a pattern of interaction among objects; it shows the objects participating in the interaction by their links to each other and the *messages* they send to each other. (2) It is a *class diagram* that contains *classifier* roles and *association* roles rather than just classifiers and associations. (3) Collaboration diagrams and sequence diagrams both show interactions, but they emphasize different aspects. Sequence diagrams show time sequences clearly but do not show object relationships explicitly. Collaboration diagrams show object relationships clearly, but time sequences must be obtained from sequence numbers. A diagram that shows interactions organized around the structure of a *model*, using either classifiers and associations or instances and links. Unlike a sequence diagram, a collaboration diagram shows the relationships among the instances. Sequence diagrams and collaboration diagrams express similar information, but show it in different ways. See: *sequence diagram.*
comment	**kommentaar**	An annotation attached to an element or a collection of elements. A note has no semantics. Contrast: *constraint.*
communicates-association	**kasutusside**	An association between an *actor class* and a *use case class*, indicating that their instances interact. The direction of the association indicates the initiator of the communication (Unified Process convention).
communication association	**suhtlusside**	In a deployment diagram an association between nodes that implies a communication. See: *deployment diagram.*
compile time	**kompileerimisaegne**	Refers to something that occurs during the compilation of a software module. See: *modeling time, run time.*
component	**komponent**	A non-trivial, nearly independent, and replaceable part of a system that fulfills a clear

		function in the context of a well-defined *architecture*. A component conforms to and provides the physical realization of a set of *interfaces*. A physical, replaceable part of a system that packages implementation and conforms to and provides the realization of a set of interfaces. A component represents a physical piece of implementation of a system, including software code (source, binary or executable) or equivalents such as scripts or command files.
component diagram	**komponendiskeem**	A diagram that shows the organizations and dependencies among *components*.
component-based development (CBD)	**komponendipõhine arendus**	The creation and deployment of software-intensive systems assembled from *components* as well as the development and harvesting of such components.
component subsystem	**komponentalamsüsteem**	A *stereotyped* subsystem (i.e. «component») representing the logical abstraction in design of a *component*. It realizes one or more *interfaces*, and may be dependent on one or more interfaces. It may enclose zero or more *classes*, *packages* or other component subsystems, none of which are visible externally (only interfaces are visible). It may also enclose zero or more diagrams which illustrate internal behavior (e.g. *state*, *sequence* or *collaboration diagrams*).
composite [class]	**liitklass**	A *class* that is related to one or more classes by a composition relationship. See: *composition.*
composite aggregation	**liitside**	Synonym: *composition*.
composite state	**liitolek**	A state that consists of either concurrent (orthogonal) substates or sequential (disjoint) substates. See: *substate*.
composite substate	**liit-alamolek**	A *substate* that can be held simultaneously with other substates contained in the same composite state. Synonym: *region*. See: *composite state*.
composition	**kompositsioon**	A form of *aggregation* association with strong ownership and coincident lifetime as part of the whole. Parts with non-fixed *multiplicity* may be created after the *composite* itself, but once created they live and die with it (i.e., they share lifetimes). Such parts can also be explicitly removed before the death of the composite. Composition may be recursive. Synonym: *composite aggregation.*
concrete	**konkreet[ne]**	An entity in a configuration that satisfies an end-use function and can be uniquely identified at a given reference point. (*ISO*)
concrete class	**konkreetklass**	A *class* that can be directly instantiated. Contrast: *abstract class.*
concurrency	**konkurrentsus**	The occurrence of two or more activities during the same time interval. Concurrency can be achieved by interleaving or simultaneously executing two or more threads. See: *thread.*
concurrent substate	**konkurrentne alamolek**	A *substate* that can be held simultaneously with other substates contained in the same composite state. See: *composite substate.* Contrast: *disjoint substate.*

configuration	**konfiguratsioon**	(1) general: The arrangement of a system or network as defined by the nature, number, and chief characteristics of its functional units; applies to both hardware or software configuration. (2) The requirements, design, and implementation that define a particular version of a system or system component. See *configuration management*.
configuration item	**konfiguratsioonielement**	An entity in a configuration that satisfies an end-use function and can be uniquely identified at a given reference point. (*ISO*)
configuration management	**konfiguratsioonihaldus**	A supporting process whose purpose is to identify, define, and baseline items; control modifications and releases of these items; report and record status of the items and modification requests; ensure completeness, consistency and correctness of the items; and control storage, handling and delivery of the items. (*ISO*)
control class	**kontrollklass**	A *class* used to model behavior specific to one, or a several *use cases*.
constraint	**kitsendus**	A semantic condition or restriction. Certain constraints are predefined in the UML, others may be user defined. Constraints are one of three extensibility mechanisms in UML. See: *tagged value, stereotype.*
construction	**konstrueerimine**	The third phase of the Unified Process, in which the software is brought from an executable architectural baseline to the point at which it is ready to be transitioned to the user community.
container	**konteiner**	1. An *instance* that exists to contain other instances, and that provides operations to access or iterate over its contents. (for example, arrays, lists, sets). 2. A *component* that exists to contain other components.
containment hierarchy	**sisalduvushierarhia**	A namespace hierarchy consisting of *model elements*, and the containment relationships that exist between them. A containment hierarchy forms an acyclic graph.
context	**kontekst**	A view of a set of related *modeling elements* for a particular purpose, such as specifying an *operation.*
core workflow	**põhivoog**	One of nine core workflows in the Rational Unified Process: Business Modeling, Requirements, Analysis & Design, Implementation, Test, Deployment, Configuration & Change Management, Project Management, Environment. An abstract business use case of the Software-Engineering Business.
critical design review (CDR)	**lahenduse kriitiline läbivaatus**	In the waterfall life-cycle, the major review held when the detailed design is completed (see Guidelines: Project Plan).
customer	**tellija**	A person or organization, internal or external to the producing organization, who takes financial responsibility for the system. In a large system this may not be the end user. The customer is the ultimate recipient of the developed product and its artifacts. See also: *stakeholder*.
cycle	**tsükkel**	One complete pass through the four phases: *inception, elaboration, construction* and

