# Galois Sub-Hierarchies Used for Use Case Modeling

Ants Torim[1]

Tallinn University of Technology, Ehitajate tee 5, 19086 Tallinn, Estonia
ants.torim@ttu.ee

**Abstract.** Use case diagrams are the core diagrams of the Unified Modeling Language (UML), de facto standard for software modeling. They are used to visualize relations between the users (Actors) and the functionality of the software system (Use Cases). Galois sub hierarchy (GSH) is a sub-order of the concept lattice that contains only concepts with object or attribute labels. This paper investigates the viability of GSH for visualizing the information contained within use case diagrams. While it is possible that a GSH diagram is more complex than a use case diagram for certain formal contexts a study of 87 student projects found no such case. On average, use case diagrams had 3.7 times more graphical elements than corresponding GSH diagrams, demonstrating the viability of GSH as a more compact alternative to the use case diagram.

## 1 Introduction

Software engineering has a long tradition of graphical modeling, there are many different diagram types like flowcharts, BPMN, ERD and even languages like UML containing over dozen diagram types. Most of these diagrams have elements connected with directed or non-directed connections. Each element is represented as a node and each connection as a line between these nodes. While this approach is easy to understand and apply, methods of Formal Concept Analysis (FCA) like Galois sub hierarchies (GSH) can represent the same information in a more concise way, significantly reducing the number of graphical elements. This is achieved because a single node in GSH diagram can represent several elements (it can have many labels) and a line can represent many connections. This conforms to the "reduce redundant data-ink" principle from E. Tufte's classic work on visual information displays [19]. GSH diagram makes it easy to see which elements have same connections and which element has a subset of other elements connections inviting interesting comparisons.

GSH diagrams are most natural to use when there are two types of elements and connections between these (a bipartite graph). This applies to the UML use case diagram that describes actors, use cases and connections between them. A study presented here compares the GSH approach with the UML use case diagram. There is also a brief overview about describing the connections between use cases and data tables with diagrams, information present in CRUD matrix, another traditional software engineering artifact.


ISBN 978–2–7466–6566–8, Laboratory L3i, University of La Rochelle, 2013.

## 2 Galois Sub-Hierarchies

Our method of visual representation is based on Galois sub- hierarchy (GSH) diagrams from the field of Formal Concept Analysis (FCA). This article uses some FCA terminology (formal concept, concept lattice, extent, intent) without explanation and definitions, these can be found from many foundational articles of this field [21], [9], [22], [23] . GSH is a subset of the concept lattice that contains only labeled concepts. More formally, concept $(A, B)$ from the formal context $(G, M, I)$ belongs to the GSH if and only if for some object $g \in G$, $(A, B) = (\{g\}'', \{g\}')$, or dually, for some attribute $m \in M$, $(A, B) = (\{m\}', \{m\}'')$. GSH as a tool for software engineering was introduced by Godin and Mili [10] for the purpose of class hierarchy re-engineering.

This work differs from the standard FCA practice as the main area of interest is not finding the concepts but visualizing the connections between the elements of $G$ and $M$ in a concise way. Semantics of $G$ and $M$ can vary: objects and attributes, use cases and actors, use cases and data tables. Users of GSH diagrams would need to be acquainted with the following properties to see the connections between $G$ and $M$:

1. GSH diagrams show nodes (concepts), connections between them and labels from the sets $G$ and $M$ attached to the nodes. Each element from $G$ and $M$ has exactly one corresponding label.
2. $g \in G$ is connected to $m \in M$ iff there is an upward path from label $g$ to label $m$ or they are labels of the same node.
3. If $g_1, g_2 \in G$ and there is an upward path from $g_1$ to $g_2$ then the set of elements $g_2$ is connected to, $g_2'$, is a subset of $g_1'$.
4. Dually, if $m_1, m_2 \in M$ and there is a downward path from $m_1$ to $m_2$ then the set of elements $m_2$ is connected to, $m_2'$, is a subset of $m_1'$.
5. If $g_1, g_2 \in G$ and $g_1$ and $g_2$ are labels of the same node then $g_2' = g_1'$.
6. Dually, if $m_1, m_2 \in M$ and $m_1$ and $m_2$ are labels of the same node then $m_2' = m_1'$.

