Production Systems and Information Engineering
Volume 11 (3), pp. 128–141.
https://doi.org/10.32968/psaie.2023.3.9

ANALYSING OF LIBRARY, LITERATURE AND SEM ONTOLOGIES

ANITA AGÁRDI
University of Miskolc
Hungary Institute of Information Technology
agardianita@iit.uni-miskolc.hu

Abstract: In this article, I present an analysis of three ontologies. These three ontologies are the followings: Library (representing a library), Literature (representing an ontology of literature), and Sem (representing software annotation). These ontologies have been downloaded from Github, all three ontologies created in OWL. The article also contains measures adapted from UML metrics and their evaluation. The article first presents some in connection with the UML and ontology metrics. Next, the article presents the ontologies, followed by the evaluation of the metrics. The article shows that the metrics are suitable for the analysis of ontologies.

Keywords: *library ontology, literature ontology, sem ontology, metrics*

1. Introduction

The demand for software and software codes is playing an increasingly important role these days. Software developers need metrics to objectively measure individual implemented functions and the entire software. [1] Today, many such metrics have been developed for UML (Unified Modeling Language), such as the following [1]:

- WMC (Weighted Methods per Class),
- DIT (Depth of Inheritance),
- NOC (Number Of Childrens),
- DAC,
- DAC’,
- NOM,
- SIZE2,
- MHF (Method Hiding Factor),
- AHF (Attribute Hiding Factor),
- MIF (Method Inheritance Factor),
- AIF (Attribute Inheritance Factor)

The use of metrics in the software development phase, especially in the early phase, greatly contributes to the quality of the developed software. [2]

Baroni & Abreu [3] discuss object-oriented design metrics. Also Chen, Boehm, Madachy, & Valerdi. [4] investigated the importace of these types of metrics. The authors of [4] discuss the following metrics:

System and Software Architecture Description (SSAD):

- Number of decomposed use cases
- Number of steps in sequence diagram
- Number of classes
- Average number of methods per class
- Average number of attributes per class

Source Code Metrics

- Requirement Metrics:
- Number of project requirements
- Number of capability requirements
- Number of interface requirements
- Number of level of service requirements

According to the authors Lavazza & Agostini [5], UML metrics play an increasingly important role in software development. According to them, it is crucial to be able to derive accurate quantitative knowledge from software products. Metrics should be measured especially in the early phase of software development, because with these metrics, managers also have more information for decision-making. They discussed the following object-oriented metrics: Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number Of Children (NOC), Coupling Between Object classes (CBO), Response For a Class (RFC), Lack of Cohesion in Methods (LCOM).

According to the authors Genero, Miranda, & Piattini [6], software maintainability is an increasingly important quality aspect. The authors also draw attention to the early phase of software development, when the most important thing is to be able to evaluate the quality characteristics of the software. The authors examine the relationship between early maintainability and complexity of UML state diagrams. The following metrics were defined in their article:

- NUMBER OF ENTRY ACTIONS (NEntryA): The total number of entry operations, i.e., each operation of the status.
- NUMBER OF EXIT ACTIONS (NExitA): The total number of exit operations, that is, the number of operations performed when leaving the state.
- NUMBER OF ACTIVITIES (NA): The total number of activities in the state chart.
- NUMBER OF STATES (NS): Total number of simple states, including simple states within complex states.
- NUMBER OF TRANSITIONS (NT): Total number of transitions.

Fourati, Bouassida, & Abdallah [7] investigated with anti-patterns in UML. Anti-patterns are bad design patterns, they greatly impair the progress of software development. It is worth avoiding these patterns and refactoring the code in such a way. Design patterns, on the other hand, provide good solutions to common problems. To use them, software developers must have thorough knowledge. The authors discuss the following patterns:

Coupling:

- CBO (Coupling Between Objects)
- RFC (Response For Call)

Cohesion:

- LCOM (Lack Of Cohesion in Methods)
- TCC (Tight Class Cohesion)
- LCC (Loose class Cohesion)
- Coh

Complexity:

- WMC (Weighted Methods per Class)
- NAtt: the Number of the Attributes
- NPrAtt: the Number of Private Attributes
- NOM: the Number Of Methods
- NII: the Number of Imported Interfaces

Inheritance:

- DIT (Depth of Inheritance of a class)
- NOC (Number Of Children)
- NAcc: the Number of the Accessors
- NAss: the Number of Associations
- NInvoc: the Number of the Invoked methods
- NReceive: the Number of the Received messages

The article is structured as follows. After this literature research, I present the ontology and the three selected ontological systems, followed by the analyzes of the metrics.