		transition. The span of time between the beginning of the inception phase and the end of the transition phase.
datatype	**andmetüüp**	A descriptor of a set of values that lack identity and whose operations do not have side effects. Datatypes include primitive pre-defined types and user-definable types. Pre-defined types include numbers, string and time. User-definable types include enumerations.
deadlock	**tupik**	A condition in which two independent threads of control are blocked, each waiting for the other to take some action. Deadlock often arises from adding synchronization mechanisms to avoid *race conditions*.
defect	**defekt**	An anomaly, or flaw, in a delivered work product. Examples include such things as omissions and imperfections found during early lifecycle phases and symptoms of faults contained in software sufficiently mature for test or operation. A defect can be any kind of issue you want tracked and resolved. See also: *change request*.
defining model [MOF]	**defineeriv mudel**	The model on which a repository is based. Any number of repositories can have the same defining model.
delegation	**delegeerimine**	The ability of an object to issue a *message* to another object in response to a message. Delegation can be used as an alternative to inheritance. Contrast: *inheritance*.
deliverable	**saadus**	An output from a process that has a value, material or otherwise, to a *customer* or other *stakeholder*.
dependency	**sõltuvus**	A relationship between two *modeling elements*, in which a change to one modeling element (the independent element) will affect the other modeling element (the dependent element).
deployment	**evitus**	A *core process workflow* in the software-engineering process, whose purpose is to ensure a successful transition of the developed system to its users. Included are *artifacts* such as training materials and installation procedures.
deployment diagram	**evitusskeem**	A diagram that shows the configuration of run-time processing nodes and the *components*, *processes*, and *objects* that live on them. Components represent run-time manifestations of code units. See: *component diagram*.
deployment view	**evitusvaade**	An *architectural view* that describes one or several system configurations; the mapping of software *components* (tasks, modules) to the computing nodes in these configurations.
derived element	**tuletiselement**	A model element that can be computed from another element, but that is shown for clarity or that is included for design purposes even though it adds no semantic information.
design	**projekteerimine**	The part of the software development process whose primary purpose is to decide how the system will be implemented. During design, strategic and tactical decisions are made to meet the required functional and quality

		requirements of a system. See *analysis.*
design time	**projekteerimisaegne**	Refers to something that occurs during a design phase of the software development process. See: *modeling time*. Contrast: *analysis time.*
design mechanism	**projekteerimismehhanism**	An *architectural mechanism* used during the design process, during the period in which the details of the design are being worked-out. They are related to associated *analysis mechanisms*, of which they are additional refinements. A design mechanism assumes some details of the implementation environment, but it is not tied to a specific implementation (as is an *implementation mechanism*). For example, the analysis mechanism for inter-process communication may be refined by several design mechanisms for interprocess communication (IPC): shared memory, function-call-like IPC, semaphore-based IPC, and so on. Each design mechanism has certain strengths and weaknesses; the choice of a particular design mechanism is determined by the characteristics of the objects using the mechanism.
design model	**projekteerimismudel**	An *object model* describing the realization of *use cases*; serves as an abstraction of the *implementation model* and its source code.
design package	**projekteerimispakett**	A collection of *classes*, *relationships*, *use-case realizations*, *diagrams*, and other *packages*; it is used to structure the *design model* by dividing it into smaller parts.
design pattern	**projekteerimismall**	A specific solution to a particular problem in software design. Design patterns capture solutions that have developed and evolved over time, expressed in a succinct and easily applied form. Generally design patterns express solutions at a lower level of granularity than mechanisms, and may very well be used to design a *design mechanism*.
design subsystem	**projekteerimis-alamsüsteem**	A *design package* that contains a collection of design packages and *classes*, and used to structure the *design model* by dividing it into smaller parts. See: *design package.*
developer	**väljatöötaja**	A person responsible for developing the required functionality in accordance with project-adopted standards and procedures. This can include performing activities in any of the requirements, analysis & design, implementation, and test workflows.
development case	**väljatöötusjuhtum**	The software-engineering process used by the performing organization. It is developed as a configuration, or customization, of the Unified Process product, and adapted to the project's needs.
development process	**väljatööteprotsess**	A set of partially ordered steps performed for a given purpose during software development, such as constructing models or implementing models.
device	**vahend**	A type of *node* which provides supporting capabilities to a *processor*. Although it may be capable of running embedded programs (device drivers), it cannot execute general-purpose applications, but instead exists only to

		serve a processor running general-purpose applications.
diagram	**skeem**	A graphical depiction of all or part of a *model*. A graphical presentation of a collection of *model elements*, most often rendered as a connected graph of arcs (relationships) and vertices (other model elements). UML supports the following diagrams: *class diagram*, *object diagram*, *use-case diagram*, *sequence diagram*, *collaboration diagram*, *statechart diagram*, *activity diagram*, *component diagram*, and *deployment diagram*.
disjoint substate	**ird-alamolek**	A *substate* that cannot be held simultaneously with other substates contained in the same composite state. See: *composite state.* Contrast: *concurrent substate.*
distribution unit	**jaotusüksus**	A set of *objects* or *components* that are allocated to a process or a processor as a group. A distribution unit can be represented by a run-time *composite* or an *aggregate*.
document	**dokument**	A document is a collection of information that is intended to be represented on paper, or in a medium using a paper metaphor. The paper metaphor includes the concept of pages, and it has either an implicit or explicit sequence of contents. The information is in text or two-dimensional pictures. Examples of paper metaphors are word processor documents, spreadsheets, schedules, Gantt charts, web-pages, or overhead slide presentations.
document description	**dokumendi kirjeldus**	Describes the contents of a particular document.
document template	**dokumendimall**	A concrete tool template, such as a Adobe® FrameMaker™ or Microsoft® Word™ template.
domain	**valdkond**	An area of knowledge or activity characterized by a family of related systems. An area of knowledge or activity characterized by a set of concepts and terminology understood by practitioners in that area.
domain model	**valdkonnamudel**	A domain model captures the most important types of objects in the context of the *domain*. The domain objects represent the entities that exist or events that transpire in the environment in which the system works. The domain model is a subset of the business object model.
dynamic classification	**dünaamiline liigitus**	A semantic variation of *generalization* in which an *object* may change *type* or *role*. Contrast: *static classification.*
elaboration	**detailimine**	The second *phase* of the process where the product *vision* and its *architecture* are defined.
element	**element**	An atomic constituent of a *model*.
enclosed document	**manusdokument**	A *document* can be enclosed by another document to collect a set of documents into a whole; the enclosing document as well as the individual enclosures are regarded as separate *artifacts.*
enhancement request	**täiendustaotlus**	A type of *stakeholder request* that specifies a new *feature* or functionality of the system. See also: *change request*
entity class	**olemiklass**	A *class* used to model information that has