Figure 1 presents a simple formal context where $G = \{1, 2, 3, 4, 5, 6\}$ and $M = \{a, b, c, d, e, f\}$ and its corresponding GSH diagram.

| | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| 1 | | x | | | x | |
| 2 | x | | | | | |
| 3 | | | | | x | |
| 4 | | | | x | | x |
| 5 | | x | | | | |
| 6 | | | x | x | | x |

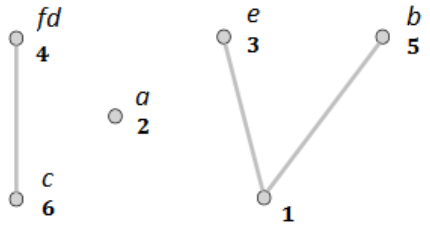


**Fig. 1.** A formal context with the corresponding GSH diagram.

There are several tools for concept lattice generation: Concept Explorer [24], ToscanaJ [4], GaLicia [20]. Of these, only GaLicia supports Galois sub-hierarchies, but its labeling scheme is not convenient for our purposes as its labels contain full concept intents and extents, therefore same element can appear many times in different labels. Two freely available tools were developed as bachelor theses, supervised by the author. One is *GSH builder* by Kristo Aun [3], another is *GSH* by Maarja Raud [15]. Both generate GSH diagrams that show the labels, not extents or intents.

## 3 Use Cases and Actors

Use Case modeling is a common tool for specifying functional requirements. An actor is something with behavior (person, computer system) who interacts with our system. A use case is a collection of related success and failure scenarios that describe an actor using a system to support a goal [13]. A detailed description of the use case is given in a text document while the use case diagram shows a general overview: actors and their relationships to use cases. Use case diagram can also show include and extend relations between use cases. A use case diagram is a diagram type within Unified Modeling Language. There are many books written about the topic including [16] and [13].

Following example (Figure 2) is redrawn from Craig Larmans partial use case diagram describing NextGen sales system: a computerized application used to record sales and handle payments in a retail store [13](pp. 29, 71). This is a basic use case diagram showing use cases, actors and connections between them.

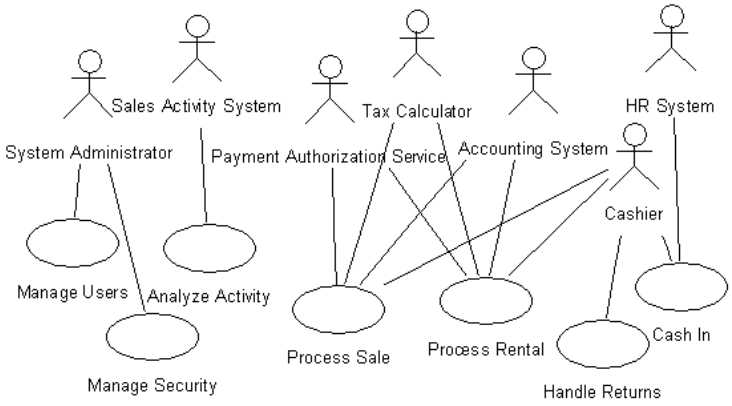


**Fig. 2.** Use case diagram. Actors, like System Administrator, are shown as stick figures. Use cases, like Manage Users, are shown as ovals. Actors participation within a use case is shown as a line. Spatial arrangement conveys no information here, unlike in diagrams of FCA.

