IEGULDĪJUMS TAVĀ NĀKOTNĒ

Semi-automatic Generation of a Software Testing Lightweight Ontology from a Glossary Based on the ONTO6 Methodology

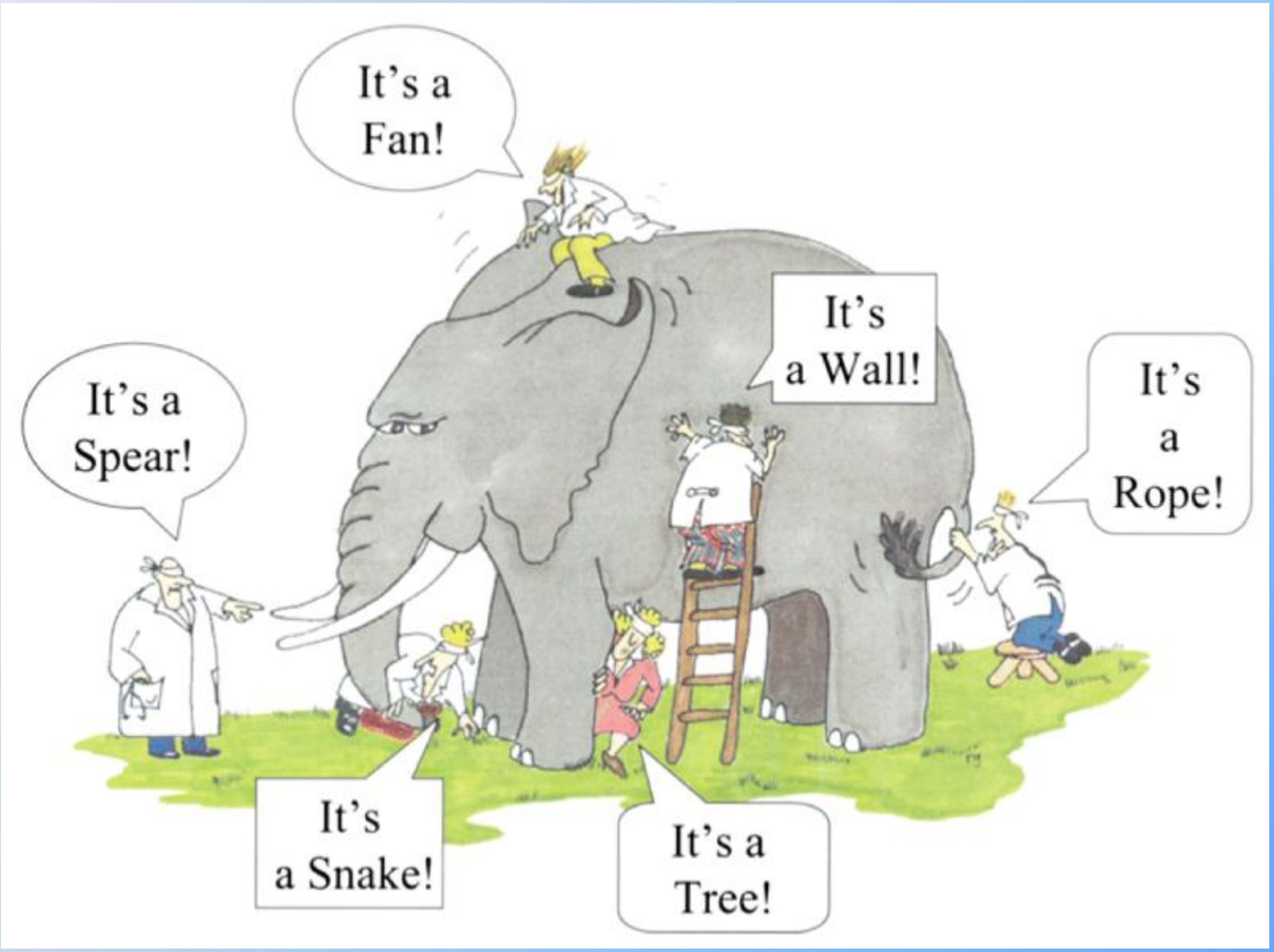
Guntis Arnicans, Dainis Romans, Uldis Straujums

University of Latvia

Tenth International Baltic Conference on Databases and Information Systems (Baltic DB&IS 2012)

The work is partly supported by a European Social Fund Project No. 2009/0216/1DP/1.1.1.2.0/09 /APIA/VIAA/044

Blind men and an elephant



Glossary of a domain

Software Testing Glossary

Last updated: Thursday, 24-May-2012 05:03:00 PDT

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A (return to top of page)

Acceptance Testing: Testing conducted to enable a user/customer to determine whether to accept a software product. Normally performed to validate the software meets a set of agreed acceptance criteria.

Accessibility Testing: Verifying a product is accessible to the people having disabilities (deaf, blind, mentally disabled etc.).

Ad Hoc Testing: A testing phase where the tester tries to 'break' the system by randomly trying the system's functionality. Can include negative testing as well. See also Monkey Testing.

Agile Testing: Testing practice for projects using agile methodologies, treating development as the customer of testing and emphasizing a test-first design paradigm. See also Test Driven Development.

Application Binary Interface (ABI): A specification defining requirements for portability of applications in binary forms across defferent system platforms and environments.

Application Programming Interface (API): A formalized set of software calls and routines that can be referenced by an application program in order to access supporting system or network services.

Automated Software Quality (ASQ): The use of software tools, such as automated testing tools, to improve software quality.