		been stored by the system, and the associated behavior. A generic class, reused in many *use cases*, often with persistent characteristics. An entity class defines a set of entity objects, which participate in several use cases and typically survive those use cases.
entry action	**sisenemistoiming**	An action executed upon entering a *state* in a *state machine* regardless of the transition taken to reach that state.
enumeration	**väärtustik**	A list of named values used as the range of a particular *attribute* type. For example, RGBColor = {red, green, blue}. Boolean is a predefined enumeration with values from the set {false, true}.
event	**sündmus**	The specification of a significant occurrence that has a location in time and space. In the context of *state diagrams*, an event is an occurrence that can trigger a *transition.*
environment	**keskkond**	A *core supporting workflow* in the software-engineering process, whose purpose is to define and manage the environment in which the system is being developed. Includes process descriptions, *configuration management*, and development tools.
evolution	**areng**	The life of the software after its initial development cycle; any subsequent cycle, during which the product evolves.
evolutionary	**arenguline**	An iterative development strategy that acknowledges that user needs are not fully understood and therefore requirements are refined in each succeeding iteration (*elaboration phase*).
exit action	**väljumistoiming**	An action executed upon exiting a *state* in a *state machine* regardless of the transition taken to exit that state.
export	**eksport**	In the context of *packages*, to make an element visible outside its enclosing namespace. See: *visibility*. Contrast: *export* [OMA], *import.*
expression	**avaldis**	A string that evaluates to a value of a particular type. For example, the expression "(7 + 5 * 3)" evaluates to a value of type number.
extend	**laiendus**	A relationship from an extension use case to a base use case, specifying how the behavior defined for the extension use case can be inserted into the behavior defined for the base use case.
extend-relationship	**laiendusseos**	An extend-relationship from a use-case class A to a use-case class B indicates that an instance of B may include (subject to specific conditions specified in the extension) the behavior specified by A. Behavior specified by several extenders of a single target use case can occur within a single use-case instance.
facade	**fassaad**	A special package, stereotyped «facade», within a subsystem that organizes and exports all information needed by the clients of the subsystem. Included in this package are *interfaces* (where the interfaces are unique to the subsystem), realization relationships to interfaces outside the subsystem, and any documentation needed by clients of the

		subsystem to use the subsystem.	
fault	**viga**	An accidental condition that causes a component in the implementation model to fail to perform its required behavior. A fault is the root cause of one or more *defects*.	
feature	**erisus**	An externally observable service provided by the system which directly fulfills a *stakeholder need*. A property, like operation or attribute, which is encapsulated within a classifier, such as an interface, a class, or a datatype.	
final state	**lõppolek**	A special kind of state signifying that the enclosing *composite state* or the entire *state machine* is completed.	
fire	**vallandama**	To execute a state transition. See: *transition*.	
focus of control	**juhtimiskese**	A symbol on a *sequence diagram* that shows the period of time during which an object is performing an action, either directly or through a subordinate procedure.	
formal parameter	**formaalparameeter**	Synonym: *parameter*.	
framework	**raamstruktuur**	A micro-*architecture* that provides an extensible *template* for applications within a specific *domain*.	
FURPS	**FURPS**	An acronym representing categories for assessing product quality: Functionality, Usability, Reliability, Performance, Supportability. **(funktsionaalsus, kasutuskõlblikkus, töökindlus, suutvus, toetatavus)**	
generalizable element	**üldistuv element**	A model element that may participate in a generalization relationship. See: *generalization*.	
generalization	**üldistus**	A taxonomic relationship between a more general element and a more specific element. The more specific element is fully consistent with the more general element and contains additional information. An instance of the more specific element may be used where the more general element is allowed. See: *inheritance*.	
generation	**põlv**	Final release at the end of a cycle.	JOKA: **valmistoode**. “Põlv” ei haaku ju kuidagi sellega, mida see kolmandas veerus olev inglise keelne tekst räägib. Ega see inglisekeelne termin kah targem ei ole – generation on minuarust rohkem nigu **põlvkond** või nii. Ma ei imesta, kui see mingi viga seal RUP’is on.
green-field development	**täisväljatöötus**	Development "starting from scratch", as opposed to "evolution of an existing system" or "reengineering of a legacy piece". (Originated from the transformation that takes place when building a new factory on an undeveloped site - with grass on it.)	
guard condition	**siirdetingimus**	A condition that must be satisfied in order to enable an associated *transition* to *fire*.	

IEEE	**IEEE**	The Institute of Electrical and Electronics Engineers, Inc.
ISO	**ISO**	The International Organization for Standardization.
implementation	**teostus**	A core process workflow in the software-engineering process, whose purpose is to implement and unit test the classes. A definition of how something is constructed or computed. For example, a class is an implementation of a type, a method is an implementation of an operation.
implementation inheritance	**teostuspärilus**	The inheritance of the implementation of a more specific element. Includes inheritance of the interface. Contrast: *interface inheritance*.
implementation mechanism	**teostusmehhanism**	An *architectural mechanism* used during the implementation process. They are refinements of *design mechanisms*, and specify the exact implementation of the mechanism. For example, one particular implementation of the inter-process communication analysis mechanism is a shared memory design mechanism utilizing a particular operating system's shared memory function calls. Concurrency conflicts (inappropriate simultaneous access to shared memory) may be prevented using semaphores, or using a latching mechanism, which in turn rest upon other implementation mechanisms.
implementation model	**teostusmudel**	The implementation model is a collection of *components*, and the *implementation subsystems* that contain them.
implementation subsystem	**teostusalamsüsteem**	A collection of *components* and other implementation subsystems, and is used to structure the *implementation model* by dividing it into smaller parts.
implementation view	**teostusvaade**	An *architectural view* that describes the organization of the static software elements (code, data, and other accompanying artifacts) on the development environment, in terms of both *packaging*, *layering*, and *configuration management* (ownership, release strategy, and so on). In the Unified Process it is a view on the implementation model.
import	**import**	In the context of *packages*, a dependency that shows the packages whose *classes* may be referenced within a given package (including packages recursively embedded within it). Contrast: *export*.
import-dependency	**impordisõltuvus**	A *stereotyped* dependency in the design whose source is a *design package*, and whose target is a different design package. The import dependency causes the public contents of the target package to be referenceable in the source package.
inception	**algatamine**	The first *phase* of the Unified Process, in which the seed idea, request for proposal, for the previous generation is brought to the point of being (at least internally) funded to enter the *elaboration* phase.
include	**sisalduvus**	A relationship from a base use case to an inclusion use case, specifying how the behavior defined for the inclusion use case can be inserted into the behavior defined for the