2. Ontology systems

2.1. Ontology

An ontology is a tool for standard knowledge representation. An ontology describes concepts and relationships between them. But ontology is actually more than that, because it also contains an inference engine, so it also contains knowledge that is not explicitly described. One of the tools of the semantic web is OWL [8] (Web

Ontology Language). Describe the individual ontologies in OWL XML style. It contains the following important elements [9]:

- Namespaces
- Ontology Headers
- Classes and Individuals: the most important elements of ontologies. An individual represents an instance of the class. Classes can be arranged in a class-subclass hierarchy.
- Properties:
  - ObjectProperty: connects two classes
  - DatatypeProperty: connects a class with a datatype
  - Annotation property
- Property characteristics :
  - Transitive property: P(x,y) and P(y,x) implies P(x,z)
  - Symmetry property: P(x,y) iff P(y,x)
  - FunctionalProperty: P(x,y) and P(x,z) implies y = z
  - inverseOf: P1(x,y) iff P2(y,x)
  - InverseFunctionalProperty: P(y,x) and P(z,x) implies y = z
- Property restrictions:
  - allValuesFrom, someValuesFrom
  - Cardinality
  - hasValue
- Equivalence between Classes and Properties:
  - equivalentClass, equivalentProperty
- Identity between Individuals:
  - sameAs
- Different Individuals
  - differentFrom, AllDifferent

In the followings, the structures of the ontologies and their VOWL diagrams are presented. The conversion of ontologies to UML will be also presented in this section, and in the next chapter, the values of UML metrics will also be detailed.

2.2. Library ontology

The library ontology [10] represents a library. It contains classes such as ‘LibraryMember’, which describes a person that can be connected to a library. This class has three subclasses, ‘AdminStaff’ (which represents the administrators), ‘GuestUser’ (which describes the guest users), ‘Student’ (the class which represents the students) The ‘LibraryPersonnel’ class represents the librarians, which has two subclasses: ‘Librarian’ and ‘Technician’. ‘LibraryResource’ contains the contents of the library, which can be borrowed and viewed. It contains the following three subclasses: ‘Book’, ‘CD’ and ‘Journal’. The library also provides services, represented by the ‘LibraryService’ class. It provides three services, which are the followings:

‘InternetAndWiFiService’, ‘NewsPaperService’ and ‘ReferenceService’. The system contains only a few object properties, these are: ‘hasName’, ‘isUsedBy’, ‘Use’ and ‘Utilize’. The ‘hasName’ is a property of a ‘Student’ class, ‘isUsedBy’ binds a ‘LibraryMember’ and an ‘InternetAndWifiService’ class, ‘Use’ binds the ‘Library Member’ and ‘InternetAndWifiService’ classes, and ‘Utilize’ connects the ‘LibraryMember’ and ‘LibraryResource’ classes. The system does not contain a datatype property, but it contains the following individuals: ‘AMaheshwari’, ‘Chip’, ‘Digit’, ‘ElectronicsForU’, ‘EmbeddedSystemDesign’, ‘IndianJournalOfLabourEconomics’, ‘IUPJournalOfMarketing’, ‘KSathish’, ‘MMadhu’, ‘MMalathi’, ‘PCQuest’, ‘Prof.ShivaramaKrishna’, ‘Prof.VenkatapathiRaju’, ‘Raju’, ‘RRajesh’ and ‘Rushi’. These includes the followings: ‘AdminStaff’, ‘CD’, ‘Journal’, ‘Librarian’, ‘GuestUser’ and ‘Technician’. The VOWL representation of the system is illustrated in Figure 1.