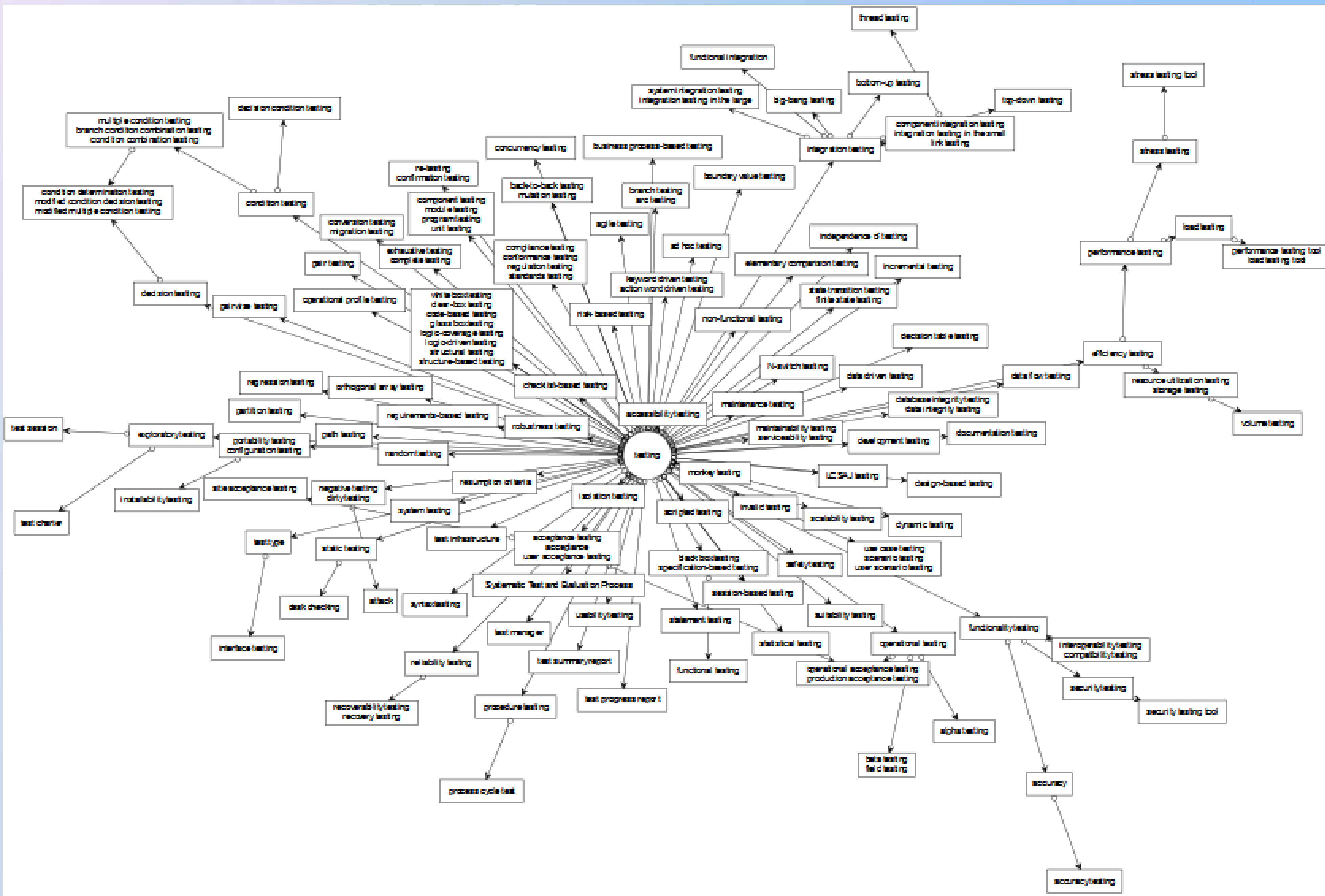
Automated Testing:

- Testing employing software tools which execute tests without manual intervention. Can be applied in GUI, performance, API, etc. testing.
- The use of software to control the execution of tests, the comparison of actual outcomes to predicted outcomes, the setting up of test preconditions, and other test control and test reporting functions.

B (return to top of page)

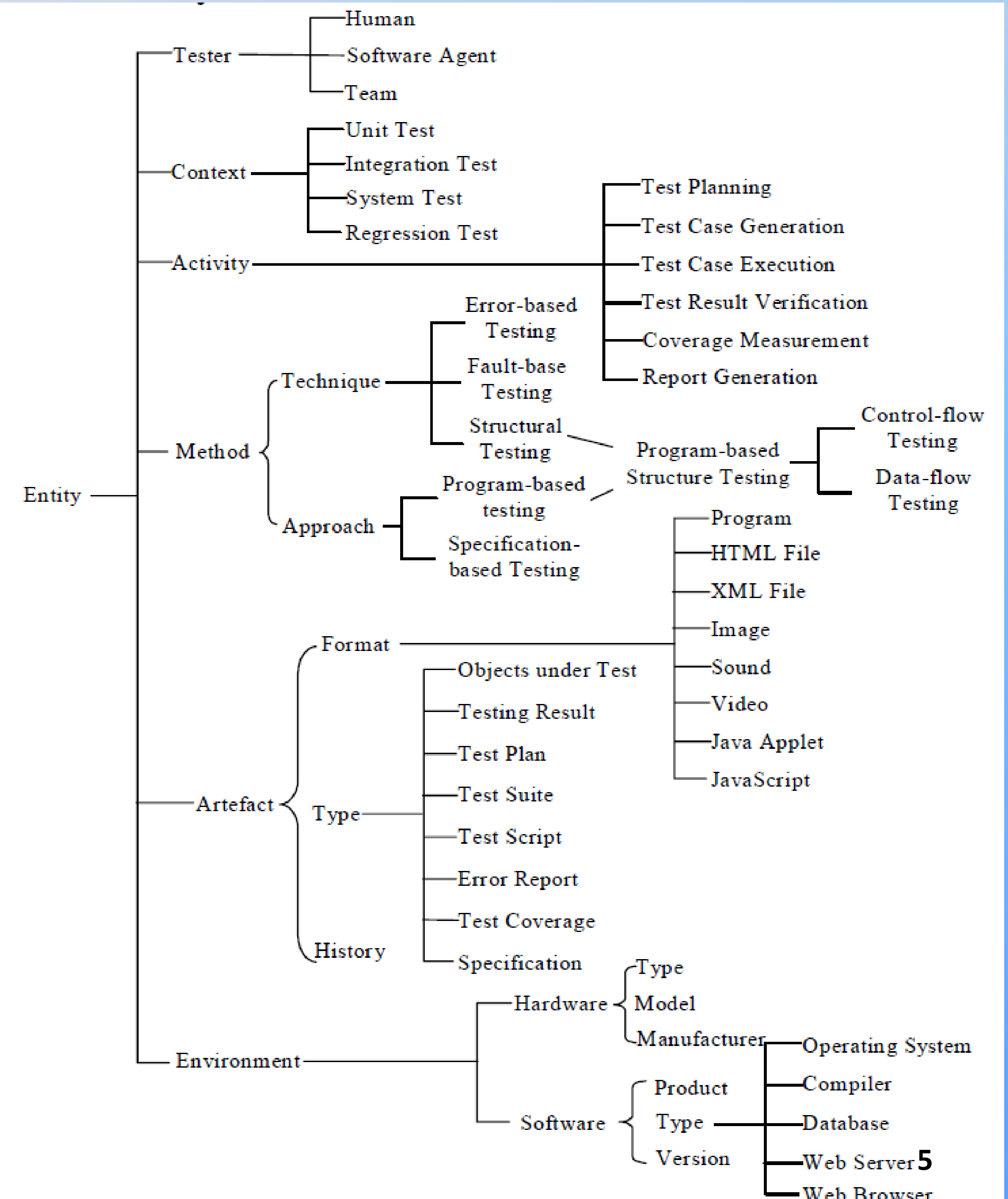
Backus-Naur Form: A metalanguage used to formally describe the syntax of a language.

Lightweight ontology of a domain



Domain ontology developed by experts

- H. Zhu and Q. Huo, 2005
- Ontology for an agent-based software environment to test web-based applications
- About 100 concepts



Related works (1/2)

Proposal of parallel construction of domain ontology and construction of complete domain terminology.

L. Bozzato, M. Ferrari, and A. Trombetta. ***Building a domain ontology from glossaries: a general methodology***. In A. Gangemi, J. Keizer, V. Presutti, and H. Stoermer, editors, Semantic Web Applications and Perspectives, SWAP 2008, volume 426 of CEUR Proceedings, 2008.

