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

Guntis ARNICANS[a,1], Dainis ROMANS[b] and Uldis STRAUJUMS[a]
[a] *Faculty of Computing, University of Latvia, Latvia*
[b] *University of Latvia, Latvia*

Abstract. We propose a methodology of semi-automatic obtaining of a lightweight ontology for software testing domain based on the glossary "Standard glossary of terms used in Software Testing" created by ISTQB. From the same glossary many ontologies might be developed depending on the strategy for extracting concepts, categorizing them, and determining hierarchical and some other relationships. Initially we use the ONTO6 methodology that allows identification of the most important aspects in the given domain. These identified aspects serve as the most general concepts in taxonomy (roots in the concept hierarchy). By applying natural language processing techniques and analyzing the discovered relations between concepts, an intermediate representation of lightweight ontology is created. Afterwards the lightweight ontology is exported to OWL format, stored in the ontology editor Protégé, and analyzed and refined by OWLGrEd – an UML style graphical editor for OWL that interoperates with Protégé. The obtained lightweight ontology might be useful for building up a heavyweight software testing ontology.

Keywords. Software testing ontology, natural language processing, ONTO6 methodology

Introduction

Despite the fact that many information and communication technology activities and projects are initiated and performed in the last decade, many information technology (IT) specialists suffer from the lack of appropriate ontologies for the corresponding domain. Software testing is one of the IT fields where the demand for useful ontologies in various domains grows, but there is no desirable offer neither from research nor industry organizations. Software testing is a very complex activity. There are various methodologies developed to test software, however the success in testing still depends on testers and their ability to choose the best techniques and tactics for the given situation. A huge amount of knowledge is accumulated, but it is not yet structured. As consequences every new player (tester) in this IT field spends much time and other resources to understand even the basic concepts and relationships between them.

[1] Corresponding Author: Department of Programming, Faculty of Computing, University of Latvia, 19 Raina Blvd., Riga, LV-1589, Latvia; E-mail: guntis.arnicans@lu.lv.

The paper [1] contains reviews and classification of ontologies developed for software engineering. Authors present efforts on applying the Semantic Web techniques on different software engineering aspects, present achieved results and future research directions. The whole software engineering ontology can be created by integrating smaller domain specific ontologies. Among 19 classes of ontologies two ones are directly related to software testing (*Testing Ontology, Defect Ontology*), and three ones also include testing issues (*Software Process Ontology, Quality Ontology, Technology Ontology*). One of the drawbacks of used approaches is "*The main goal of these approaches is to better manage the information. There used to be less help for the software engineers to access the needed information from the large amount of information or to understand the information.*" [1]. Goal of our research is to offer methodology that can help to understand the information.

Let us have a closer look at the needed Testing Ontology class. In the review [1] we succeeded to find only one reference [2] to such kind of ontology. This ontology initially was developed for multi-agent system that performs automatic testing [3]. The latest public available version might be the most developed software testing ontology, although it is developed for specific domain (ontology needed for multi-agent system that tests web-based applications). The highest level of ontology defines following concepts: *Tester*, *Environment*, *Context*, *Artefact*, *Method*, and *Activity*. Totally this ontology consists of about 100 concepts.

Another activities related to software testing ontology are the creation of ontology *OntoTest* built on basis of *ISO/IEC 12207 standard* [4], and the creation of aspect-based software architecture *RefTEST* that comprises the knowledge to develop testing tools [5].

There are various initiatives to "bring order" into software engineering knowledge engineering. The paper [6] describes problems like – why it is so hard to create useful ontologies. It offers methodology and tools used in project *Guide to the Software Engineering Body of Knowledge (SWEBOK)* [7]. The project has defined Software Testing as the knowledge area. The ontology for software testing is not developed yet.

We propose a methodology for creating software testing ontology from a glossary semi-automatically. Our process is similar to some other approaches: original document is converted to text file, most important concepts are extracted, relationships between concepts are discovered, graph of related concepts is built up, description of ontology is generated using OWL RDF/XML notation, the ontology is imported into ontology creation environment Protégé, and visualized by some graphical tool (OWLGrEd in our case). Our methodology for ontology creation – the ONTO6 methodology [8] differs by its conceptualization model from other well-known methodologies [9] – Uschold and King's, Grüninger and Fox's and METHONTOLOGY. ONTO6 methodology uses a knowledge model that contains the meta-concept – aspect space. The aspect set in ONTO6 methodology is chosen to be {What, Where, When, How, Why, Who}.