		base use case.
include-relationship	**sisalduvusseos**	An include-relationship is a relationship from a base use case to an inclusion use case, specifying how the behavior defined for the inclusion use case is explicitly inserted into the behavior defined for the base use case.
increment	**inkrement**	The difference (*delta*) between two releases at the end of subsequent *iterations.*
incremental	**inkrementaalne**	Qualifies an iterative development strategy in which the system is built by adding more and more functionality at each *iteration.*
inheritance	**pärilus**	The mechanism that makes generalization possible; a mechanism for creating full class descriptions out of individual class segments. The mechanism by which more specific elements incorporate structure and behavior of more general elements related by behavior. See *generalization*.
input	**sisendtehis**	An *artifact* used by a process. See static artifact.
inspection	**inspekteerimine**	A formal evaluation technique in which some *artifact* (model, document, software) is examined by a person or group other than the originator, to detect faults, violations of development standards, and other problems.
instance	**isend**	An individual entity satisfying the description of a *class* or *type*. An entity to which a set of operations can be applied and which has a state that stores the effects of the operations. See: *object.*
integration	**integratsioon**	The software development activity in which separate software components are combined into an executable whole.
integration build plan	**integratsiooni järguplaan**	Defines the order in which components are to be implemented and integrated in a specific iteration. Enclosed in the Iteration Plan.
interaction	**interaktsioon**	A specification of how stimuli are sent between *instances* to perform a specific task. The interaction is defined in the context of a collaboration. See *collaboration.*
interaction diagram	**interaktsiooniskeem**	A generic term that applies to several types of diagrams that emphasize object interactions. These include: *collaboration diagrams*, *sequence diagrams*, and *activity diagrams.*
interface	**liides**	A collection of *operations* that are used to specify a service of a *class* or a *component.* A named set of operations that characterize the behavior of an element.
interface inheritance	**liidesepärilus**	The inheritance of the interface of a more specific element. Does not include inheritance of the implementation. Contrast: *implementation inheritance*.
internal transition	**sisesiire**	A *transition* signifying a response to an event without changing the *state* of an object.
iteration	**iteratsioon**	A distinct sequence of activities with a base-lined plan and valuation criteria resulting in a *release* (internal or external).
key mechanism	**võtmemehhanism**	A description of how an *architectural patterns* is realized in terms of patterns of interaction between elements in the system. Presented in the Software Architecture Document
layer	**kiht**	A specific way of grouping *packages* in a model at the same level of abstraction. The

		organization of classifiers or packages at the same level of abstraction. A layer represents a horizontal slice through an architecture, whereas a partition represents a vertical slice. Contrast: *partition.*	
link	**link**	A semantic connection among a tuple of objects. An instance of an association. See: *association.*	
link end	**lingiots**	An instance of an association end. See: *association end*.	
logical view	**loogikavaade**	An *architectural view* that describes the main classes in the design of the system: major business-related classes, and the classes that define key behavioral and structural mechanisms (persistency, communications, fault-tolerance, user-interface). In the Unified Process, the logical view is a *view* of the *design model*.	
management	**haldus**	A *core supporting workflow* in the software-engineering process, whose purpose is to plan and manage the development project.	
message	**teade**	A specification of the conveyance of information from one instance to another, with the expectation that activity will ensue. A message may specify the raising of a signal or the call of an operation.	
metaclass	**metaklass**	A class whose instances are classes. Metaclasses are typically used to construct *metamodels.*	
meta-metamodel	**meta-metamudel**	A model that defines the language for expressing a *metamodel*. The relationship between a meta-metamodel and a metamodel is analogous to the relationship between a metamodel and a *model*.	
metamodel	**metamudel**	A model that defines the language for expressing a *model*.	
metaobject	**metaobjekt**	A generic term for all metaentities in a metamodeling language. For example, metatypes, metaclasses, metaattributes, and metaassociations.	
method	**meetod**	(1) A regular and systematic way of accomplishing something; the detailed, logically ordered plans or procedures followed to accomplish a task or attain a goal. (2) UML 1.1: The implementation of an operation, the algorithm or procedure that effects the results of an operation. The implementation of an operation. It specifies the algorithm or procedure associated with an operation.	
milestone	**tähtpunkt**	The point at which an iteration formally ends; corresponds to a *release* point.	JOKA: **verstapost**?
model [MOF]	**mudel**	A semantically closed abstraction of a system. In the Unified Process, a complete description of a system from a particular perspective ('complete' meaning you don't need any additional information to understand the system from that perspective); a set of model elements. Two models cannot overlap. A semantically closed abstraction of a subject system. See: *system.* Usage note: In the context of the MOF specification, which describes a *meta-metamodel*, for brevity the meta-metamodel is frequently referred to as	

		simply the model.
model aspect	**mudeli aspekt**	A dimension of modeling that emphasizes particular qualities of the *metamodel*. For example, the structural model aspect emphasizes the structural qualities of the metamodel.
model elaboration	**mudeli detailimine**	The process of generating a *repository* type from a published model. Includes the generation of interfaces and implementations which allows repositories to be instantiated and populated based on, and in compliance with, the model elaborated.
model element [MOF]	**mudeli element**	An element that is an abstraction drawn from the system being modeled. Contrast: *view element.* In the MOF specification model elements are considered to be *metaobjects*.
modeling conventions	**modelleerimisreeglid**	How concepts will be represented, restrictions on the modeling language that the project team management has decided upon (i.e. dictums such as "Do not use inheritance between subsystems."; "Do not use extend or include associations in the Use Case Model."; "Do not use the friend construct in C++."). Presented in the Software Architecture Document.
modeling time	**modelleerimisaegne**	Refers to something that occurs during a modeling phase of the software development process. It includes analysis time and design time. Usage note: When discussing object systems, it is often important to distinguish between modeling-time and run-time concerns. See: *analysis time, design time.* Contrast: *run time*.
module	**moodul**	A software unit of storage and manipulation. Modules include source code modules, binary code modules, and executable code modules. See: *component.*
multiple classification	**mitmene liigitus**	A semantic variation of *generalization* in which an object may belong directly to more than one class. See: *dynamic classification*.
multiple inheritance	**mitmene pärilus**	A semantic variation of *generalization* in which a type may have more than one *supertype*. Contrast: *single inheritance.*
multiplicity	**võimsustik**	A specification of the range of allowable cardinalities that a set may assume. Multiplicity specifications may be given for roles within associations, parts within composites, repetitions, and other purposes. Essentially a multiplicity is a (possibly infinite) subset of the non-negative integers. Contrast: *cardinality*.
multi-valued [MOF]	**mitmeväärtuseline**	A model element with *multiplicity* defined whose Multiplicity Type:: upper attribute is set to a number greater than one. The term multi-valued does not pertain to the number of values held by an attribute, parameter, etc. at any point in time. Contrast: *single-valued.*
n-ary association	**n-ndassotsiatsioon**	An association among three or more classes. Each instance of the association is an n-tuple of values from the respective classes. Contrast: *binary association*.
name	**nimi**	A string used to identify a *model element.*
namespace	**nimeruum**	A part of the model in which the names may