Related works (2/2)

Obtaining of the ontology OntoGLOSE from the "*IEEE Standard Glossary of Software Engineering Terminology*".

Creating in some phases uses semi-automatic steps and uses semi-automatic linguistic analysis.

(No details of the automatization and results available)

Hilera José R., Pages Carmen, Martinez J. Javier, Gutierrez J. Antonio, De-Marcos Luis, ***An Evolutive Process to Convert Glossaries into Ontologies,*** Information technology and libraries, vol. 29, no4(**2010**), 195-204.

Principles stated by Noy and McGuinness (2001)

<u>Principle 1</u>: "*There is no one correct way to model a domain — there are always viable alternatives. The best solution almost always depends on the application that you have in mind and the extensions that you anticipate*";

<u>Principle 2</u>: "*Ontology development is necessarily an iterative process*";

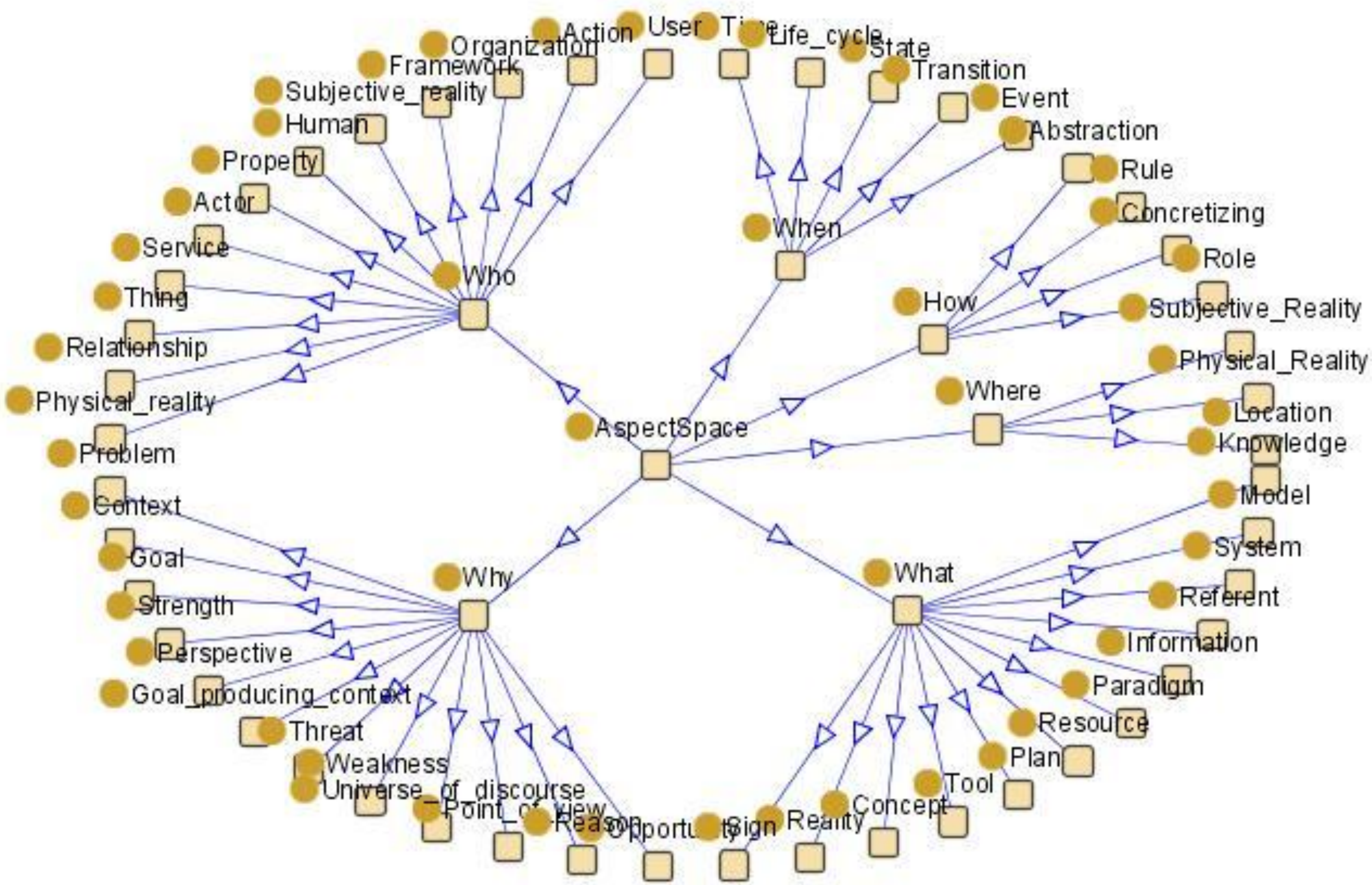
<u>Principle 3</u>: "*Concepts in the ontology should be close to objects (physical or logical) and relationships in your domain of interest. These are most likely to be nouns (objects) or verbs (relationships) in sentences that describe your domain*".

Classic steps to obtain an initial ontology

Noy and McGuinness (2001):

1. *Determine the domain and scope of the ontology;*
2. *Consider reusing existing ontologies;*
3. *Enumerate important terms in the ontology;*
4. *Define the classes and the class hierarchy;*
5. *Define the properties of classes-slots;*
6. *Define the facets of the slots;*
7. *Create instances.*

ONTO6 Meta-Ontology top level simplified visualization



Source glossary

ISTQB®
International Software
Testing Qualifications Board

Standard glossary of terms used in Software Testing

Version 2.1 (dd. April 1st, 2010)

Produced by the ‘Glossary Working Party’
International Software Testing Qualifications Board

Standard glossary of terms used in Software Testing

bottom-up testing: An incremental approach to integration testing where the lowest level components are tested first, and then used to facilitate the testing of higher level components. This process is repeated until the component at the top of the hierarchy is tested. See also *integration testing*.

boundary value: An input value or output value which is on the edge of an equivalence partition or at the smallest incremental distance on either side of an edge, for example the minimum or maximum value of a range.

boundary value analysis: A black box test design technique in which test cases are designed based on boundary values. See also *boundary value*.

boundary value coverage: The percentage of boundary values that have been exercised by a test suite.

boundary value testing: See *boundary value analysis*.

branch: A basic block that can be selected for execution based on a program construct in which one of two or more alternative program paths is available, e.g. case, jump, go to, if-then-else.