The main contribution of this paper is the description of an approach and algorithms that help to recognize the most important aspects from the aspect set of ontology and the concepts that belong to these aspects according to the glossary of the given domain.

The paper is structured as follows. In the first section we give the background of our research and mention related works with similar approaches. There are several studies about creation of ontologies from glossaries, but building process is not automated or it is at low level. An overview of ONTO6 methodology is presented and short description of our methodology is provided. The second section contains information about the exploited glossary and our original method for determining the important aspects of software testing, and the revealed aspects are given. The creation of the graph that represents ontology is described in the third section. This section contains algorithms for full automation of this task. The fourth section offers the results we have achieved; an excerpt from the ontology is given.

1. Development of a Lightweight Ontology from Glossary

The development of ontology from a glossary is a known approach, the general idea and methodology of this approach in a simplified way are given in [10]. But the whole potential of this approach it is not yet exploited.

1.1. Methodologies for Building up Ontotogy From Glossary

In [10] the authors propose parallel construction of domain ontology and construction of complete domain terminology. The methodology includes several phases: *Clustering, Saturation, Relationship identification, Disambiguation, Class grouping, Conceptual modelling, Schema representation, Ontology representation, Annotation*. Each phase is explained, the input and output is defined. The methodology is demonstrated by example – realization of a project for definition and representation of a standard terminology for *orthodontics* (medical domain). All phases are performed manually and automatization is mentioned for the future research.

More detailed description of such methodology is given in [13]. The methodology is applied to obtain the ontology *OntoGLOSE* from the “IEEE Standard Glossary of Software Engineering Terminology”. The glossary defines approximately 1300 terms. Ontology created includes 1521 classes and 324 properties or relationships. Creating in some phases uses semi-automatic steps and semi-automatic linguistic analysis. Unfortunately the paper does not contain precise information what was automated and how it was implemented. Similar to authors of *orthodontics ontology* [10] authors of *OntoGLOSE* are going to use the obtained ontology to improve vocabulary.

There are various methods how to automate some steps in ontology development process based on documents. A comparative analysis of experiences and exploited software for automatic ontology generation is given in [11]. The most difficult task at first steps is the finding of the most important concepts, categorizing them and creating of a meaningful hierarchy and relationships between concepts. Methods for data clustering can be used, for instance. TaxGen Framework can generate taxonomy for a collection of previously unstructured documents [12].

In the case, when documents are semi-structured, it is easier to generate taxonomy of most important concepts revealed. We consider a glossary as a semi-structured document.

1.2. ONTO6 Methodology

In our methodology we take some principles from ONTO6 methodology [8]. Initially ONTO6 was developed to conceptualize informatization in several domains. But ONTO6 methodology contains principles that make possible to apply it in more general way to find the essential concepts of the given domain and to create the appropriate ontology.

The ONTO6 methodology makes use of the 6W framework: What, Where, When, How, Why, Who.

It is aimed at identifying concepts, determining the interaction between objects corresponding to those concepts and determining the functionality of the objects.

The ONTO6 methodology is based on a semi-informal meta-ontology.

The stages of applying the ONTO6 methodology are:

- the development of a meta-ontology instance appropriate for the domain to be conceptualized;
- the development of an initial ontology from the meta-ontology instance; and
- the gradual refinement of domain concepts that appear in the initial ontology – the development of enriched initial ontologies.

The end result is an ontology cluster, comprising a meta-ontology, a meta-ontology instance, an initial ontology, and enriched initial ontologies.

The ontology cluster is examined for its comprehensibility and its suitability for the domain, thus obtaining answers to several questions of competence.

The model, the conceptual schema and the detailed descriptions of the conceptualization aspects obtained through this methodology permit several questions of competence to be answered:

- what are the essential concepts in the given problem domain?
- what are the relevant sub-concepts of these concepts?
- which aspects of conceptualization must be examined in more detail?
- what kind of functionality is inherent (or is desired) in the specific aspect?
- which problem domains have mutual similarity?

ONTO6 methodology has been applied to several domains [8], including the Latvian Education Informatization System (LIIS). In the following we briefly describe the stages of ONTO6 methodology for the LIIS domain.