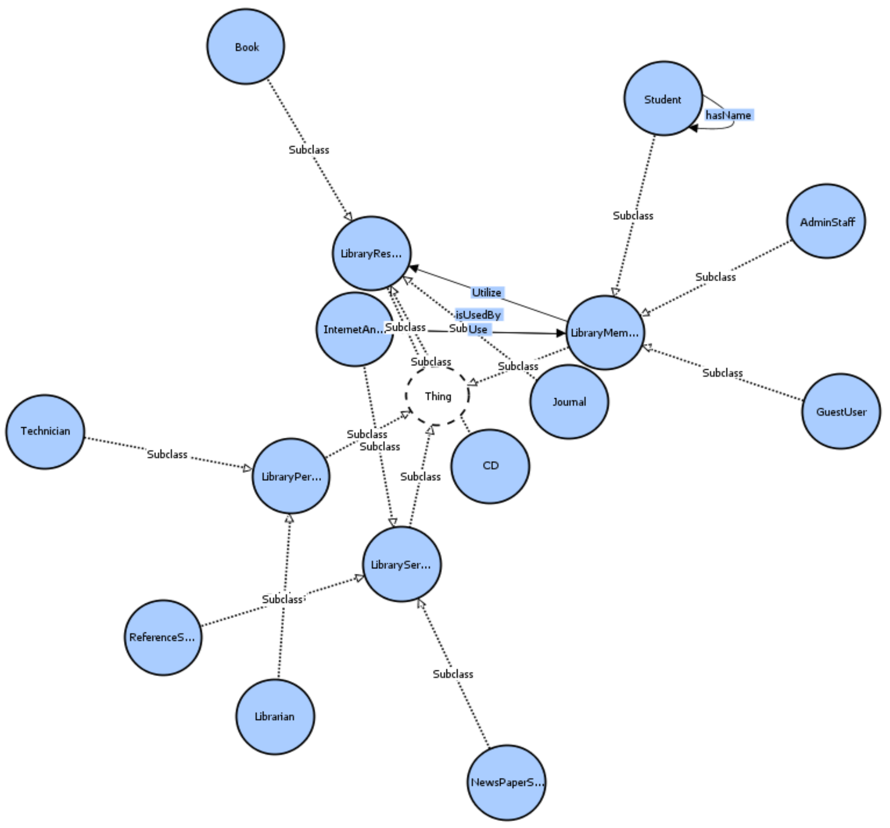

Figure 1. *Library ontology VOWL representation*

During the UML conversion of the Library ontology, the OWL classes will be UML classes. Class-subclass relationships are remained in UML. The figure shows that most ontological classes had no properties, and the classes that have properties in

OWL remained properties in UML as well. UML class diagrams could also be supplemented with methods, but based on OWL, this was not created due to the structure of the ontology.

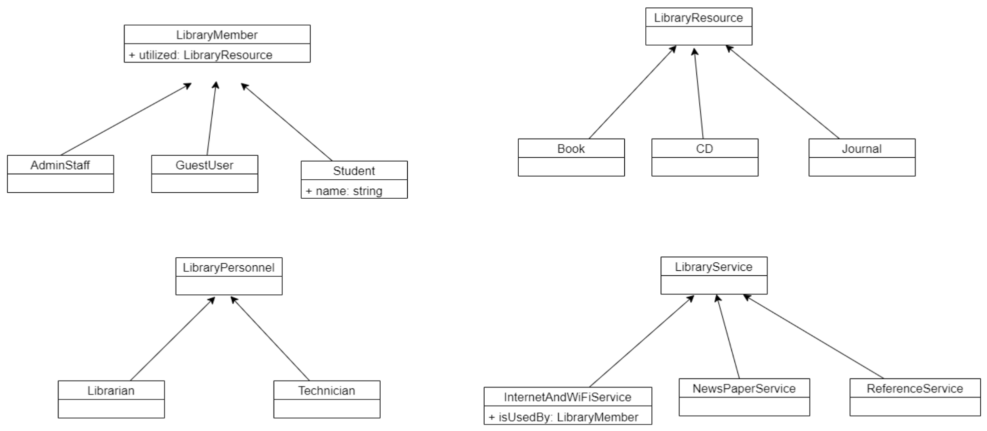

Figure 2. *Library ontology UML representation*

2.3. Literature ontology

The Literature ontology [11] is an ontology describing literature. This ontology contains only a few classes, which are arranged in a hierarchy. A subclass of 'Author' is 'Book'. The child of 'Book' is 'Fiction', which is further specified by the authors, with two subclasses, which are as follows: 'Murder_Mystery' and 'Science_Fiction'. This ontology does not contain properties (neither object properties nor datatype properties). But it contains individuals, but only the 'Author' class, which is the following: 'Agatha_Christie', 'Charles_Dickens', 'Dick_Francis', 'Ernest_Hemingway', 'James_Agee', /James_Joyce', 'John_Grishom', 'John_Steinbeck', 'John_Updike', 'Ken_Follet', 'Mave_Binchey', 'PD_James', 'Scott_Turow' and 'William_Falkner'. The VOWL representation of these system is illustrated in Figure 2.