- The glossary contains 724 entries
- For comparison, "IEEE Standard Glossary of Software Engineering Terminology" (1990) contains approximately 1300 entries

Structure of the glossary

black box testing: Testing, either functional or non-functional, without reference to the internal structure of the component or system.

specification-based testing: See *black box testing*.

functional testing: Testing based on an analysis of the specification of the functionality of a component or system. See also *black box testing*.

configuration control board (CCB): A group of people responsible for evaluating and approving or disapproving proposed changes to configuration items, and for ensuring implementation of approved changes. [IEEE 610]

Structure of the glossary

black box testing: Testing, either functional or non-functional, without reference to the internal structure of the component or system.

specification-based testing: See *black box testing*.

functional testing: Testing based on an analysis of the specification of the functionality of a component or system. See also *black box testing*.

configuration control board (CCB): A group of people responsible for evaluating and approving or disapproving proposed changes to configuration items, and for ensuring implementation of approved changes. [IEEE 610]

Entries

Structure of the glossary

black box testing: Testing, either functional or non-functional, without reference to the internal structure of the component or system.

specification-based testing: See *black box testing*.

functional testing: Testing based on an analysis of the specification of the functionality of a component or system. See also *black box testing*.

configuration control board (CCB): A group of people responsible for evaluating and approving or disapproving proposed changes to configuration items, and for ensuring implementation of approved changes. [IEEE 610]

Term Definition

Structure of the glossary

black box testing: Testing, either functional or non-functional, without reference to the internal structure of the component or system.

specification-based testing: See *black box testing*.

functional testing: Testing based on an analysis of the specification of the functionality of a component or system. See also *black box testing*.

configuration control board (CCB): A group of people responsible for evaluating and approving or disapproving proposed changes to configuration items, and for ensuring implementation of approved changes. [IEEE 610]

Source Cross-reference Acronym Synonym

Finding of significant aspects (words)

configuration control board (CCB): A group of people responsible for evaluating and approving or disapproving proposed changes to configuration items, and for ensuring implementation of approved changes. [IEEE 610]

We can observe that:

1. The most semantically significant word of a term is at right hand side, usually it is the last word of term;
2. The most semantically significant word or words of definition are located at the beginning part of definition.

entry normalization

functional testing: Testing based on an analysis of the specification of the functionality of a component or system. See also *black box testing*.

↓

functional testing : testing based analysis specification functionality component system see black box testing

Indexing of words

- Assign an index to each instance of word
  - from right to left in term
  - from left to right in definition

functional(1) testing(0) : testing(0) based(1) analysis(2) specification(3) functionality(4) component(5) system(6) see(7) black(8) box(9) testing(10)

Weighting of words

- Assign a weight to each instance of word
- Formula: $\mathbf{2^{-word\_index}}$

functional(2^{-1}) testing(2^{0}) : testing(2^{0}) based(2^{-1}) analysis(2^{-2}) specification(2^{-3}) functionality(2^{-4}) component(2^{-5}) system(2^{-6}) see(2^{-7}) black(2^{-8}) box(2^{-9}) testing(2^{-10})

Total weight for word «testing» in the entry is

$$\mathbf{2^{1} + 2^{1} + 2^{-10} = 2.0009765625}$$

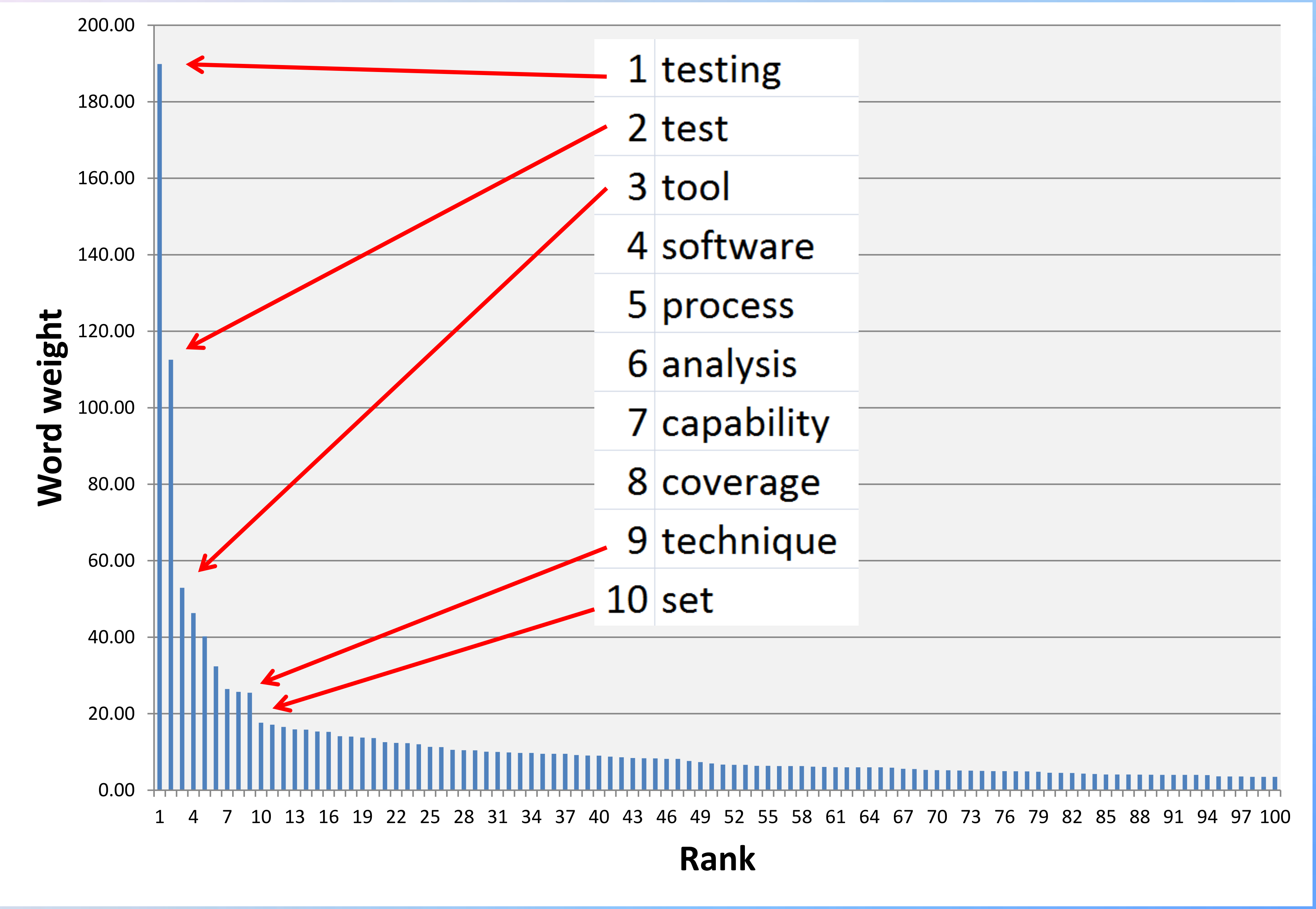
Word weighting process result (1/2)