- first ONTO6 stage finds the significant aspect space (meta-ontology instance) for the particular domain. As the main input document for the meta-ontology instance creation the National “Informatics” program document was used. The document contained 9753 words, it was processed stripping stop-words and rare words. To the remaining 2985 words the threshold parameter was applied, leaving 28 significant word. Afterwords the matches between the significant words and the corresponding terms in the meta-ontology were calculated. As the result the meta-ontology instance containing only the aspects (Where, What) was created.
- Second ONTO6 stage develops an initial ontology. The attributes of the top-level aspects were analyzed – the corresponding instances significance was evaluated. As the result the most significant terms corresponding to the aspects were found – Where(Public, School, School Board, Ministry), What(Education Content, Training, Management, Infrastructure, InfoService).
- Third ONTO6 stage creates enriched initial ontologies. The initial ontology was refined several times during the implementation of the LIIS to reflect more concrete aspects. For example, the InfoService part was refined adding such terms as User, Resource, Developer, Constructor.

1.3. Proposed Process of Semi-automatic Generation of Lightweight Ontology

Similarly to the methodology described in [13] our methodology also is based on popular basic methodology described by Noy and McGuinness [14]. We take into account the important principles stated in [14]:

Principle 1: "*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*";

Principle 2: "*Ontology development is necessarily an iterative process*";

Principle 3: "*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*".

Our main goal is to create a complementary material for glossary that allows developers of glossary to improve glossary and that helps glossary users to easier understand the domain of glossary (Principle 1).

We use a semi-automatic process to get corrections from experts and take these corrections into account during the next refinement iterations (Principle 2). In this paper we describe only the first iteration that concentrates on getting the essence of given domain. After the experts have approved this backbone of ontology, the refinement iterations start.

We choose one of the best documents for the chosen field that describes actual concepts and relations between them (Principle 3).

Performing the first iteration we obtain a *lightweight ontology* according to explanation given in [9]: „*The ontology community distinguishes ontologies that are mainly taxonomies from ontologies that model the domain in a deeper way and provide more restrictions on domain semantics. The community calls them lightweight and heavyweight ontologies respectively. On the one hand, lightweight ontologies include concepts, concept taxonomies, relationships between concepts and properties that describe concepts. On the other hand, heavyweight ontologies add axioms and constraints to lightweight ontologies*".

Authors of paper [14] advice seven steps that might be done obtaining initial ontology. In this paper we show only activities that relate to the first four steps:

1. *Determine the domain and scope of the ontology*. We chose the software testing domain and want to create lightweight ontology that express the essence of software testing. The aim is to create a semi-automatic process that allows to maintain the ontology later.
2. *Consider reusing existing ontologies*. After analysis we decided do not use any existing ontology at first iteration of our approach. They can be exploited at next iteration while refining the ontology. We chose the glossary of the domain that is appropriate for our task – it is a semi-structured document and contains features of *dictionary* and *thesaurus*.
3. *Enumerate important terms in the ontology*. We use ideas from ONTO6 methodology to find the most important aspects of domain. Improved algorithm is used to detect most important words that characterize the main aspects of software testing.
4. *Define the classes and the class hierarchy*. Ontology is created as a set of related aspects. Each aspect can be viewed as small ontology. For each chosen aspect we apply a new original algorithm that automatically defines classes and relations between them.

Afterwards a description of obtained lightweight ontology has been generated using OWL RDF/XML notation. Then the description has been imported into the ontology editor Protégé [15]. To visualize, analyse, and edit the ontology we chose OWLGrEd [16] [17] that provides interoperability with Protégé.

At next stages for refining ontology we plan to accumulate corrections made by experts and regenerate ontology. This refinement process can be performed several times implementing the next steps mentioned in [14]:

5. *Define the properties of classes-slots*;
6. *Define the facets of the slots*;
7. *Create instances*.

2. Aspects of Software Testing

In the following chapter the aspects of software testing will be considered.

2.1. Glossary as Source Document

Glossary usually is a good document that represents the most important things of the domain for which the glossary is developed. Glossary is a semi-structured document that contains domain concepts and its explanations. Some relations between concepts are defined in an explicit way.

According to the authors of this paper the best glossary of the software testing is Version 2.1 of "*Standard glossary of terms used in Software Testing*" issued on April 1st, 2010 [18]. This glossary is produced by the "*Glossary Working Party*" of International Software Testing Qualifications Board (ISTQB) [19]. Glossary accumulates terms and explanations from the most significant sources in the software testing field. Contribution was made from testing communities throughout the world. This glossary contains 724 entries. For comparison, the glossary "*IEEE Standard Glossary of Software Engineering Terminology*" [20] contains approximately 1300 entries to describe terms in the whole field of software engineering.

Let us have a closer look at the chosen glossary. It consists of *entries*. Each entry has two parts – a *term* and a *definition*. The term is the concept what is explained (e.g. 'black box testing' in the 1-st entry in Figure 1 and the definition is the text that follows after term and colon symbol ':' (e.g. 'Testing, either ... or system.' in Figure 1).