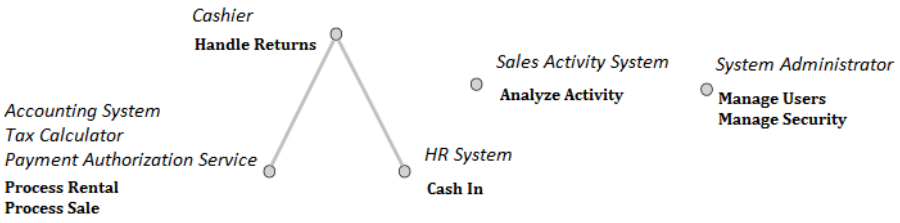


**Fig. 3.** GSH diagram showing connections between use cases and actors.

Figure 3 shows a GSH diagram, equivalent to the use case diagram from Figure 2. It is more concise with only 5 nodes and 2 lines, compared to 14 nodes and 14 lines from Figure 2. A comparison of diagram types based on counting the number of visual elements may seem simplistic but it is in accordance with the principle stated by E. Tufte in his influential work on information displays [19]: " erase redundant data-ink, within reason". Possible reasons for redundancy being: "giving a context and order to complexity, facilitating comparisons over various parts of data, perhaps crafting an aesthetic balance." It is much easier to see from GSH diagrams the actors that are related to same use cases, for example Accounting System, Tax Calculator and Payment Authorization System. GSH diagram makes also visible subset relationships between the use case sets that actors participates in, for example Cashier can do anything that a HR System can do. Therefore GSH diagram both reduces the data ink and compares favorably with the use case diagram in giving a context and order to complexity and facilitating comparisons over various parts of data.

Use case diagrams can contain relations between the use cases or between the actors. Relating use cases is described by C. Larman [13] as "an optional step to possibly improve their comprehension or reduce duplication" GSH diagram showing relations between actors and use cases can not contain this information. Figure 4 presents an example about generalization and include relationships. Generalization is shown as a relation with a big arrowhead from less general subtype to more general supertype. Subtype inherits relations that its supertypes have. Actor *Moderator* is a subtype of an actor *User* and thus inherits its connection to *Post a comment* use case. Generalization relation between use cases is defined dually. Generalization relations can be used to reduce the number of connections within the use case diagrams. While defining formal contexts we add inherited relations to the subtypes.

Include and extend relations between use cases describe sub-functionality: more complex use case includes the behavior of a smaller use case. They allow to introduce different levels of abstraction: A. Cockburn [6] defines three common levels of abstraction: *summary*, *user goal* (default level) and *sub-function* level.

He also mentions *very high summary* (used rarely) and *too low* (should not be used) abstraction levels. S. Ambler recommends to avoid more than two levels of use case associations [1]. Use cases *Post a comment* and *Delete inappropriate comment* include a common *Log in* sub-functionality. Use case *Manage comments* includes use cases *Post a comment* and *Delete inappropriate comment*. Levels of abstraction different from the default *user goal* level are shown through UML stereotypes.

Extend and include arrows correspond to the direction of reference within the use case documentation. In the case of an include relation, use case containing the sub-functionality has a reference to it, in the case of an extend relation, the sub-functionality has a reference to the use case containing it. Extend relation is treated here as an include relation going to the opposite direction.

A method used here to deal with the include and extend relations is to focus on a single level of abstraction (preferably *user goal* or *summary*). Use cases at higher or lower levels of abstractions are removed and their relations to actors are added to the use cases they have include/extend relations with. This can introduce superfluous relations: it is impossible to deduce from the use case diagram if a particular actor from the higher level use case participates in certain sub-functionality or not, use case text has to be examined for that.

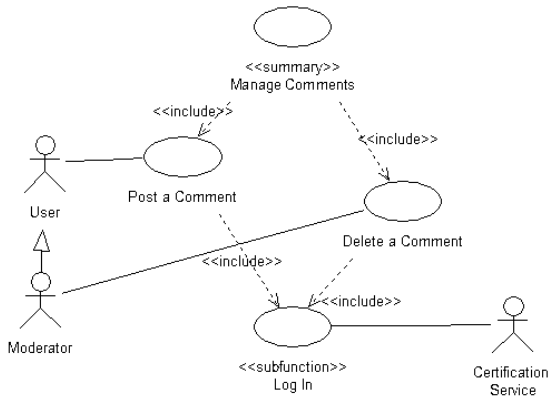


**Fig. 4.** Use case diagram with generalization and include relations.

Figure 5 shows how previous diagram (Figure 4) has been flattened as described to the *user goal* abstraction level through the removal of generalization and include relations and is now in the form that can be used for GSH generation.