		be defined and used. Within a namespace, each name has a unique meaning. See: *name*.
node	**sõlm**	A node is classifier that represents a run-time computational resource, which generally has at least a memory and often processing capability. Run-time objects and components may reside on nodes.
object	**objekt**	An entity with a well-defined boundary and identity that encapsulates *state* and *behavior*. State is represented by *attributes* and *relationships*, behavior is represented by *operations*, *methods*, and *state machines.* An object is an instance of a class. See: *class*, *instance*.
object diagram	**objektiskeem**	A diagram that encompasses *objects* and their relationships at a point in time. An object diagram may be considered a special case of a class diagram or a collaboration diagram. See: *class diagram, collaboration diagram.*
object flow state	**objekti voo-olek**	A *state* in an *activity graph* that represents the passing of an object from the output of actions in one state to the input of actions in another state.
object lifeline	**objekti eluiga**	A line in a sequence diagram that represents the existence of an object over a period of time. See: *sequence diagram.*
object model	**objektmudel**	An abstraction of a system's implementation.
operation	**operatsioon**	A service that can be requested from an object to effect behavior. An operation has a *signature*, which may restrict the actual parameters that are possible.
operating system process	**operatsioonisüsteemi protsess**	An unique address space and execution environment in which instances of classes and subsystems reside and run. The execution environment may be divided into one or more threads of control. See also *process* and *thread.*
originator	**lähetaja**	An originator is anyone who submits a *change request* (CR). The standard change request mechanism requires the originator to provide information on the current problem, and a proposed solution in accordance with the change request form.
output	**väljundtehis**	Any artifact that is the result of a process step. See *deliverable.*
package	**pakett**	A general purpose mechanism for organizing elements into groups. Packages may be nested within other packages.
parameter	**parameeter**	The specification of a variable that can be changed, passed, or returned. A parameter may include a name, type, and direction. Parameters are used for operations, messages, and events. Synonyms: *formal parameter*. Contrast: *argument.*
parameterized element	**parameetritega element**	The descriptor for a class with one or more unbound parameters. Synonym: *template.*
parent	**ema**	In a *generalization* relationship, the generalization of another element, the child. See: *subclass*, *subtype*. Contrast: *child.*
participates	**osaleb**	The connection of a model element to a relationship or to a reified relationship. For example, a class participates in an association, an actor participates in a use case.

partition	**sektsioon**	1. *activity graphs*: A portion of an activity graphs that organizes the responsibilities for actions. See: *swimlane*. 2. *architecture*: A subset of classifiers or packages at the same level of abstraction. A partition represents a vertical slice through an architecture, whereas a layer represents a horizontal slice. Contrast: *layer*.
pattern	**mall**	A scheme for describing design fragments or collections of class templates so that they can be configured and reused.
persistent object	**püsiv objekt**	An object that exists after the process or thread that created it has ceased to exist.
phase	**faas**	The time between two major project milestones, during which a well-defined set of objectives is met, artifacts are completed, and decisions are made to move or not move into the next phase.
post-condition	**järeltingimus**	A textual description defining a constraint on the system when a use case has terminated. A constraint that must be true at the completion of an operation.
pre-condition	**eeltingimus**	A textual description defining a constraint on the system when a use case may start. A constraint that must be true when an operation is invoked.
preliminary design review (PDR)	**alglahenduse läbivaatus**	In the waterfall life-cycle, the major review held when the architectural design is completed (see Guidelines: Project Plan).
primitive type	**primitiivtüüp**	A pre-defined basic datatype without any substructure, such as an integer or a string.
process	**protsess**	(1) A thread of control that can logically execute concurrently with other processes, specifically an operating system process. See also: *thread.* (2) A set of partially ordered steps intended to reach a goal; in software engineering the goal is to build a software product or to enhance an existing one; in process engineering, the goal is to develop or enhance a process model; corresponds to a business use case in business engineering. 1. A heavyweight unit of concurrency and execution in an operating system. Contrast: *thread*, which includes heavyweight and lightweight processes. If necessary, an implementation distinction can be made using stereotypes. 2. A software development process—the steps and guidelines by which to develop a system. 3. To execute an algorithm or otherwise handle something dynamically.
process view	**protsessivaade**	An *architectural view* that describes the concurrent aspect of the system: tasks (processes) and their interactions.
processor	**protsessor**	A type of node which possesses the capability to run one or more processes. Generally this requires a computational capability, memory, input-output devices, etc. See also: *node*, *process*, and *device*.
product	**toode**	Software that is the result of development, and some of the associated *artifacts* (documentation, release medium, training).
product champion	**tootejuht**	A high-ranking individual who owns the *vision* of the product and acts as an advocate

		between development and the *customer*.
product requirements document (PRD)	**tootenõuete dokument (PRD)**	A high level description of the product (system), its intended use, and the set of *features* it provides.
project manager	**projektijuht**	The worker with overall responsibility for the project. The Project Manager needs to ensure tasks are scheduled, allocated and completed in accordance with project schedules, budgets and quality requirements.
Project Review Authority (PRA)	**projekti läbivaatusorgan**	The organizational entity to which the Project Manager reports. The PRA is responsible for ensuring that a software project complies with policies, practices and standards (see Concepts: Organizational Context for the Rational Unified Process).
projection	**projektsioon**	A mapping from a set to a subset of it.
property	**omadus**	A named value denoting a characteristic of an element. A property has semantic impact. Certain properties are predefined in the UML; others may be user defined. See: *tagged value.*
protocol	**protokoll**	A specification of a compatible set of messages used to communicate between *capsule*s. The protocol defines a set of incoming and outgoing messages types (e.g. operations, signals), and optionally a set of sequence diagrams which define the required ordering of messages and a state machine which specifies the abstract behavior that the participants in a protocol must provide.
prototype	**prototüüp**	A release that is not necessarily subject to *change management* and *configuration control*.
pseudo-state	**pseudoolek**	A vertex in a state machine that has the form of a state, but doesn't behave as a state. Pseudo-states include initial and history vertices.
published model [MOF]	**avaldatud mudel**	A model which has been frozen, and becomes available for instantiating repositories and for the support in defining other models. A frozen model's model elements cannot be changed.
qualifier	**kvalifikaator**	An association attribute or tuple of attributes whose values partition the set of objects related to an object across an association.
quality assurance (QA)	**kvaliteedi tagamine**	The function of Quality Assurance is the responsibility of (reports to) the Project Manager and is responsible for ensuring that project standards are correctly and verifiably followed by all project staff.
race condition	**trügimine**	A condition which occurs when two or more independent tasks may simultaneously access and modify the same state information. This condition can lead to inconsistent behavior of the system and is a fundamental issue in concurrent system design.
rank	**kaalukus**	An attribute of a *use case* or *scenario* that describes its impact on the *architecture*, or its importance for a *release.*
rationale	**põhjendus**	A statement, or explanation of the reasons for a choice
receive [a message]	**vastu võtma**	The handling of a stimulus passed from a sender instance. See: *sender, receiver.*
receiver [object]	**vastuvõtja**	The object handling a stimulus passed from a sender object. Contrast: *sender.*