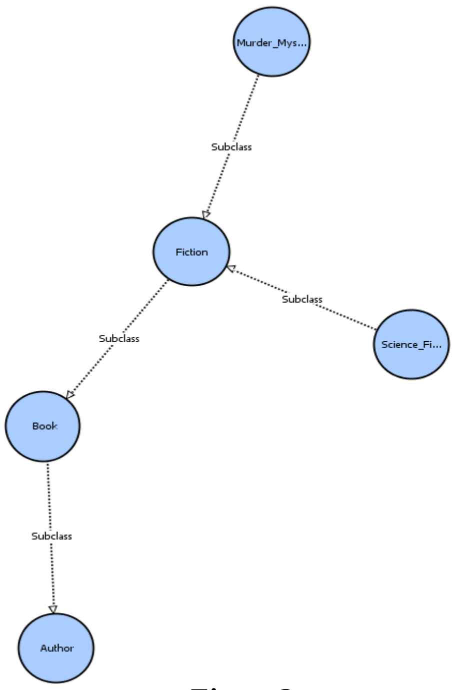

Figure 3
Literature ontology VOWL representation

The Literature ontology is a small ontology, so it is easy to convert less to UML. It only consists of classes and subclasses, so this hierarchy must also be represented in the UML.

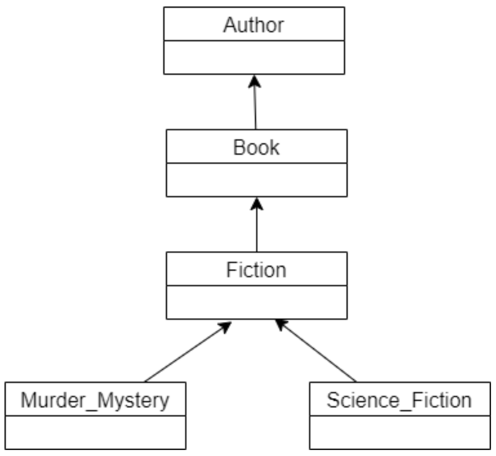

Figure 4. *Literature ontology UML representation*

2.4. Sem ontology

Ontology, which describes software annotations is called as the Sem ontology [12]. This ontology does not contain individuals or properties (neither datatype property nor object property). Its main class is ‘comment’, which represents software commenting. This class contains a single subclass, ‘I_have_written_it’, which indicates that the comment was made by the user. Here, the other subclasses of the ‘comment’ class could be expanded, which indicates who created the comment, because a larger software is not created by just one person, the software is created by working in a team. You could also specify which part of the software has the comment (which could even be specialized with subclasses: which class, backend or frontend for a web application, which function it is related to, which jira ticket, which software version, etc.) This class only contains the following subclasses: ‘autogenerated’ (which indicates that it is automatically generated by the IDE), ‘documention_exists’ (there is additional documentation), ‘documention_is_missing’ (there is no additional documentation), ‘i_do_not_remember_why_I_wrote_it’ (I don’t remember why this code was written), ‘i_do_not_understand_how_it_could_work’, ‘i_do_not_understand_ my_code’, ‘i_have_to_used_it_for_years’, ‘it_compiles’ code), ‘it_is_deprecated’, ‘it_was_a_one_shot’ and ‘just_a_proof_of_concept’. The VOWL representation of these system is illustrated in Figure 3.