GSH diagrams scale well when the number of use cases increases. Figure 6 is based on the example project *Chair in University* from the course "Introduction to information systems" in Tallinn University of Technology. Original documen-

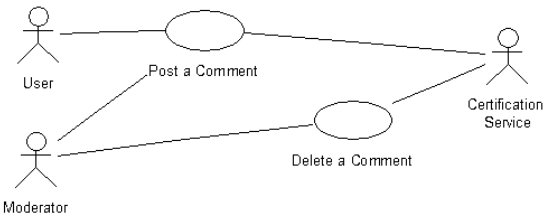


**Fig. 5.** Previous use case diagram flattened to the user goal level of abstraction.

tation had entire use case model split into four use case diagrams containing 25 use cases, 3 actors and 30 connections between actors and use cases. These 4 diagrams are not reproduced here due to limited space. Equivalent GSH diagram contains 5 nodes and 4 lines. That is a significant improvement in conciseness.

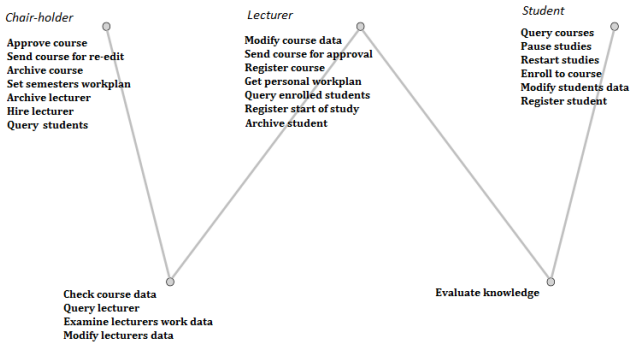


**Fig. 6.** Labelled line diagram for Chair in University information system.

In the previous examples GSH diagrams have all been simpler (less nodes, less connections) than use case diagrams. It is easy to see that GSH diagram can have no more nodes than the corresponding use case diagram as each GSH node must have at least one label and labels don't repeat. However, for certain contexts, GSH diagram can have more connections than the corresponding use case diagram, as shown in the following example.

Figure 7 shows a formal context with 16 connections and a corresponding GSH diagram with 18 connections. That raises a question if GSH diagrams are really simpler than use case diagrams for practical applications. Following study tries to answer this.

| | A | B | C | D |
|---|---|---|---|---|
| 1 | x | x | | |
| 2 | x | | x | |
| 3 | x | | | x |
| 4 | | x | x | |
| 5 | | x | | x |
| 6 | | | x | x |
| 7 | x | x | x | x |

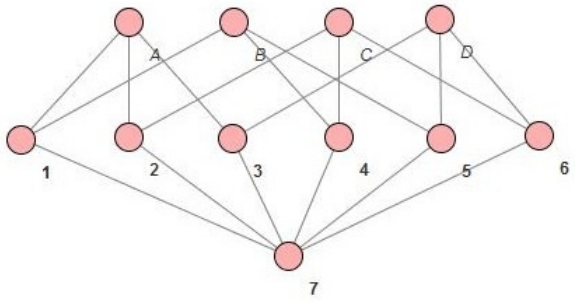


**Fig. 7.** A formal context with a corresponding GSH diagram that has more lines than the number of original connections.

## 4 Study

A study of 87 student projects, that were presented to the author for 2012 "Introduction to information systems" course, was completed to compare the use case and GSH diagrams. Student projects contained the analysis documentation for a freely chosen information system, including use case diagrams. Some of these projects were done in groups and were larger and there was also a variation of effort and quality. For all these projects a corresponding GSH diagram was generated, based on its use case diagram.

Some diagrams contained generalization relations between actors. In this case sub-actors inherited all the relations from super-actors in the corresponding formal context. Some diagrams contained ≪include≫relationships between use cases. For these cases, only use cases at the *user goal* level were kept, use cases included in these and their connections with actors were merged into the *user goal* level use cases as described in the previous section. Use case and connection counts are for diagrams after removing the use cases not at the *user goal* level of abstraction but before the removal of generalization relations. Generalization relation is counted as one line.

**Table 1.** Results of the study. $UC$: number of use cases, $A$: number of actors, $L$: number of lines in the use case diagram, $GSH_C$: number of concepts in GSH, $GSH_L$: number of lines in GSH.

| | $UC$ | $A$ | $UC+A$ | $L$ | $UC+A+L$ | $GSH_C$ | $GSH_L$ | $GSH_{C+L}$ |
|---|---|---|---|---|---|---|---|---|
| Minimum | 3 | 2 | 6 | 4 | 10 | 2 | 0 | 2 |
| Average | 10.77 | 3.55 | 14.32 | 14.86 | 29.18 | 4.64 | 3.2 | 7.84 |
| Maximum | 27 | 9 | 32 | 40 | 68 | 13 | 14 | 24 |

Ratio between the average number of elements of the use case diagrams and the GSH diagrams is $(UC + A + L)/GSH_{C+L} = 3.72$.