The "See" indicator is used in the definition to say that term is synonym of other term (e.g. 'See *black box testing*.' in the 2-nd entry in Figure 1).

"See also" cross-references are used as a part of definition. "See also" cross-references are constructed for relationships such as broader term to a narrower term, and overlapping meaning between two terms (e.g. 'See also *black box testing*.' in the 3-rd entry in Figure 1).

Some other concepts, things, relations can be found in terms and definitions, for instance, acronym or reference to the source document of entry (e.g. '(CCB)' and '[IEEE 610]' in the 4-th entry in Figure 1).

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]

Figure 1. Structure of the glossary.

2.2. Finding of significant aspects

ONTO6 methodology suggests to find the most significant aspects by analyzing the occurrences of terms in input document and setting the threshold for the inclusion in the list of significant aspects.

Let us see the sample entries in Figure 1. We can observe that:

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

Contrary to the analysis of arbitrary text, it is more easy to detect the essence of domain using glossaries due to the principles of glossary construction.

We introduce a new method for extracting the most significant words from document in the form of a glossary. To each word from the glossary we assign a *weight*. The total weight of the word is the sum of the word weights in each entry.

The closer a word is to the border between term and definition, the weightier it is. Let us number all words in a term from the right to the left starting with 0, and number all words in definition from the left to the right starting with 0. Let us call the word number as *word_index*. The weight of a word is the sum of all weights of all instances of this word in entry, and weight of the word instance is calculated by the formula

$$2^{\text{-word\_index}}.$$

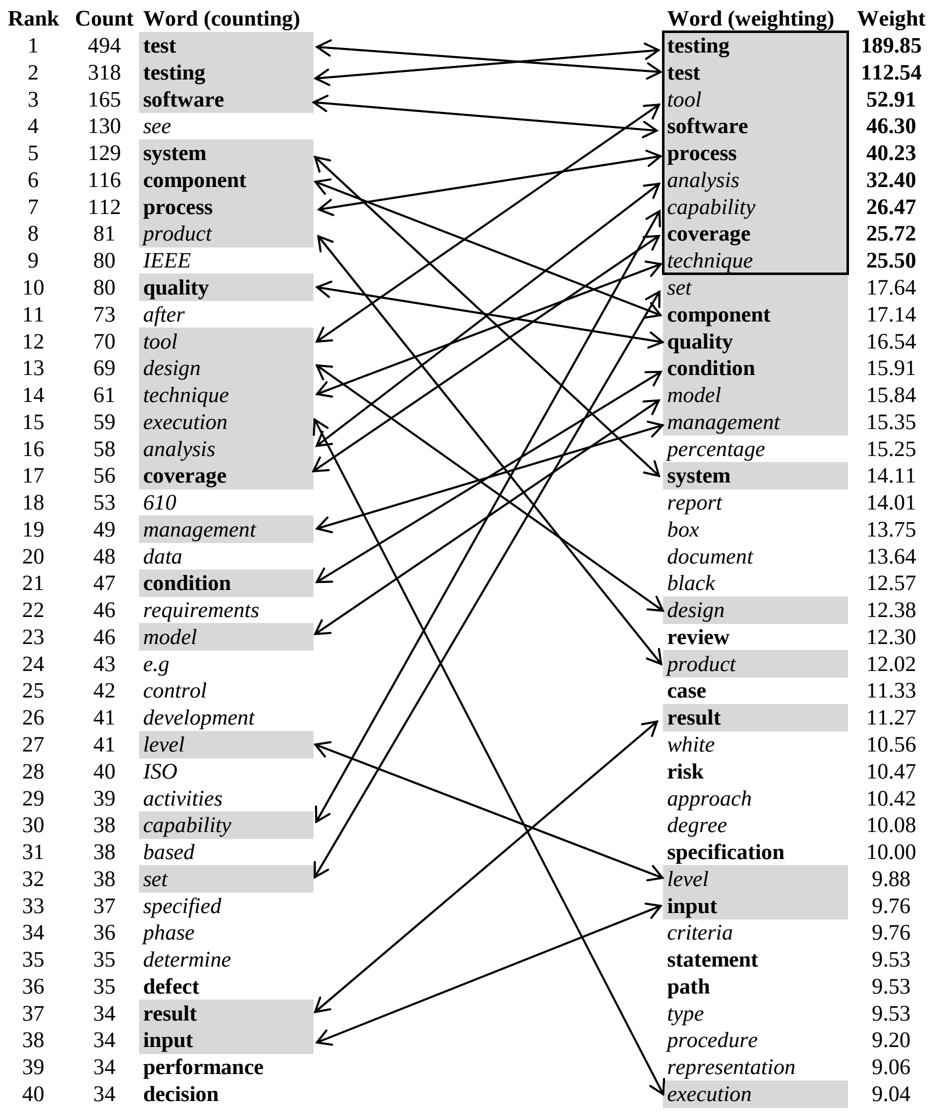
Before weight calculations the following entry transformations are made:

1. Extract and delete acronyms from term;
2. Delete all punctuation marks;
3. Replace uppercase letters with lowercase ones if the word is not an acronym.
4. Delete all *stop words* (we use 1200 stop word list compiled from public sources).

For instance, entry "**functional testing:** Testing based on an analysis of the specification of the functionality of a component or system. See also *black box testing*." Weight calculation for this entry results in "functional (1, 2^{-1}) testing (1, 2^{0}) : testing (1, 2^{0}) based (1, 2^{-1}) analysis (1, 2^{-2}) specification (1, 2^{-3}) functionality (1, 2^{-4}) component (1, 2^{-5}) system (1, 2^{-6}) see (1, 2^{-7}) also (1, 2^{-8}) black (1, 2^{-9}) box (1, 2^{-10}) testing (1, 2^{-11})", where the word index and weight are given in brackets. So the weight for word "testing" in given entry is sum $2^{0} + 2^{0} + 2^{-11} = 2.00048828125$.

The result of words weighting process is shown at Figure 2. On the left side we include the result of word counting that is used in the original ONTO6 methodology. For both methods we show the top 40 words. Word, which is in both categories, has a grey background. The word in bold is a term of the given glossary, and the word in italic is not a term.

ONTO6 suggests to define a threshold and use a small amount of significant words. After analysing weight curve we decided to take the first 9 weightiest words: *testing, test, tool, software, process, analysis, capability, coverage, technique*.



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

Figure 2. Word weighting process result.

3. Creation of the aspect graphs

In this section we offer ideas and algorithms that allow the creating of a lightweight ontology automatically concluding with the generated description in OWL RDF/XML notation that is ready to be imported into Protégé.

Graph is constructed by nodes that in simplified form have the structure:

```
Structure Node
   string term
   string definition
   set children
   set synonyms
```

At the highest level the algorithm is given in Figure 3. At first we find all entries that belong to given aspect. Then a graph is created, any two nodes are connected by edge if a relation between corresponding terms is discovered. Afterwards graph is simplified by reducing nodes (merging of nodes that correspond to synonym terms) and by reducing edges (deleting excessive relations assuming that all relations at this iteration are transitive). And the last step is the generation of the ontology description expressed by OWL notation.

```
createAspectOntology(set glossary, string aspect)
  aspectEntrySet = createEntrySet(glossary, aspect)
  aspectGraph = createAspectGraph(aspect, aspectEntrySet)
  mergeSynonyms(graph aspectGraph)
  reduceRelations(graph aspectGraph)
  aspectOwlDesription = generateOwlDescription(graph aspectGraph)
end createAspectOntology
```

Figure 3. Creation of the aspect ontology.

3.1. Conditions

Let us introduce six conditions that are necessary for the following algorithms. We can consider them as functions:

- **cond_term_1**(<u>string</u> *term*, <u>string</u> *pattern*): <u>bool</u> – checks whether the word *pattern* is among words in the *term*;
- **cond_term_2**(<u>string</u> *term*, <u>string</u> *pattern*): <u>bool</u> – checks whether the sequence of words *pattern* is at the very beginning of the sequence of words *term*;
- **cond_term_3**(<u>string</u> *term*, <u>string</u> *pattern*): <u>bool</u> – checks whether the sequence of words *pattern* is at the very end of the sequence of words *term*;
- **cond_def_1**(<u>string</u> *definition*, <u>string</u> *pattern*, <u>int</u> *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**(<u>string</u> *definition*, <u>string</u> *ref_pattern*, <u>string</u> *pattern*): bool – checks whether the sequence of words *pattern* is at the beginning of the sequence of words *definition*, and correspond to pattern *ref_pattern* (for instance *ref_pattern* = “see <word_list>”);
- **cond_def_3**(<u>string</u> *definition*, <u>string</u> *ref_pattern*, <u>string</u> *pattern*): bool – checks whether the sequence of words *pattern* is at the end part of the sequence of words *definition*, and correspond to pattern *ref_pattern* (for instance *ref_pattern* = “see also <word_list>”).

Value of *ref_pattern* might be regular expression that is used in the implementation of conditions.