Rank	Count	Word (counting)	Word (weighting)	Weight
1	494	**test**	**testing**	**189.85**
2	318	**testing**	**test**	**112.54**
3	165	**software**	*tool*	**52.91**
4	130	*see*	**software**	**46.30**
5	129	**system**	**process**	**40.23**
6	116	**component**	*analysis*	**32.40**
7	112	**process**	*capability*	**26.47**
8	81	*product*	**coverage**	**25.72**
9	80	*IEEE*	*technique*	**25.50**
10	80	**quality**	*set*	17.64
11	73	*after*	**component**	17.14
12	70	*tool*	**quality**	16.54
13	69	*design*	**condition**	15.91
14	61	*technique*	*model*	15.84
15	59	*execution*	*management*	15.35
16	58	*analysis*	*percentage*	15.25
17	56	**coverage**	**system**	14.11
18	53	*610*	*report*	14.01
19	49	*management*	*box*	13.75
20	48	*data*	*document*	13.64
21	47	**condition**	*black*	12.57
22	46	*requirements*	*design*	12.38
23	46	*model*	**review**	12.30
24	43	*e.g*	*product*	12.02
25	42	*control*	**case**	11.33
26	41	*development*	**result**	11.27
27	41	*level*	*white*	10.56
28	40	*ISO*	**risk**	10.47
29	39	*activities*	*approach*	10.42
30	38	*capability*	*degree*	10.08
31	38	*based*	**specification**	10.00
32	38	*set*	*level*	9.88
33	37	*specified*	**input**	9.76
34	36	*phase*	*criteria*	9.76
35	35	*determine*	**statement**	9.53
36	35	**defect**	**path**	9.53
37	34	**result**	*type*	9.53
38	34	**input**	*procedure*	9.20
39	34	**performance**	*representation*	9.06
40	34	**decision**	*execution*	9.04

Word weighting process result (2/2)

Rank	Count	Word (counting)	Word (weighting)	Weight
1	494	**test**	**testing**	**189.85**
2	318	**testing**	**test**	**112.54**
3	165	**software**	*tool*	**52.91**
4	130	*see*	**software**	**46.30**
5	129	**system**	**process**	**40.23**
6	116	**component**	*analysis*	**32.40**
7	112	**process**	*capability*	**26.47**
8	81	*product*	**coverage**	**25.72**
9	80	*IEEE*	*technique*	**25.50**
10	80	**quality**	*set*	17.64
11	73	*after*	**component**	17.14
12	70	*tool*	**quality**	16.54
13	69	*design*	**condition**	15.91
14	61	*technique*	*model*	15.84
15	59	*execution*	*management*	15.35
16	58	*analysis*	*percentage*	15.25
17	56	**coverage**	**system**	14.11
18	53	*610*	*report*	14.01
19	49	*management*	*box*	13.75
20	48	*data*	*document*	13.64
21	47	**condition**	*black*	12.57

Word weight distribution



Creation of the aspect ontology

```
createAspectOntology(set glossary, string aspect)
    aspectEntrySet = createEntrySet(glossary, aspect)
    aspectGraph = createAspectGraph(aspect, aspectEntrySet)
    mergeSynonyms(graph aspectGraph)
    reduceRelations(graph aspectGraph)
    aspectOwlDesription = generateOwlDescription(graph aspectGraph)
    // a generation to any other output format may be placed here
    // for instance, DOT language scripts for Graphviz
end createAspectOntology
```

Conditions used in ontology creation

cond_term_1(string *term*, string *pattern*): bool – checks whether the word *pattern* **is among** words in the *term*;

cond_term_2(string *term*, string *pattern*): bool – checks whether the sequence of words *pattern* is **at the very beginning** of the sequence of words *term*;

cond_term_3(string *term*, string *pattern*): bool – checks whether the sequence of words *pattern* **is at the very end** of the sequence of words *term*;

cond_def_1(string *definition*, string *pattern*, int *n*): bool – checks whether the sequence of words *pattern* **is at the beginning** of the sequence of words *definition*, skipping not more than *n* words;

cond_def_2(string *definition*, string *ref_pattern*, string *pattern*): bool – checks whether the sequence of words *pattern* is **at the beginning** of the sequence of words *definition*, **and corresponds** to pattern *ref_pattern* (for instance *ref_pattern* = "see <word_list>");

cond_def_3(string *definition*, string *ref_pattern*, string *pattern*): bool – checks whether the sequence of words *pattern* **is at the end part** of the sequence of words *definition*, **and corresponds** to pattern *ref_pattern* (for instance *ref_pattern* = "see also <word_list>").

Creation of the entry set (1/2)

```
createEntrySet(set glossary, string aspect)
  aspectEntrySet = EMPTY_SET
  for each entry of glosary
    if cond_term_1(entry.term, aspect) or
       cond_def_1(entry.definition, aspect, N) or
       cond_def_2(entry.definition, SYNONYM_REF_PATTERN, aspect) or
       cond_def_3(entry.definition, SEE_ALSO_REF_PATTERN, aspect)
      put entry into aspectEntrySet
  return aspectEntrySet
end createEntrySet
```

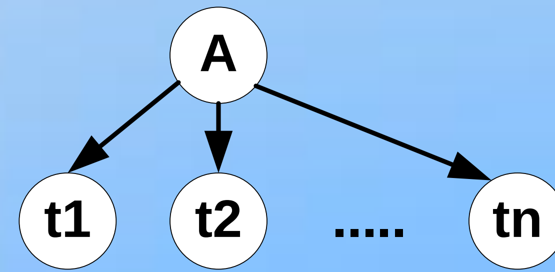
Creation of the entry set (2/2)

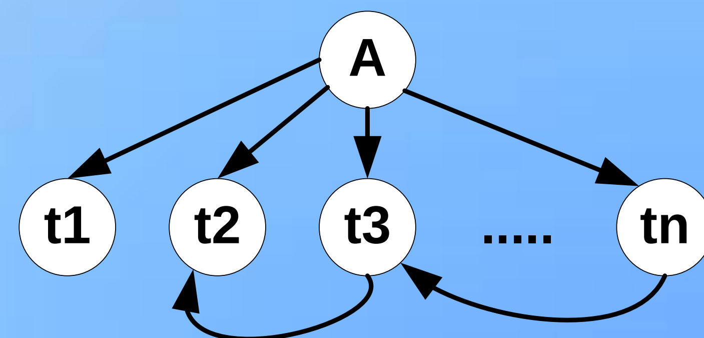
```
createEntrySet(set glossary, string aspect)
  aspectEntrySet = EMPTY_SET
  for each entry of glosary
    if cond_term_1(entry.term, aspect) or
      cond_def_1(entry.definition, aspect, N) or
      cond_def_2(entry.definition, SYNONYM_REF_PATTERN, aspect) or
      cond_def_3(entry.definition, SEE_ALSO_REF_PATTERN, aspect)
     put entry into aspectEntrySet
  return aspectEntrySet
end createEntrySet
```

Modify this expression for your own more sofisticate algorithm!