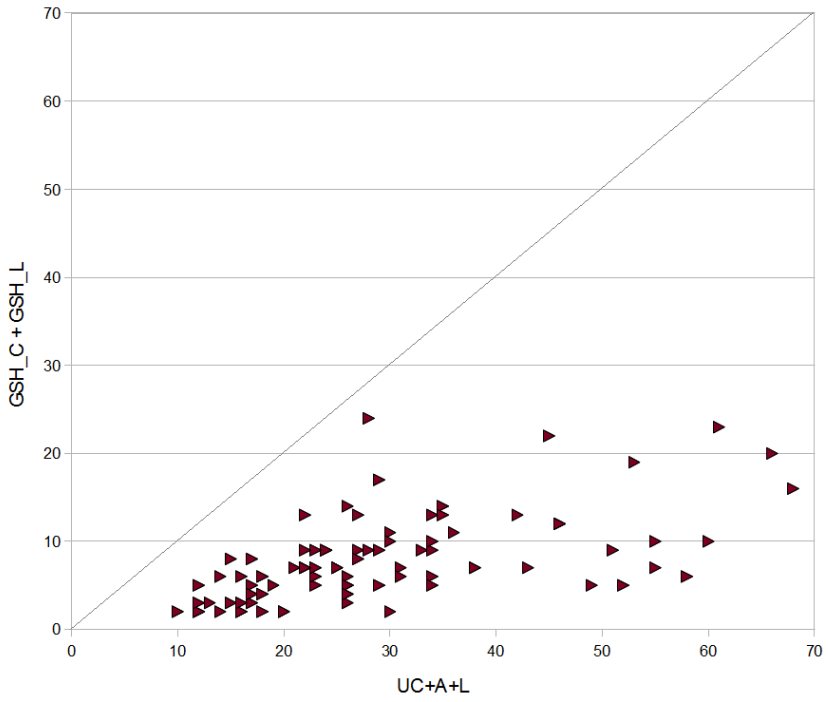


**Fig. 8.** Scatter plot showing the number of visual elements (use case and GSH diagrams) for the 87 student projects.

Figure 8 shows the scatter plot of student projects showing the number of visual elements on use case and GSH diagrams. It is easy to see that in all cases GSH diagram was simpler than use case diagram, as all the data points lie below the diagonal $(UC + A + L) = GSH_{C+L}$ line. This confirms that, at least for the information systems with 30 or less use cases, GSH diagrams are much more concise than the use case diagrams.

Use case diagrams have their own advantages. They are easier to sketch and modify by hand and they are easier to decompose into several diagrams. That seems to suggest complementary roles for use case and GSH diagrams, use case diagrams for quick sketching and GSH diagrams for a well-formated and concise view of the system.

## 5 CRUD matrix

Use cases are not only connected to the actors who require such a functionality but they operate on data tables. GSH diagrams are useful for modeling these connections too. CRUD matrix is a well known artifact of software engineering that describes relations between data tables and use cases. It is described in several popular books about systems design [7] and databases [14]. Use of CRUD matrix as a basis for GSH diagrams was described by author in [18]. It is shortly summarized here to show the usefulness of GSH diagrams for different software engineering activities. There are 4 basic actions performed on data tables by use cases: (C)reate, (R)ead, (U)pdate, (D)elete. In some variations use cases are replaced with actors or business processes.

Table 2 contains a CRUD matrix for a simplified library system. Columns correspond to data tables and rows to use cases. Letters c, r, u, d inside the cells of the matrix correspond to 4 basic actions. For example, use case Add New Task reads data from the table Employee and creates (adds) new data to the table Task.