3.2. Finding of Vertices

Before we can start build up a graph, we find all entries that correspond to the selected aspect (to the one of the top weightiest words, see Figure 2). For finding these entries we use the algorithm given in Figure 4.

```
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
```

Figure 4. Creation of entry set.

Result of the algorithm varies on parameter N. The value N=1 gives very good results – algorithm selects large amount of entries and there are small amount of inappropriate entries. We use N=4 that gives a larger set including supplemental significant entries, but there is a minor risk to select an inappropriate entry.

3.3. Finding of Edges and Creating of Graph

The first and main node (*aspect node*) in the graph is the node that corresponds to the given aspect. For each entry from selected entry set we create a node and join it as a child node of the aspect node (Figure 5).

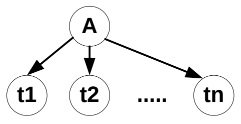


Figure 5. Node creation.

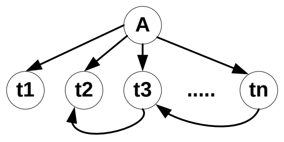


Figure 6. Edge creation.

Afterwards we find and create edges between all nodes which are not the aspect node (Figure 6). The relation between nodes is defined if any of used conditions returns true value. Both activities are detailed by algorithm in Figure 7. After this step we have got a pseudo-hierarchical graph which consists of the aspect node at the first level and its children at the second level. Many relations may exist between second level nodes.

```
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_2 <> 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
```

Figure 7. Creation of an aspect graph.

3.4. Graph Transformation

The next two steps of our approach are transformation of the pseudo-hierarchical graph to hierarchical graph by reducing nodes (merging of synonyms) and edges (deleting excessive transitive relations).

Synonyms can be reduced by algorithm given in Figure 8. See idea of algorithm in Figure 10.

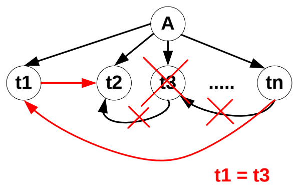
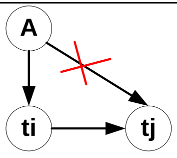
```
mergeSynonyms(graph aspectGraph)
  for each node_1 of aspectGraph
    for each node_2 of aspectGraph
      if node_1 <> node_2 and
         node_1 <> aspectNode and
         node_2 <> aspectNode
        if cond_def_2(node_2.definition, SEE_REF_PATTERN,
                                                   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
```

Figure 8. Merging of synonyms.

Excessive edges are detected by algorithm given in Figure 9. We recognize all situations like shown at Figure 11 and delete the edge.

```
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
```

Figure 9. Reducing of relations.

 **Figure 10.** Reducing of synonyms.	 **Figure 11.** Detection of excessive edges.

When the graph is transformed we generate the ontology description. This is a simple technical work, and we do not describe any details in this paper.

4. Results

Our developed generator can generate ontology aspect from any given word. Following the ONTO6 methodology we generate the ontology aspects only for nine weightiest words: *testing*, *test*, *tool*, *software*, *process*, *analysis*, *capability*, *technique*. Obtained ontologies were visualized by OWLGrEd to evaluate results. Sample of ontology aspect *technique* is given in Figure 12. Software testing expert has analysed the context of the aspect *technique* and has deduced its correspondence to the top level aspect WHAT according to the ONTO6 methodology.

After analyzing all nine ontology aspects we establish a fact that together these aspects include 629 unique entries from all 724 entries of chosen glossary. That means that nine words cluster and create a hierarchy for 87% terms of the whole glossary. We have created ontology aspects for all the weightiest 40 words (Figure 2). They add a small amount of additional terms, nonetheless we have got a significant amount of new nodes with non-term words that give new insight to the glossary.

By seeing the whole generated ontology in one diagram, we concluded that it is difficult to comprehend it. Relations between aspects (clusters) make diagram too complex to manage it. If we look at the ontology as a set of nine aspects then complexity is quite low (a principle of ONTO6 methodology).

5. Conclusion and future works

We have shown that it is possible to semi-automatically generate a lightweight ontology from a good quality glossary. Support of experts at first iteration is minimal.

We offer an original method to discover the important aspects of domain that is represented by a glossary. These aspects are weightiest words that are found by rather simple method. According to the ONTO6 methodology we construct an ontology for each important aspect. We carried out various experiments designing and investigating algorithms that can automatically create a ontology graph. The most simple and powerful principles, that we have found, are expressed by algorithms that are presented in this paper.