reception	**vastuvõtt**	A declaration that a classifier is prepared to react to the receipt of a signal.
reference	**viide**	1. A denotation of a model element. 2. A named slot within a classifier that facilitates navigation to other classifiers. Synonym: *pointer*.
refinement	**täpsustus**	A relationship that represents a fuller specification of something that has already been specified at a certain level of detail. For example, a design class is a refinement of an analysis class.
relationship	**seos**	A semantic connection among model elements. Examples of relationships include *associations* and *generalizations.*
release	**redaktsioon**	A subset of the end-product that is the object of evaluation at a major milestone. See: *prototype*, *baseline*.
release manager	**redaktsioonijuht**	A release manager is responsible for ensuring that all software assets are controlled and configurable into internal and external *releases* as required.
report	**aruanne**	An automatically generated description, describing one or several *artifacts.* A report is not an artifact in itself. A report is in most cases a transitory product of the development process, and a vehicle to communicate certain aspects of the evolving system; it is a snapshot description of artifacts that are not documents themselves.
repository	**hoidla**	A storage place for object models, interfaces, and implementations.
requirement	**nõue**	A requirement describes a condition or capability to which a system must conform; either derived directly from user needs, or stated in a contract, standard, specification, or other formally imposed document. See: Concept: Requirements A desired feature, property, or behavior of a system.
requirement attribute	**nõude atribuut**	Information associated with a particular requirement providing a link between the requirement and other project elements - e.g., priorities, schedules, status, design elements, resources, costs, hazards.
requirements	**nõuded**	A *core process workflow* in the software-engineering process, whose purpose is to define what the system should do. The most significant activities are to develop a vision, a use-case model and software requirements specifications.
requirements management	**nõudehaldus**	A systematic approach to eliciting, organizing and documenting the requirements of the system, and establishing and maintaining agreement between the customer and the project team on the changing requirements of the system. See: Concept: Requirements Management.
requirements tracing	**nõudejälitus**	The linking of a *requirement* to other requirements and to other associated project elements.
requirement type	**nõude tüüp**	A categorization of requirements (e.g., stakeholder need, feature, use case, supplementary requirement, test requirement, documentation requirement, hardware

		requirement, software requirement, etc.) based on common characteristics and attributes. See: Concept: Requirement Types
responsibility	**kohustus**	A contract or obligation of a classifier.
result	**tulem**	Synonym of output. See also *deliverable.*
review	**läbivaatus**	A review is a group activity carried out to discover potential defects and to assess the quality of a set of *artifacts.*
reuse	**taaskasutus**	Further use or repeated use of an *artifact* The use of a pre-existing artifact.
risk	**risk**	An ongoing or upcoming concern that has a significant probability of adversely affecting the success of major milestones.
role	**roll**	The behavior of a design element participating in a particular context (e.g. use-case realization). See also: *analysis class.* The named specific behavior of an entity participating in a particular context. A role may be static (e.g., an association end) or dynamic (e.g., a collaboration role).
run time	**käitusfaas**	The period of time during which a computer program executes. Contrast: *modeling time*.
scenario	**stsenaarium**	A described *use-case instance*, a subset of a *use case*. A specific sequence of actions that illustrates behaviors. A scenario may be used to illustrate an interaction or the execution of a use case instance. See: *interaction.*
scope management	**ulatuse haldus**	The process of prioritizing and determining the set of requirements that can be implemented in a particular release cycle, based on the resources and time available. This process continues throughout the lifecycle of the project as changes occur. See also: *change management.*
schema [MOF]	**komplekt**	In the context of the MOF, a schema is analogous to a *package* which is a container of *model elements.* Schema corresponds to an MOF package. Contrast: *metamodel*, package corresponds to an MOF package.
semantic variation point	**semantilise vabaduse punkt**	A point of variation in the semantics of a *metamodel*. It provides an intentional degree of freedom for the interpretation of the metamodel semantics.
send [a message]	**saatma**	The passing of a stimulus from a sender instance to a receiver instance. See: *sender, receiver*.
sender [object]	**saatja**	The object passing a stimulus to a receiver object. Contrast: *receiver*.
sequence diagram	**järgnevusskeem**	A diagram that shows object interactions arranged in time sequence. In particular, it shows the objects participating in the interaction and the sequence of messages exchanged. Unlike a collaboration diagram, a sequence diagram includes time sequences but does not include object relationships. A sequence diagram can exist in a generic form (describes all possible *scenarios*) and in an instance form (describes one actual scenario). Sequence diagrams and collaboration diagrams express similar information, but show it in different ways. See: *collaboration diagram.*
signal	**signaal**	The specification of an asynchronous stimulus

		communicated between instances. Signals may have parameters.	
signature	**signatuur**	The name and parameters of a behavioral feature. A signature may include an optional returned parameter.	
single inheritance	**ainupärilus**	A semantic variation of *generalization* in which a *type* may have only one *supertype*. Synonym: *multiple inheritance* [OMA]. Contrast: *multiple inheritance*.	
single valued [MOF]	**üheväärtuseline**	A model element with *multiplicity* defined is single valued when its Multiplicity Type:: upper attribute is set to one. The term single-valued does not pertain to the number of values held by an attribute, parameter, etc., at any point in time, since a single-valued attribute (for instance, with a multiplicity lower bound of zero) may have no value. Contrast: *multi-valued.*	
software architecture	**tarkvara arhitektuur**	Software architecture encompasses: (1) the significant decisions about the organization of a software system, (2)the selection of the structural elements and their interfaces by which the system is composed together with their behavior as specified in the collaboration among those elements, (3)the composition of the structural and behavioral elements into progressively larger subsystems, (4)the architectural style that guides this organization, these elements and their interfaces, their collaborations, and their composition. Software architecture is not only concerned with structure and behavior, but also with usage, functionality, performance, resilience, reuse, comprehensibility, economic and technology constraints and tradeoffs, and aesthetic concerns.	
Software Engineering Process Authority (SEPA)	**tarkvaratehniline protsessiorgan**	The organizational entity with responsibility for process definition, assessment and improvement (see Concepts: Organizational Context for the Rational Unified Process).	
software requirement	**tarkvaranõue**	A specification of an externally observable behavior of the system, (e.g., inputs to the system, outputs from the system, functions of the system, attributes of the system, or attributes of the system environment).	
software requirements specifications (SRS)	**terkvaranõuete spetsifikatsioon**	A set of requirements which completely defines the external behavior of the system to be built. (sometimes called a functional specification)	
software specification review (SSR)	**tarkvaraspetsifikatsiooni läbivaatus**	In the waterfall life-cycle, the major review held when the software requirements specification is complete (see Guidelines: Project Plan).	
specification	**spetsifikatsioon**	A declarative description of what something is or does. Contrast: *implementation.*	
stakeholder	**osanik**	An individual who is materially affected by the outcome of the system.	JOKA: **asjast huvitatu**. "Osanik jätab mulje, justkui oleks kah pappi sisse pannud juba, aga ei pruugi. Stakeholder