**Table 2.** CRUD matrix for a simplified library system. Reused from previous article [18].

| | Employee | Reader | Task | Loan | EmployeePosition | Book |
|---|---|---|---|---|---|---|
| Manage readers | | crud | | | | |
| Manage employees | crud | | | | cd | |
| Manage books | | | | | | crud |
| Add loan | | | | c | | r |
| Add new task | r | | c | | | |
| Return loaned book | | | | u | | r |

We can think of a CRUD matrix as defining dependencies between use case layer and data layer. To describe only dependencies we introduce a new binary matrix where all entries with no actions in the original CRUD matrix are replaced with 0 and all entries with at least one action are replaced with 1. We refer to such a matrix as a usage matrix. Table 3 is a usage matrix for Table 2.

It is obvious that usage matrix, and therefore GSH diagram, can be generated automatically from CRUD matrix. That kind of tool could provide visual representation of CRUD matrix without extra effort from the tool user.

Figure 9 is a GSH diagram based on the usage matrix from Table 3. From the GSH diagram it is much easier to see the elements with same dependencies, like use cases *Add loan* and *Return loaned book* and disconnected subsystems, like use case *Manage Readers* with data table *Reader*. GSH diagrams are also helpful for detecting hidden similarities/isomorphisms between different contexts. It is

**Table 3.** Usage matrix for a simplified library system. Based on Table 2.

| | Employee | Reader | Task | Loan | EmployeePosition | Book |
|---|---|---|---|---|---|---|
| Manage readers | | x | | | | |
| Manage employees | x | | | | x | |
| Manage books | | | | | | x |
| Add loan | | | | x | | x |
| Add new task | x | | x | | | |
| Return loaned book | | | | x | | x |

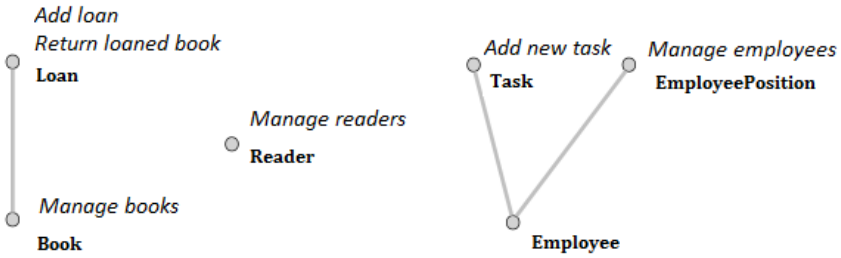


**Fig. 9.** Labelled line diagram based on Table 3.

much easier to see that GSH diagrams from Figures 1 and 9 are isomorphic than that matrices from Figure 1 and Table 3 are isomorphic.

## 6 Related work

The use of methods of FCA for software engineering is not a novel idea. A thorough survey of FCA support for software engineering activities is given in [17]. Most of such research is about extracting potential class hierarchies from different contexts.

Dolques et al [8] propose a FCA-based method for simplifying use case diagrams through the introduction of new generalizations. The result of this method is a refactored use case diagram. Their method preserves the information of include and extend relations. Reduction of diagram elements seems to be smaller than with GSH based method reviewed here though these results are hard to compare exactly as they present them in the terms of edge density (ratio of existing edges to all possible edges). It is possible that the edge density goes down while the actual number of edges increases when new generalized use cases and actors are introduced.

Wolfgang Hesse and Thomas Tilley [12] use a concept lattice connecting use cases and "things" as a tool for identifying candidate classes for object oriented design. There has been much research into using FCA and GSH for class hierarchy design [10], [11]. Algorithms for GSH generation are compared in the article by Arévalo et al [2], a newer algorithm Hermes is presented by Berry et al [5].

## 7 Conclusions

GSH diagrams can be a concise replacement for UML use case diagrams. In our study they had 3.7 times less visual elements. GSH diagrams are likely to be useful wherever there are connections between two types of elements: actors and use cases, use cases and data tables, use cases and classes, business processes and use cases and so on.

One area for further research is the software engineering activity of grouping elements into subsystems. Similar use cases can be grouped into functional subsystems, similar data tables can be grouped into registers. GSH and concept lattice diagrams can help here by organizing elements by similar connections. Use cases that depend on the same data tables are likely to be similar. Grouping elements with similar connections into same subsystems would also help to minimize the connections at subsystem level. GSH diagrams can also be used to visualize the subsystem level connections, thus promising to be a quite universal tool for software engineering.