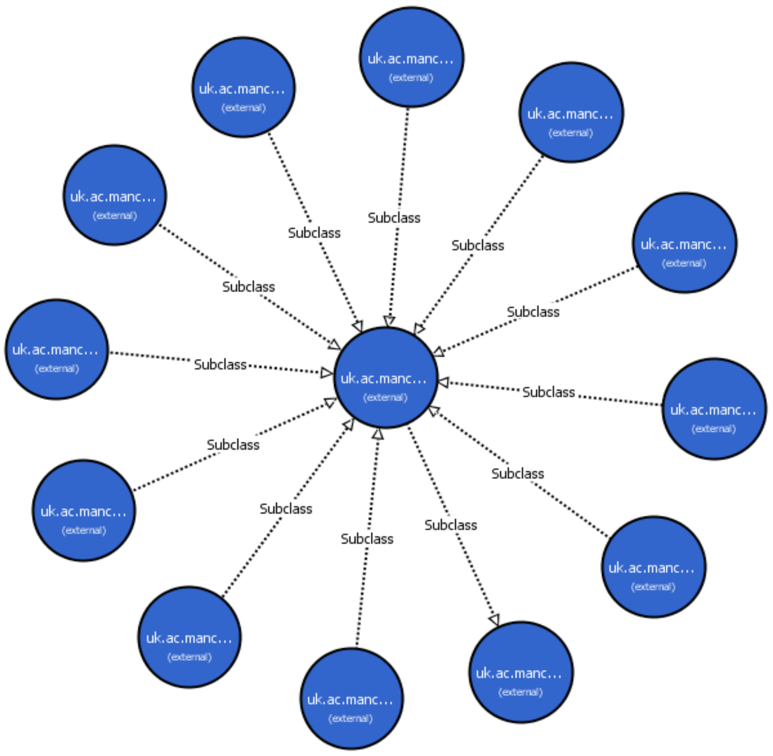

Figure 5. *Sem ontology VOWL representation*

I have represented the UML representation of this ontology in a slightly different way. I created some properties for ‘Comment’ even though OWL does not contain any properties. The elements in the enumeration were originally classes in OWL, but in UML they correspond to a ‘type’ property. This example clearly illustrates that it is not possible to convert ontologies to UML completely according to the rules, because if we followed the rules, the UML would contain many classes and the classes would be without properties. The human modeling approach is also necessary during conversion, it is not enough to know the conversion rules, there are cases when it is better if the conversion differs from them.

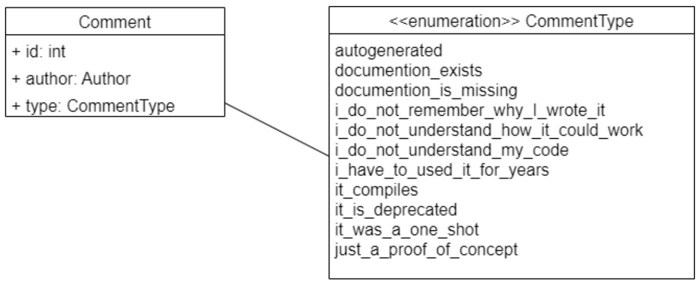

Figure 6. *Sem ontology UML representation*

3. Ontology systems evalutation

In this chapter, I present the evaluation of the three ontologies. I used the following evaluation metrics, which I adapted from the UML metrics, which I already presented in more detail in the following publication [13]:

- WMC (Weighted Methods per Class) and Average WMC
- DIT (Depth of Inheritance) and Average DIT
- NOC (Number Of Childrens) and Average NOC
- DAC and Average DAC
- OA1
- OA2

3.1. Library ontology

The WMC (Weighted Methods per Class) values of the Library ontology are between 0 and 3. Each class has this many properties. Many classes don't have a single property, but the 'LibraryMember' class has 3 properties. Average WMC (Weighted Methods per Class) is 0.466666667. The WMC diagram is illustrated in Figure 4.

The DIT (Depth of Inheritance) values are between 0 and 1, which means that the system has 2 levels. 4 classes with a value of 0, these classes are on the first level. Average DIT (Depth of Inheritance): 0.733333333. The DIT diagram is illustrated in Figure 5.

The average NOC is 0.733333333. The NOC diagram is illustrated in Figure 6. OA1 (total number of classes) is 15, while OA2 value (total number of inheritance hierarchies) is 2.