Creation of an aspect graph

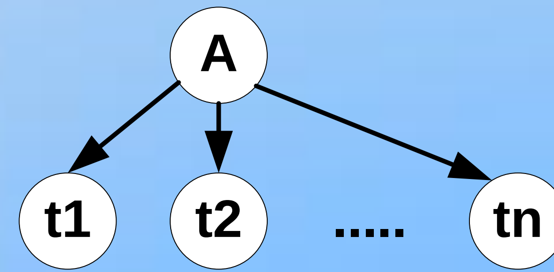
```
createAspectGraph(string aspect, set entrySet): graph
 aspectGraph = EMPTY_GRAPH
 add aspect as aspectNode of type Node into aspectGraph
 for each entry of entrySet
   add entry as entryNode of type Node into aspectGraph
   put entryNode into aspectNode.children

 for each node_1 of aspectGraph
   for each node_2 of aspectGraph
     if node_1 <> node_2 and
        node_1 <> aspectNode and
        node_1 <> aspectNode
       if cond_term_2(node_2.term, node_1.term) or
          cond_term_3(node_2.term, node_1.term) or
          cond_def_1(node_2.definition, node_1.term, N) or
          cond_def_3( (node_2.definition, SEE_ALSO_REF_PATTERN, node_1.term)
         put node_2 into node_1.children
 return aspectGraph
end createAspectGraph
```

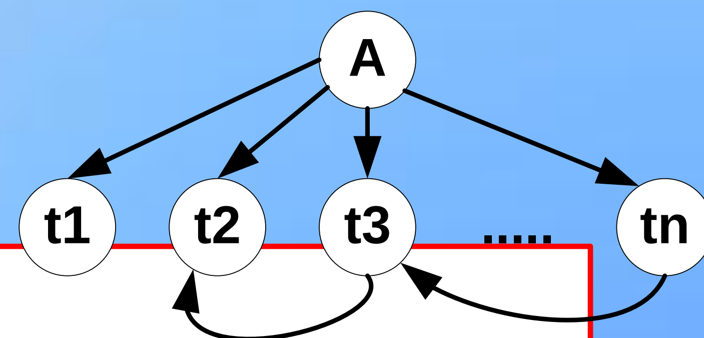




Creation of an aspect graph

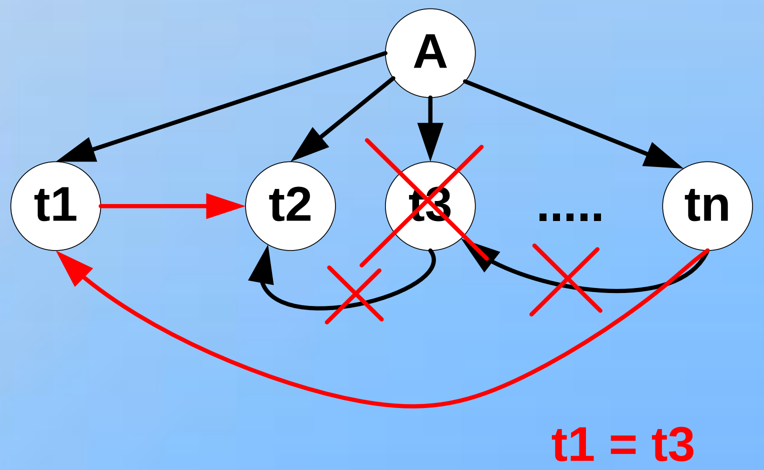
```
createAspectGraph(string aspect, set entrySet): graph
  aspectGraph = EMPTY_GRAPH
  add aspect as aspectNode of type Node into aspectGraph
  for each entry of entrySet
    add entry as entryNode of type Node into aspectGraph
    put entryNode into aspectNode.children

  for each node_1 of aspectGraph
    for each node_2 of aspectGraph
      if node_1 <> node_2 and
         node_1 <> aspectNode and
         node_1 <> aspectNode
        if cond_term_2(node_2.term, node_1.term) or
          cond_term_3(node_2.term, node_1.term) or
          cond_def_1(node_2.definition, node_1.term, N) or
          cond_def_3( (node_2.definition, SEE_ALSO_REF_PATTERN, node_1.term)
          put node_2 into node_1.children
  return aspectGraph
end createAspectGraph
```





Modify this expression for your own more sophisticated algorithm!

Merging of synonyms



mergeSynonyms(graph aspectGraph)

for each node_1 of aspectGraph

for each node_2 of aspectGraph

if node_1 <> node_2 and

node_1 <> aspectNode and

node_1 <> aspectNode

if **cond_def_2**(node_2.definition, SEE_REF_PATTERN, node_1.term)

put node_2.term into node_1.synonyms

put all node_2.children into node_1.children

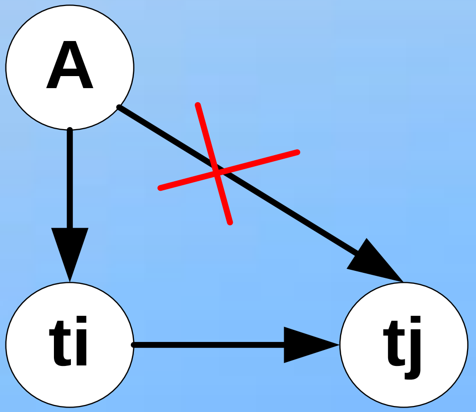
for each node_3 of aspectGraph

replace node_2 with node_1 in node_3.children

delete node_2 from aspectGraph

end mergeSynonyms

Reducing of relations




```
reduceRelations(graph aspectGraph)
 for each node_1 of aspectGraph
  for each node_2 of aspectGraph
   if node_1 <> node_2
    if node_2 is in node_1.children and
      existIndirectPathBetween(node_1, node_2)
     delete node_2 from node_1.children
end reduceRelations
```

This algorithm assumes that all relations have the same type! The algorithm has to be improved for the next iterations taking into account the types of relations.