## References

1. Ambler, S., W. (2005): The Elements of UML 2.0 Style. Cambridge University Press.
2. Arévalo, G., Berry, A., Huchard, M., Perrot, G., Sigayret, A. (2007): Performances of Galois Sub-Hierarchy-Building Algorithms. Formal Concept Analysis, LNCS, vol. 4390, 166-180.
3. Aun, K.: Galois sub-hierarchy builder. [WWW] http://sourceforge.net/projects/gshbuilder/ (15.09.2013).
4. Becker, P., Hereth, J., Stumme, G. (2002): ToscanaJ - an Open Source Tool for Qualitative Data Analysis. Advances in Formal Concept Analysis for Knowledge Discovery in Databases. Proc. Workshop FCAKDD of the 15th European Conference on Artificial Intelligence, 1-2.
5. Berry, A., Huchard, M., Napoli, A., Sigayret, A. (2012): Hermes: an Efficient Algorithm for Building Galois Sub-hierarchies. CLA 2012: 21-32.
6. Cockburn, A. (2000): Writing Effective Use Cases. Boston: Addison-Wesley.
7. Dennis, A., Haley Wixom, B., Roth, R. (2008): Systems Analysis and Design, 4th ed. Wiley.
8. Dolques, X., Huchard, M., Nebut, C., Reitz, P. (2012): Fixing Generalization Defects in UML Use Case Diagrams. Fundam. Inform. 115(4): 327-356.
9. Ganter, B.,Wille, R. (1998): Formal Concept Analysis, Mathematical Foundations. Berlin: Springer.
10. Godin, R., Mili, H. (1993): Building and Maintaining Analysis-Level Class Hierarchies using Galois Lattices. Proceedings of OOPSLA 28, 394-410.

11. Godin, R., Valtchev, P. (2005): Formal Concept Analysis-Based Class Hierarchy Design in Object-Oriented Software Development. Formal Concept Analysis 2005, LNCS, vol. 3626, 304-323.
12. Hesse, W., Tilley, T. (2005): Formal Concept Analysis Used for Software Analysis and Modelling. Formal Concept Analysis 2005, LNCS, vol. 3626, 259-282.
13. Larman, C. (2002): Applying UML and Patterns, 2nd ed. Upper Saddle River, NJ: Prentice Hall.
14. Oppel, A. J. (2004): Databases Demystified. New York: McGraw-Hill.
15. Raud, M.: GSH. [WWW] http://staff.ttu.ee/ torim/fca.html (15.09.2013).
16. Rumbaugh, J., Jacobson, I., Booch, G. (1999): The Unified Modelling Language Reference Manual. Boston: Addison Wesley.
17. Tilley, T., Cole, R., Becker, P., Eklund, P. (2005): A Survey of Formal Concept Analysis Support for Software Engineering Activities. Formal Concept Analysis 2005, LNCS, vol. 3626, 250-271.
18. Torim, A. (2011): A Visual Model of the CRUD Matrix. Proceedings of the 21th European-Japanese Conference on Information Modelling and Knowledge Bases. (Ed.) Jaak Henno, Yasuhi Kiyoki, Takehiro Tokuda, Naofumi Yoshida. Tallinn: TTU Press, 114 - 122.
19. Tufte, E. R. (2001): The Visual Display of Quantitative Information, 2nd ed. Cheshire, Connecticut: Graphics Press.
20. Valtchev, P., Grosser, D., Roume, C., Hacéne, R. (2003): Galicia: an Open Platform for Lattices. Using Conceptual Structures: Contrib. to the 11th ICCS, 241-254.
21. Wille, R.: Restructuring lattice theory (1982): an approach based on hierarchies of concepts. Ordered Sets, pp. 445-470.
22. Wille, R., Stumme, G., Ganter, B. (2005): Formal Concept Analysis: Foundations and Applications, Berlin: Springer.
23. Wille, R. (2005): Formal Concept Analysis as Mathematical Theory of Concepts and Concept Hierarchies. Formal Concept Analysis 2005, LNCS, vol. 3626, 47-70.
24. Yevtushenko, S. A. (2000): System of Data Analysis "Concept Explorer" (In Russian). Proceedings of the 7th national conference on Artificial Intelligence KII, 127-134.