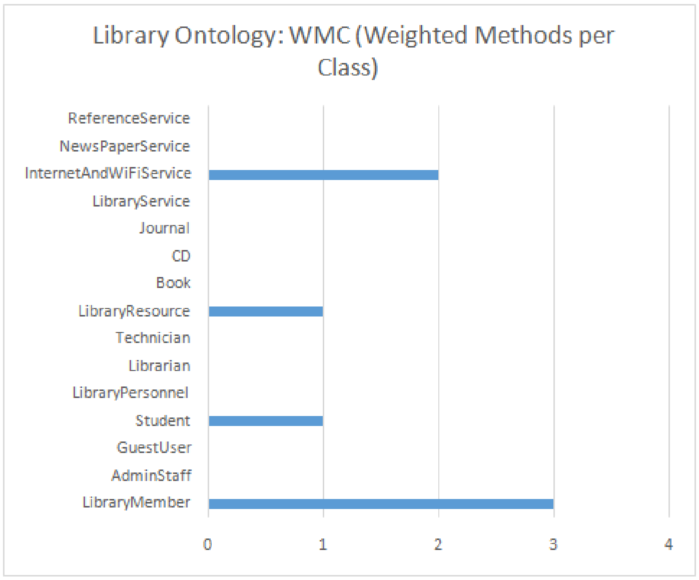

Figure 7. *Library Ontology: WMC (Weighted Methods per Class)*

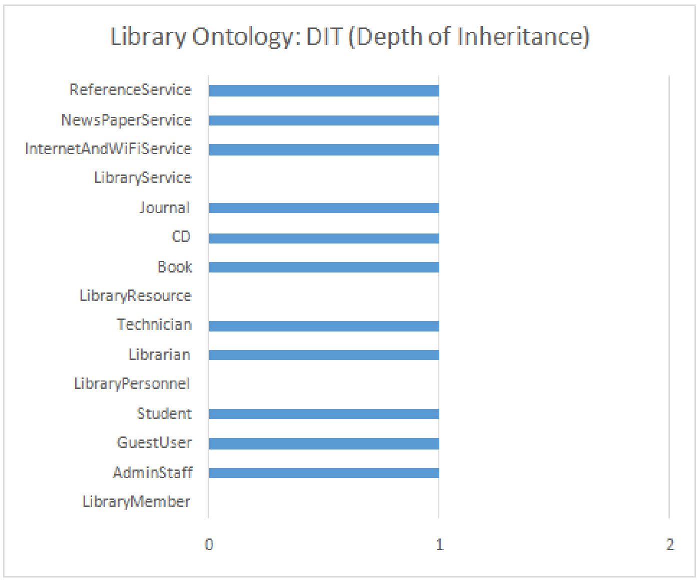

Figure 8. *Library Ontology: DIT (Depth of Inheritance)*

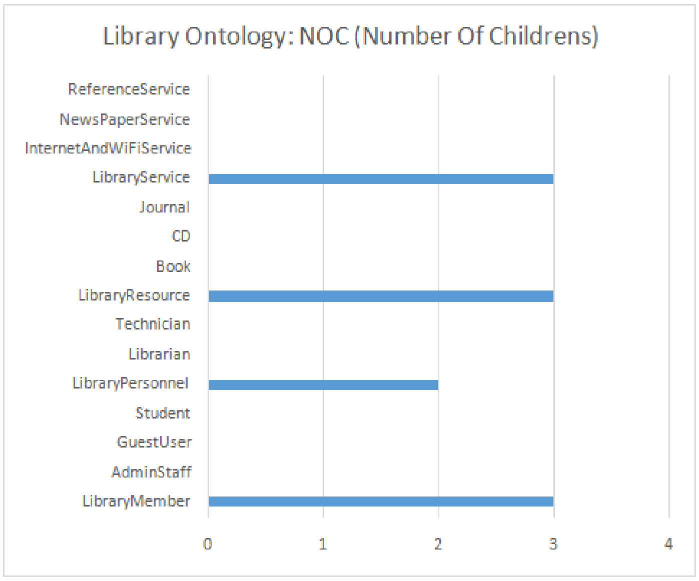

Figure 9. *Library Ontology: NOC (Number Of Childrens)*

3.2. Literature ontology

The Literature ontology has no properties, so the WMC (Weighted Methods per Class) and Average WMC (Weighted Methods per Class) values are 0.

Literature: DIT (Depth of Inheritance) values are between 0 and 3. This means that ‘Author’ is at the first level and there are two classes that are at the last level, Average DIT value (Depth of Inheritance) is 1.8. The DIT diagram is illustrated in Figure 7.

The NOC (Number Of Childrens) values are between 0 and 4. This is how many descendants each class has. Average NOC (Number Of Childrens) is 1.8, so a class has this many children on average. the NOC diagram is illustrated in Figure 7.

The OA1 value (total number of classes) is 5, the OA2 value (total number of inheritance hierarchies) is 3, so it is a three-tier system.

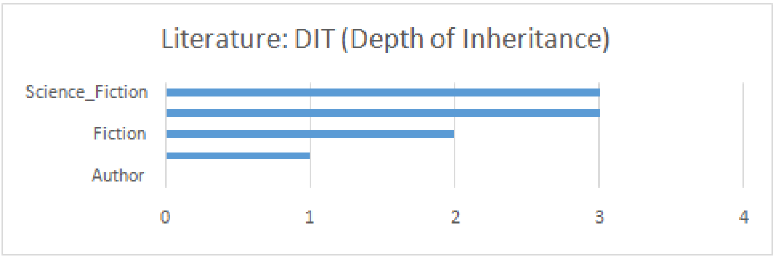

Figure 10. *Literature: DIT (Depth of Inheritance)*

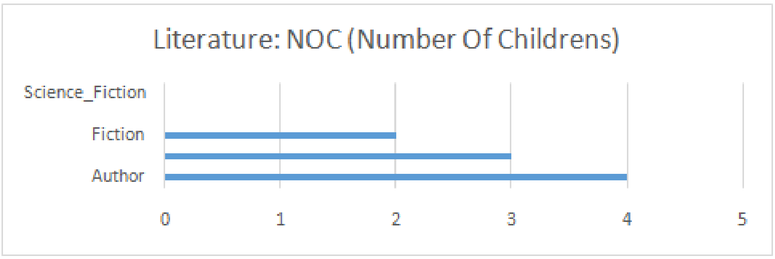

Figure 11. *Literature: NOC (Number Of Childrens)*

3.3. Sem ontology

The Sem ontology contains no properties, so the WMC (Weighted Methods per Class) and Average WMC (Weighted Methods per Class) values are 0.