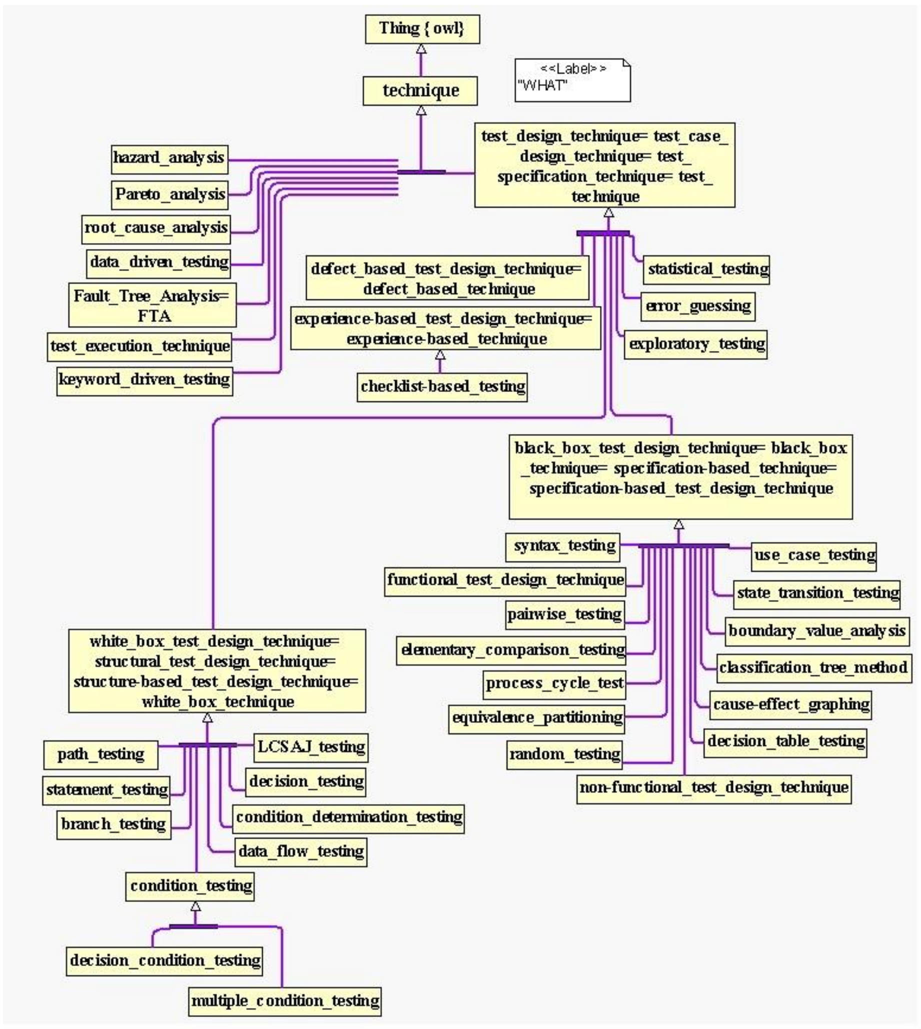

Thing { owl}
technique
<<Label>>
"WHAT"
test_design_technique= test_case_design_technique= test_specification_technique= test_technique
hazard_analysis
Pareto_analysis
root_cause_analysis
data_driven_testing
Fault_Tree_Analysis= FTA
test_execution_technique
keyword_driven_testing
defect_based_test_design_technique= defect_based_technique
experience-based_test_design_technique= experience-based_technique
checklist-based_testing
statistical_testing
error_guessing
exploratory_testing
black_box_test_design_technique= black_box_technique= specification-based_technique= specification-based_test_design_technique
syntax_testing
functional_test_design_technique
pairwise_testing
elementary_comparison_testing
process_cycle_test
equivalence_partitioning
random_testing
non-functional_test_design_technique
use_case_testing
state_transition_testing
boundary_value_analysis
classification_tree_method
cause-effect_graphing
decision_table_testing
white_box_test_design_technique= structural_test_design_technique= structure-based_test_design_technique= white_box_technique
path_testing
statement_testing
branch_testing
LCSAJ_testing
decision_testing
condition_determination_testing
data_flow_testing
condition_testing
decision_condition_testing
multiple_condition_testing

Figure 12. Technique ontology.

Our approach prescribes at first iteration to make ontology as simple as possible. We do not fix attributes for concepts and relationships. It will be done at next iterations while refining ontology up to the needed details.