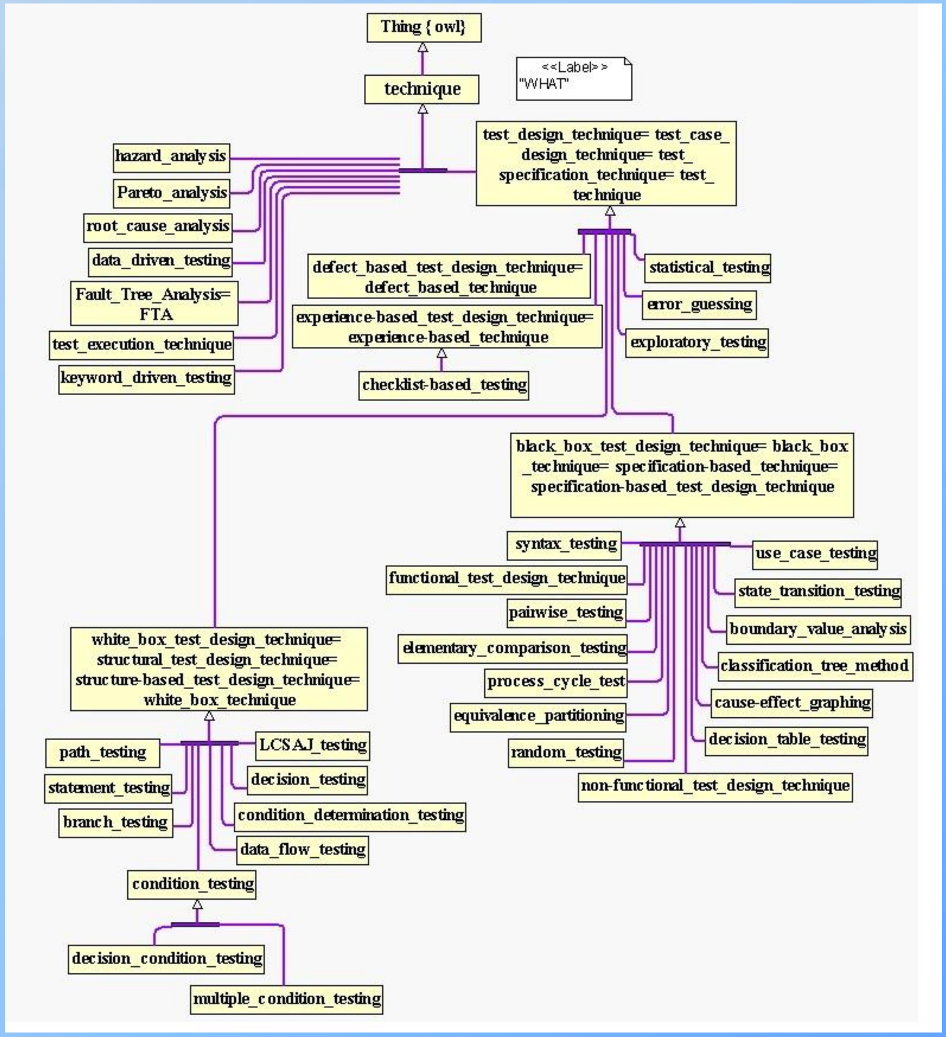
Results

- Obtained lightweight ontology
  - is exported in OWL RDF/XML notation;
  - is imported into the ontology creation environment Protégé;
  - is visualized by the graphical tool OWLGrEd.
- We plan to use the OWLGrEd to refine the ontology and store the refinements for the next iterations

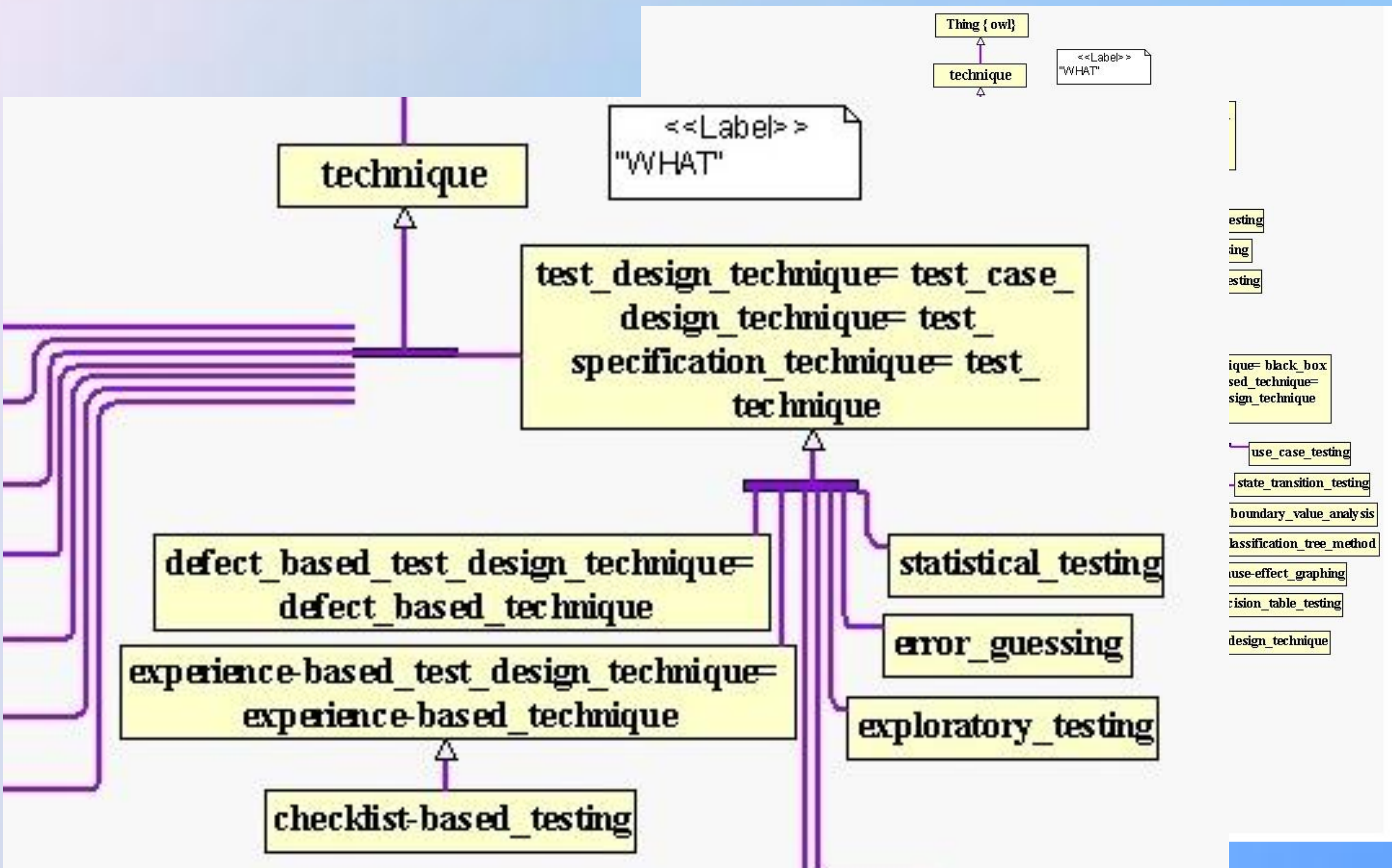
Evaluation of the results

Following ONTO6 methodology only 9 aspects are taken (WHAT): *testing*, *test*, *tool*, *software*, *process*, *analysis*, *capability*, *technique*.