			võib olla ka potentsiaalne tulevane kasutaja, kes pole veel millegi eest maksnud, aga kelle arvamust kuulda võetakse.
stakeholder need	**osaniku tarve**	The business or operational problem (opportunity) that must be fulfilled in order to justify purchase or use.	
stakeholder request	**osaniku taotlus**	A request of any type (e.g., *Change Request*, *enhancement request*, request for a requirement change, *defect*) from a *stakeholder*.	
state	**olek**	A condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event. Contrast: *state* [OMA].	
statechart diagram	**olekuskeem**	A diagram that shows a state machine. See: *state machine*.	
state machine	**olekumasin**	A state machine specifies the behavior of a *model element*, defining its response to events and the life cycle of the object. A behavior that specifies the sequences of *states* that an object or an interaction goes through during its life in response to events, together with its responses and actions.	
static artifact	**staatiline tehis**	An artifact that is used, but not changed, by a process.	
static classification	**staatiline liigitus**	A semantic variation of *generalization* in which an object may not change type or may not change role. Contrast: *dynamic classification.*	
stereotype	**stereotüüp**	A meta-classification of an element. Stereotypes have semantic implications which can be specified for every specific stereotype value. See UML Stereotypes in the Rational Unified Process for information on the pre-defined stereotypes in use in the Rational Unified Process. A new type of modeling element that extends the semantics of the metamodel. Stereotypes must be based on certain existing types or classes in the metamodel. Stereotypes may extend the semantics, but not the structure of pre-existing types and classes. Certain stereotypes are predefined in the UML, others may be user defined.	
stimulus	**stiimul**	The passing of information from one instance to another, such as raising a *signal* or invoking an *operation.* The receipt of a signal is normally considered an *event*. See: *message*.	
string	**string**	A sequence of text characters. The details of string representation depend on implementation, and may include character sets that support international characters and graphics.	
structural feature	**struktuurne erisus**	A static feature of a *model element*, such as an *attribute*.	
structural model aspect	**struktuurne mudeliaspekt**	A model aspect that emphasizes the structure of the objects in a system, including their *types*, *classes*, *relationships*, *attributes*, and *operations.*	

stub	**makett**	A component containing functionality for testing purposes. A stub is either a pure "dummy", just returning some predefined values, or it is "simulating" a more complex behavior.
subactivity state	**alamtegevusolek**	A *state* in an *activity graph* that represents the execution of a non-atomic sequence of steps that has some duration.
subclass	**alamklass**	In a generalization relationship, the specialization of another class; the superclass. See: *generalization.* Contrast: *superclass.*
submachine state	**alammasinaolek**	A *state* in a *state machine* which is equivalent to a *composite state* but its contents is described by another state machine.
substate	**osaolek**	A state that is part of a *composite state.* See: *concurrent substate, disjoint substate.*
subsystem	**alamsüsteem**	A *model element* which has the semantics of a *package*, such that it can contain other model elements, and a *class*, such that it has behavior. (The behavior of the subsystem is provided by classes or other subsystems it contains). A subsystem realizes one or more interfaces, which define the behavior it can perform. A subsystem is a grouping of model elements, of which some constitute a specification of the behavior offered by the other contained model elements. See *package*. See: *system*.
subtype	**alamtüüp**	In a generalization relationship, the specialization of another type; the supertype. See: *generalization.* Contrast: *supertype.*
superclass	**ülaklass**	In a generalization relationship, the generalization of another class; the subclass. See: *generalization.* Contrast: *subclass.*
supertype	**ülatüüp**	In a generalization relationship, the generalization of another type; the subtype. See: *generalization.* Contrast: *subtype.*
supplier	**tarnija**	A classifier that provides services that can be invoked by others. Contrast: *client.*
swimlane	**rada**	A partition on a *activity diagram* for organizing the responsibilities for actions. Swimlanes typically correspond to organizational units in a business model. See: *partition.*
synch state	**sünkroolek**	A vertex in a *state machine* used for synchronizing the concurrent regions of a state machine.
synchronous action	**sünkroonne toiming**	A request where the sending object pauses to wait for results. Contrast: *asynchronous action*.
system	**süsteem**	As an instance, an executable configuration of a software application or software application family; the execution is done on a hardware platform. As a class, a particular software application or software application family that can be configured and installed on a hardware platform. In a general sense, an arbitrary system instance. 1. A collection of connected units that are organized to accomplish a specific purpose. A system can be described by one or more models, possibly from different viewpoints. Synonym: physical system. 2. A top-level subsystem.