Regardless of our simplification of ontology, the results already are useful to better understand the field represented by the given glossary. Created aspects of ontology might be useful to developers of glossary to evaluate the quality of their glossary. Complimentary material to the paper, containing graphical representation of relations between aspects, etc., is available on our expanding site [21].

The next research activities include: formal fixing of improvements made by experts with OWLGrEd; developing methodology for next iterations (automatic finding of attributes, new compound concepts and relationships); automatic taking into account of experts refinements; following up the changes made in the glossary; generation of ontologies from other glossaries in the same field and integrating them.

Acknowledgement

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

References

[1] Zhao Yajing, Dong Jing, Peng Tu, *Ontology Classification for Semantic-Web-Based Software Engineering,* IEEE Transactions on Services Computing, v.2 n.4(2009), 303-317.
[2] Zhu H., Huo Q., *Developing A Software Testing Ontology in UML for A Software Growth Environment of Web-Based Applications,* in: Software Evolution with UML and XML, IDEA Group Inc., Ed.(2005), 263-295.
[3] Huo Q., Zhu H. & Greenwood S., *A Multi-Agent Software Environment for Testing Web-based Applications,* in: Proceedings of the 27th IEEE Annual Computer Software and Applications Conference (COMPSAC'03), Dallas, Taxas, USA(2003), 210-215.
[4] Barbosa E. F., Nakagawa E. Y., Maldonado J. C., *Towards the establishment of an ontology of software testing,* in: 18th Int. Conf. on Soft. Engineering and Knowledge Engineering (SEKE'06)(2006), 522-525.
[5] Nakagawa E. Y., Simão A. S., Ferrari F. C., Maldonado J. C., Towards a reference architecture for software testing tools, in: SEKE'07(2007), 157-162.
[6] Mendes O., Abran A., *Software Engineering Ontology: A Development Methodology,* Metrics News 9 (2004), 68–76.
[7] IEEE, *Guide to the Software Engineering Body of Knowledge (SWEBOK),* IEEE Computer Society, Los Alamitos, Calif, 2004, Available: http://www.swebok.org
[8] Straujums Uldis, *Conceptualising Informatization with the ONTO6 Methodology,* in: Acta Universitatis Latviensis. Volume 733. Computer Science and Information Technologies. University of Latvia(2008), 241-260. ISBN 987-9984-825-47-0
[9] Gómez-Pérez,Asunción, Fernández-López,Mariano, Corcho,Oscar. Ontological Engineering: with examples from the areas ofknowledge management,e-commerce and the semantic web. Springer-Verlag, 2004, 411p. ISBN 1-85233-551-3.
[10] Bozzato Loris, Ferrari Mauro, Trombetta Alberto, *Building a domain ontology from glossaries: A general methodology,* Integration The Vlsi Journal, 426(2008), 1-10.
[11] Bedini Ivan, Nguyen Benjamin, *Automatic ontology generation: State of the art,* in: PRiSM Laboratory Technical Report. University of Versailles, 2007.
[12] Miiller Adrian, Dorre Jochen, *The TaxGen Framework: Automating the Generation of a Taxonomy for a Large Document Collection,* in: Proceedings of the Thirty-Second Annual Hawaii International Conference on System Sciences-Volume 2(1999), 9p.
[13] 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.
[14] Noy N. F., McGuinness D. L., *Ontology Development 101: A Guide to Creating Your First Ontology*,2001, Available: http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html
[15] Protégé. An ontology editor, Available: http://protege.stanford.edu
[16] Bārzdiņš Jānis, Bārzdiņš Guntis, Čerāns Kārlis, Liepiņš Renārs, Sproģis Artūrs, *UML Style Graphical Notation and Editor for OWL 2* , in: Perspectives in Business Informatics Research, Lecture Notes in Business Information Processing, Volume 64, Part 2(2010), 102-114.
[17] *OWLGrEd, Editor for Compact UML-style OWL Graphic Notation,* Available: http://owlgred.lumii.lv/
[18] International Software Testing Qualifications Board, *Standard glossary of terms used in Software Testing. Version 2.1,* 2010.
[19] ISTQB, *International Software Testing Qualifications Board,* Available: http://istqb.org
[20] Institute of Electrical and Electronics Engineers (IEEE), *IEEE Std 610.12-1990(R2002): IEEE Standard Glossary of Software Engineering Terminology (Reaffirmed 2002),* 84p.
[21] Arnicans, Guntis, Romans, Dainis, Straujums, Uldis. *Complementary material to the paper: Semi-automatic Generation of a Software Testing Lightweight Ontology from a Glossary Based on the ONTO6 Methodology* [online]. Available: http://science.df.lu.lv/ars12/