Sem ontology DIT (Depth of Inheritance) values are between 0 and 2. A class (‘comment’) has a value of 0, it is the ancestor of all other classes. A single class (‘I_have_written_it’) has a value of 1 and is the ancestor of all other classes (except ‘comment’). Average DIT (Depth of Inheritance) value is 1.769230769. This means that, on average, the classes would be located between levels 1 and 2. The DIT diagram is illustrated in Figure 8.

The NOC (Number Of Childrens) values are between 0 and 12. These values also show that all other classes are subclasses of two classes. Average NOC (Number Of Childrens) is 1.769230769. The NOC diagram is illustrated in Figure 9. The OA1 (Total number of classes) is 13, while OA2 (Total number of inheritance hierarchies) is 3.

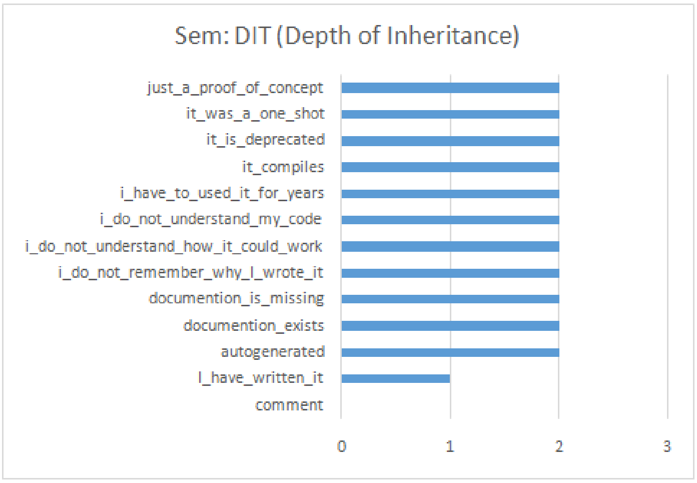

Figure 12. *Sem: DIT (Depth of Inheritance)*

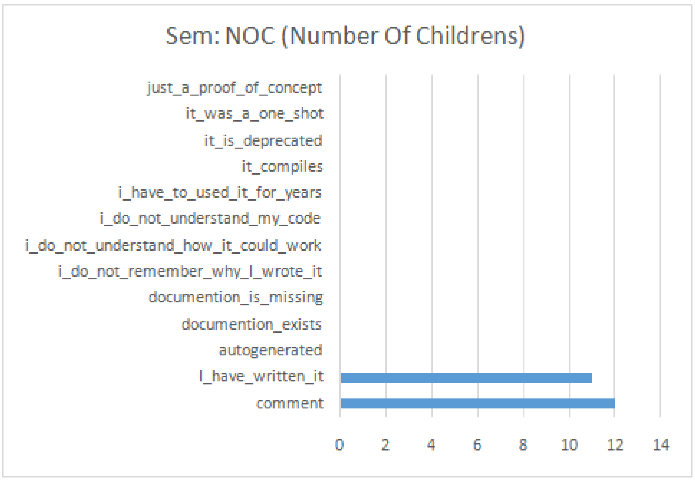

Figure 13. *Sem: NOC (Number Of Childrens)*

3.4. Summary results

The summary results are illustrated with a diagram in Figure 9. Based on the diagram, the Library ontology and the Sem ontology are larger ontologies than the Literature ontology. The value of the average WMC is not 0 only for the Library ontology. The average DIT and NOC values are not high for any of the ontologies, this means, that there is no class-subclass relationship, it is not important, there is no great specialization. However, the number of classes is large compared to the other metrics, as indicated by the OA1 values. The low WMC and DIT of the Library and Literature ontology suggests that the classes are simple and the hierarchy shallow and medium. A higher DIT and NOC in the Sem ontology indicates that the classes are more complex and the hierarchy is deeper. Based on the measures of the ontologies, the average WMC in all three cases is 0, which indicates that the classes do not have methods, that is, their structure is more descriptive. Based on the DIT and NOC values, the Library ontology is the simplest, the Literature ontology is moderately complex, while the Sem ontology has a more complex and hierarchical structure. Based on these observations, the Library ontology has a simpler structure, while the Literature and Sem ontologies have a more complex hierarchy