system requirements review (SRR)	**süsteeminõuete läbivaatus**	In the waterfall life-cycle, the name of the major review held when the system specification is completed (see Guidelines: Project Plan).
tagged value	**sildiga väärtus**	The explicit definition of a property as a name-value pair. In a tagged value, the name is referred as the tag. Certain tags are predefined in the UML; others may be user defined. Tagged values are one of three extensibility mechanisms in UML. See: *constraint, stereotype.*
target (for test)	**testredaktsioon**	A build that is an object for testing. See: *build.*
task	**tegum**	See: *operating system process, process* and *thread.*
team leader	**tiimijuht**	The team leader is the interface between project management and developers. The team leader is responsible for ensuring that a task is allocated and monitored to completion. The team leader is responsible for ensuring that development staff follow project standards, and adhere to project schedules.
technical authority	**tehniline organ**	The project's technical authority has the authority and technical expertise to arbitrate on if, and how, a change request is to be implemented. The technical authority defines change tasks, and estimates the effort of engineering the work tasks (corresponding to a *change request*).
template	**tehisemall**	A pre-defined structure for an *artifact*. Synonym: *parameterized element*.
test	**test**	A *core process workflow* in the software-engineering process whose purpose is to integrate and test the system.
test case	**testjuhtum**	A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific *requirement.*
test coverage	**testi katvus**	The degree to which a given test or set of tests addresses all specified test cases for a given *system* or *component*.
test driver	**testidraiver**	A software module or application used to invoke a test item and, often, provide test inputs (data), control and monitor execution, and report test results. A test driver automates the execution of test procedures.
test item	**testredaktsioon**	A build which is an object of testing. See: *build.*
test procedure	**testimisprotseduur**	A test procedure is a set of detailed instructions for the set-up, execution, and evaluation of results for a given *test case*.
thread	**lõim**	An independent computation executing within an the execution environment and address space defined by an enclosing operating system process. Also sometimes called a 'lightweight process'.
thread [of control]	**käsulõim**	A single path of execution through a program, a dynamic model, or some other representation of control flow. Also, a stereotype for the implementation of an active object as lightweight process. See *process*.
time	**hetk**	A value representing an absolute or relative moment in time.

time event	**ajasündmus**	An event that denotes the time elapsed since the current state was entered. See: *event.*	
time expression	**ajaavaldis**	An expression that resolves to an absolute or relative value of time.	
timing mark	**ajamärgis**	A denotation for the time at which an event or message occurs. Timing marks may be used in constraints.	
tool mentor	**instrumentaaljuhis**	A description which provides practical guidance on how to perform specific process activities or steps using a specific software tool.	
traceability	**jälitatavus**	The ability to trace a project element to other related project elements, especially those related to *requirements*.	
trace	**jälg**	A dependency that indicates a historical or process relationship between two elements that represent the same concept without specific rules for deriving one from the other.	
transient object	**ajutine objekt**	An object that exists only during the execution of the process or thread that created it.	
transition	**siire**	The fourth *phase* of the process in which the software is turned over to the user community. A relationship between two *states* indicating that an object in the first state will perform certain specified actions and enter the second state when a specified event occurs and specified conditions are satisfied. On such a change of state, the transition is said to *fire*.	
type	**tüüp**	Description of a set of entities which share common characteristics, relations, attributes, and semantics. A stereotype of class that is used to specify a domain of instances (objects) together with the operations applicable to the objects. A type may not contain any methods. See: *class, instance*. Contrast: *interface*.	
type expression	**tüübiavaldis**	An expression that evaluates to a reference to one or more *types*.	
UML	**UML**	Unified Modeling Language [UML98]. In the Rational Unified Process Glossary, definitions from the Unified Modeling Language are indicated by the symbol:	
uninterpreted	**tõlgenduseta**	A placeholder for a type or types whose implementation is not specified by the UML. Every uninterpreted value has a corresponding string representation. See: *any* [CORBA].	
usage	**kasutus**	A dependency in which one element (the *client*) requires the presence of another element (the *supplier*) for its correct functioning or implementation.	
use case (class)	**kasutusklass**	A use case defines a set of *use-case instances*, where each instance is a sequence of actions a system performs that yields an observable result of value to a particular actor. A use-case class contains all main, alternate flows of events related to producing the 'observable result of value'. Technically, a use-case is a class whose instances are *scenarios*. The specification of a sequence of actions, including variants, that a system (or other entity) can perform, interacting with *actors* of the system. *See: use-case instances.*	JOKA: **kasutusjuhtum**. "Klass" on kuidagi liiga tehniline, kuna sellest terminist (RUP'i arvates) peaks ka klient aru saama. ARNE: sekundeerin jokale. Sama ka ülejäänud UC terminite puhul
use-case diagram	**kasutusklassiskeem**	A diagram that shows the relationships among actors and *use cases* within a system.	

use-case instance	**kasutusklassi isend**	A sequence of actions performed by a system that yields an observable result of value to a particular actor. The performance of a sequence of actions being specified in a *use case*. An instance of a use case. See: *use-case class*.
use-case model	**kasutusklassimudel**	A model that describes a system's functional *requirements* in terms of *use cases*.
use-case package	**kasutusklassipakett**	A use-case package is a collection of use cases, actors, relationships, diagrams, and other packages; it is used to structure the use-case model by dividing it into smaller parts.
use-case realization	**kasutusklassi teostus**	A use-case realization describes how a particular use case is realized within the *design model*, in terms of collaborating objects.
use-case view	**kasutusklassivaade**	An *architectural view* that describes how critical use cases are performed in the system, focusing mostly on architecturally significant components (objects, tasks, nodes). In the Unified Process, it is a view of the *use-case model*.
utility	**utiliit**	A stereotype that groups global variables and procedures in the form of a class declaration. The utility attributes and operations become global variables and global procedures, respectively. A utility is not a fundamental modeling construct, but a programming convenience.
version	**versioon**	A variant of some artifact; later versions of an artifact typically expand on earlier versions.
view	**vaade**	A simplified description (an abstraction) of a model, which is seen from a given perspective or vantage point and omits entities that are not relevant to this perspective. See also *architectural view.* A projection of a model, which is seen from a given perspective or vantage point and omits entities that are not relevant to this perspective.
view element	**vaate element**	A view element is a textual and/or graphical projection of a collection of *model elements.*
view projection	**vaate projektsioon**	A projection of *model elements* onto *view elements.* A view projection provides a location and a style for each view element.
visibility	**nähtavus**	An enumeration whose value (public, protected, or private) denotes how the *model element* to which it refers may be seen outside its enclosing *namespace*.
vision	**nägemus**	The user's or *customer's* view of the product to be developed, specified at the level of key *stakeholder needs* and *features* of the system.
value	**väärtus**	An element of a type domain.
vertex	**tipp**	A source or a target for a transition in a state machine. A vertex can be either a state or a pseudo-state. See: *state, pseudo-state*.
work guideline	**tööjuhis**	A description which provides practical guidance on how to perform an activity or set of activities. It usually considers techniques which are useful during the activity.
worker	**töötaja**	A definition of the behavior and responsibilities of an individual, or a set of individuals working together as a team, within the context of a software engineering

		organization. The worker represents a *role* played by individuals on a project, and defines how they carry out work.
workflow	**töövoog**	The sequence of activities performed in a business that produces a result of observable value to an individual actor of the business.
workflow detail	**töövoolõik**	A grouping of activities which are performed in close collaboration to accomplish some result. The activities are typically performed either in parallel or iteratively, with the output from one activity serving as the input to another activity. Workflow details are used to group activities to provide a higher level of abstraction and to improve the comprehensibility of workflows.