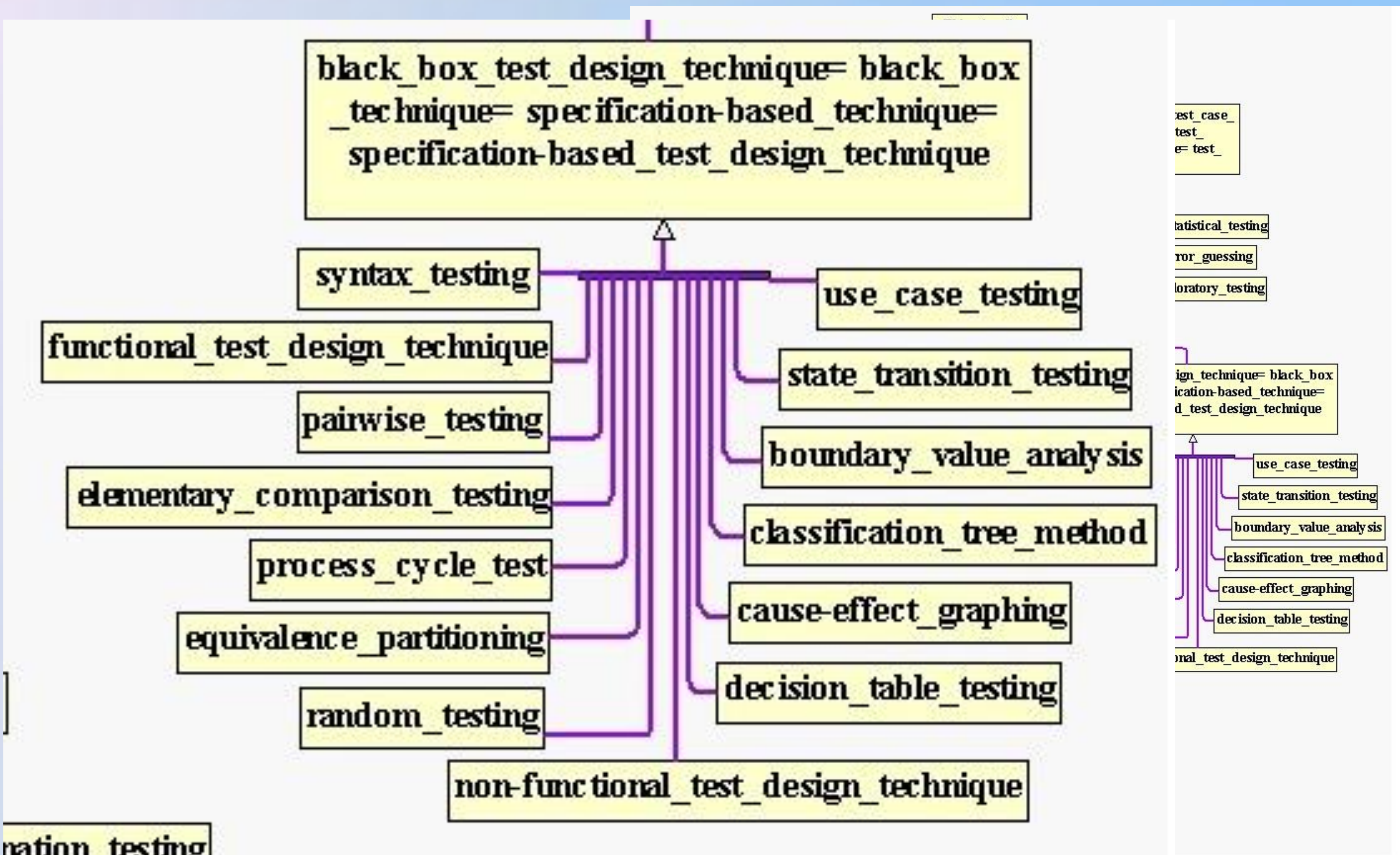
These **9** aspects serve as roots for the 629 unique entries from the 724 entries included in the glossary (**87%**).



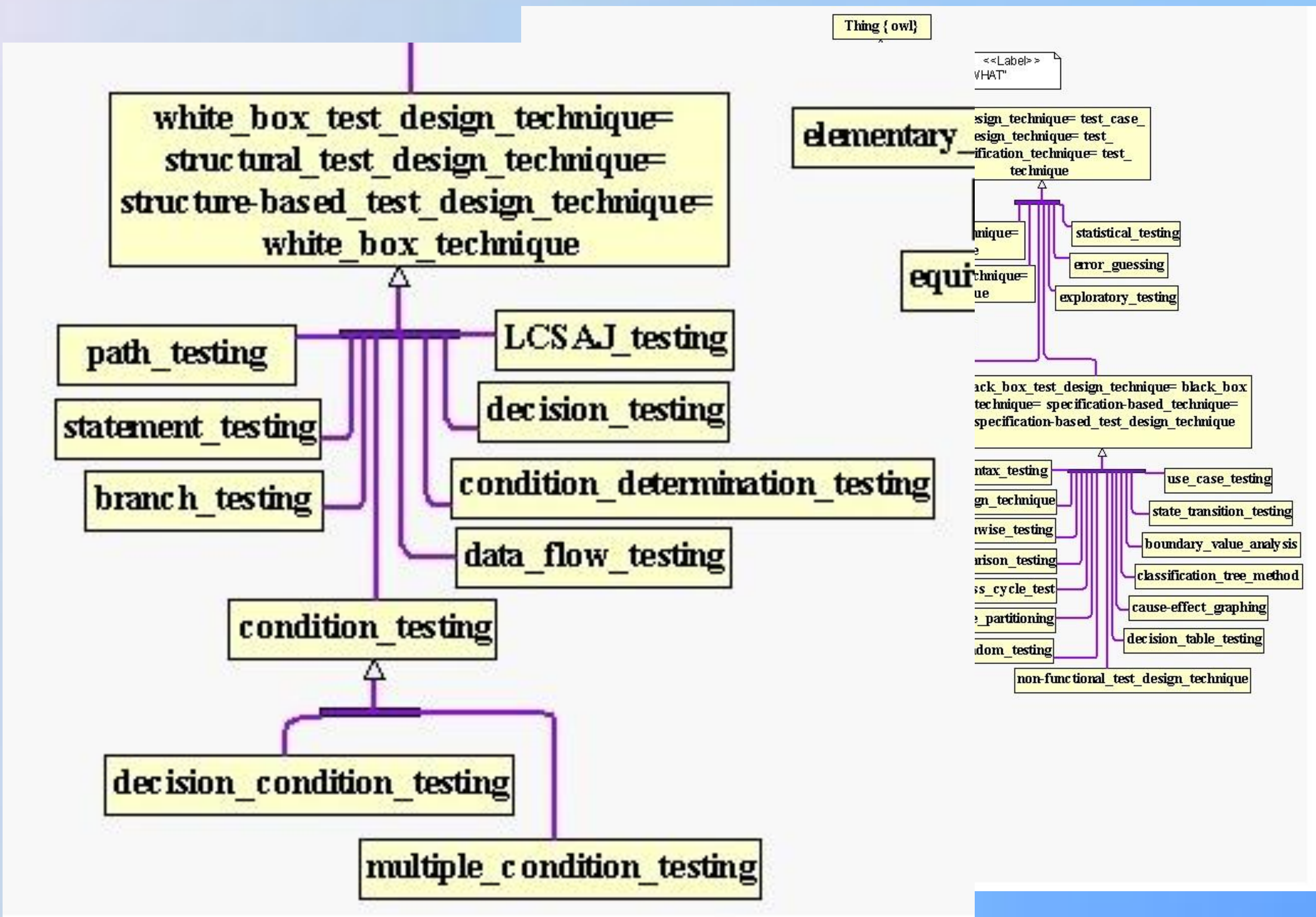
Ontology aspect *Technique* (1/3)



Ontology aspect *Technique* (2/3)



Ontology aspect *Technique* (3/3)



Demo

- Top 9 aspects (integrated)
- Top 40 aspects with definitions (not integrated)

Conclusion and future works

- It is possible to semi-automatically generate a lightweight ontology from glossary
- We offer the principles and algorithms how to discover the significant concepts and to find simple relations between concepts

- We are going to develop the methodology for the next iterations to improve the initially created ontology and create useful *Software Testing* ontology for the teaching purpose.

Thank you very much for your attention