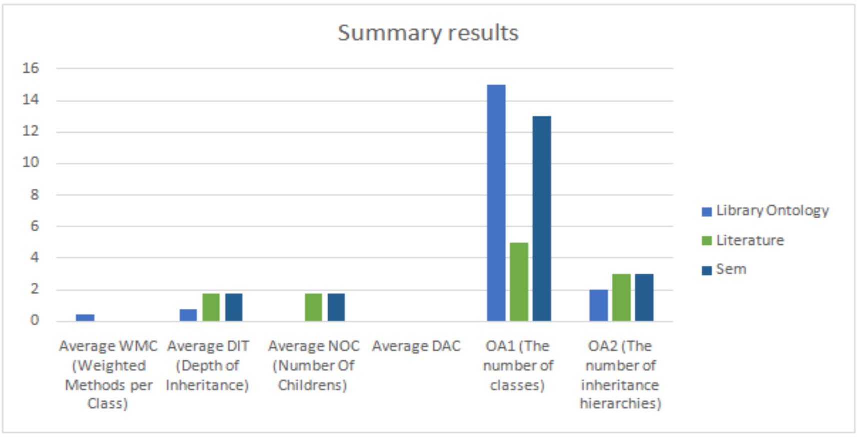

***Figure 14.** Summary results*

4. Conclusions and future work

In this article, three ontologies were presented and evaluated in terms of UML metrics. The three ontologies are the followings: Library, Literature and Sem ontology. All three ontologies can be downloaded as OWL format, from GitHub, all are open-source. The Library ontology represents libraries, the Literature ontology represents literature, and the Sem ontology represents software annotation. The following metrics were evaluated: WMC (Weighted Methods per Class) and Average WMC, DIT (Depth of Inheritance) and Average DIT, NOC (Number Of Childrens) and Average NOC, DAC and Average DAC, OA1, OA2. By evaluating the metrics, it can be seen that the number of classes is large, but the number of class hierarchies and the average number of descendants are small.

Acknowledgement

Supported by the ÚNKP-22-3 New National Excellence Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund.

References

[1] Genero, M., Piattini, M. & Calero, C. (2005). A survey of metrics for UML class diagrams. *Journal of object technology*, 4 (9), pp. 59–92.

[2] Manso, M. E., Genero, M. & Piattini, M. (2003, June). No-redundant metrics for UML class diagram structural complexity. In: *International Conference on Advanced Information Systems Engineering*, Springer, Berlin, Heidelberg, pp. 127–142. https://doi.org/10.1007/3-540-45017-3_11

[3] Baroni, A. L. & Abreu, F. B. (2002, October). Formalizing object-oriented design metrics upon the UML meta-model. In Brazilian Symposium on Software Engineering, Gramado-RS, Brazil.

[4] Chen, Y., Boehm, B. W., Madachy, R. & Valerdi, R. (2004, August). An empirical study of eServices product UML sizing metrics. In: *Proceedings. 2004 International Symposium on Empirical Software Engineering, 2004, ISESE'04*. IEEE, pp. 199–206. https://doi.org/10.1109/ISESE.2004.1334907

[5] Lavazza, L. & Agostini, A. (2005). Automated Measurement of UML Models: an open toolset approach. *J. Object Technol.*, 4 (4), pp. 115–134.

[6] Genero, M., Miranda, D. & Piattini, M. (2002). Defining and validating metrics for UML statechart diagrams. *Proceedings of QAOOSE*.

[7] Fourati, R., Bouassida, N. & Abdallah, H. B. (2011). A metric-based approach for anti-pattern detection in UML designs. In: *Computer and Information Science 2011*, Springer, Berlin, Heidelberg, pp. 17–33). https://doi.org/10.1007/978-3-642-21378-6_2

[8] *OWL documentation*. https://www.w3.org/OWL/, accessed: 29. 11. 2022.

[9] *OWL guide*. https://www.w3.org/TR/2004/REC-owl-guide-20040210, accessed: 29. 11. 2022.

[10] *Library ontology*. https://github.com/ayesha-banu79/Owl-Ontology, accessed: 29. 11. 2022.

[11] *Literature ontology*. https://github.com/detnavillus/rdf-owl-ontologies.git, accessed: 29. 11. 2022.

[12] *sem ontology*. https://github.com/lindenb/semontology.git, accessed: 29. 11. 2022.

[13] Agárdi, A. (2023). Ontology metrics as UML metrics aspect. *Production Systems and Information Engineering*, 11 (3), pp. 128-139 https://doi.org/10.32968/psaie.2023.3.10