DETECTING DEFECTS IN OBJECT-ORIENTED DIAGRAMS USING UML-HAZOP

Janusz GÓRSKI[*], Aleksander JARZĘBOWICZ[*]

Abstract. The paper presents a method supporting detection of defects in UML based software documentation. The method named UML-HAZOP is the adoption of HAZOP (Hazard and Operability Studies) – a technique widely applied to safety-related systems, and concentrates on analysing "flows" between system's components in order to detect anomalies related to these flows. The adoption of HAZOP to UML diagram analysis required defining proper interpretations of "flows" and "anomalies" with respect to each UML diagram. At present, UML-HAZOP has been defined and checked in some industrial case studies for class diagrams. Further works aim to extend it to other UML diagrams. This paper describes the method and the results of some experiments related to its application to two real systems: a billing system of a telephone exchange and a management support system.

1. Introduction

One of the most important problems concerning software engineering is the presence of defects in developed software. Identification and correction of these defects requires significant time and financial resources and can also mean overcoming the planned schedule and budget. Undetected residual defects that are left in software delivered to the client result in decreased level of users' satisfaction.

[*] Department of Applied Informatics, Technical University of Gdansk, Narutowicza 11/12, 80-952 Gdańsk, Poland. Email: jango@pg.gda.pl, olek@eti.pg.gda.pl.

Defects can be injected in the early phases of software development e.g. requirements specification or system analysis. If they are not detected and corrected early enough, they are „passed" to the subsequent software representations and the cost of their correction usually increases significantly.

This paper addresses the problem of defects introduced to the object models developed during the analysis and design phases of software lifecycle. We concentrate on models expressed using UML notation [1], because of its growing popularity in the industrial software production.

The analysis of UML models to detect defects is not a simple task. UML has a partially formalised (graphical) syntax, but its semantics is still informal and depends on human's interpretation. Therefore the analysis process can be automated only to a limited extent and in most cases the final decisions (what is a defect and what is not) have to be made by humans. Additional difficulties arise from the complexity the models. Even if proper techniques of reduction of complexity (abstraction, decomposition) are being applied, the models often are quite complex and difficult to understand. Therefore it is desirable that a method of analysis concentrates on just a part of a model once at a time and this way reduces the analyst's intellectual effort. Such methods can be called *partial analysis methods*. An example of partial analysis could be checking if the name of each class is specified according to given rules – the analyst's attention is just paid to each class in separation to its name, without the need to investigate its role in whole system.

The fact, that more advanced analysis of UML models requires participation of humans (at least until the semantics of these models become formalised to higher extend that is rather unlikely in the nearest time) doesn't mean that the analysis process should be executed in *ad hoc* manner, without proper planning and controlling the activities. The obvious requirements for an analysis method ask for it being:

- Complete – the analysis is conducted in a way that guarantees detection of all defects from within the scope of the method;
- Systematic – the method guides the analyst and leads him/her through the model focusing his/her attention on just those aspects that are important for detecting the addressed kinds of defects;
- Universal - the method can be applied to the systems of all kinds, without any limitations related to their application and environment nor technologies used to their development;
- Supported – the application of the method is supported by appropriate tools.

One of possible ways to "discover" a new analysis method is an adoption of an existing method that have proved its effectiveness in other problem areas. This is the approach we present in this paper. Our subject of interests is HAZOP (Hazard and Operability Studies), the method widely applied to analyse safety-related systems. HAZOP concentrates on connections between system's components and explores, if and how deviations from intended values of connection's attributes can lead to a failure of the considered system. Deviations are suggested to the analyst by the guidewords, which in association with particular attributes describe specific situations that need to be taken under consideration.

The research on adopting HAZOP to software analysis took place in the first half of the nineties and was summarised in *Interim Defence Standard 00-58* [2], published by the British Ministry of Defence. The standard contains the description of the analysis process and the criteria of identifying defects in various representations of systems. However, these criteria are defined in rather general way and refer to notations that are now being replaced by the *Unified Modelling Language (UML)*. The problem of adaptation of HAZOP to analysis of systems expressed using various software modelling notations was also described in other publications ([3],[4]). There were also attempts to adopt the method to more specific problems, e.g.

generation of mutation operators for basic elements of JAVA [5], formalised safety analysis of user interface [6], security analysis [7].

This paper presents the results of our work aimed at adoption of HAZOP to the analysis of object-oriented models based on the UML language. Adoption requires: giving proper (specific to particular UML models) interpretations to connections, connection attributes, HAZOP guidewords and deviations; defining the procedure to be followed during the analysis (the analysis process); and conducting the experimental validation of the proposed solution. The results obtained in each of these areas (interpretation of deviations, process definition and experimental validation) are presented in the succeeding sections.

2. UML-HAZOP method

In the rest of this text we assume that during the analysis an element classified by the analyst as incorrect or doubtful is called *anomaly*. An anomaly confirmed as an actual fault by an independent reviewer is called *defect*.

To adopt HAZOP to UML notation, it is necessary to define: which elements of particular UML diagrams are considered as HAZOP “connections” and “attributes” and which type of anomaly is suggested by applying a particular HAZOP guideword to a particular attribute. As the result we obtain checklists that list all possible anomalies of a considered element of the model.

We use the set of guidewords proposed in the HAZOP standard [2]. It is shown in Table 1.

Table 1. Generic HAZOP guidewords

Guideword	**Generic interpretation**
NO	The complete negation of the design intention. No part of the intention is achieved and nothing else happens.
MORE	A quantitative increase.
LESS	A quantitative decrease.
AS WELL AS	All the design intention is achieved together with additions.
PART OF	Only some of the design intention is achieved.
REVERSE	The logical opposite of the intention is achieved.
OTHER THAN	Complete substitution, where no part of the original intention is achieved but something quite different happens.
EARLY	Something happens earlier than expected relative to clock time.
LATE	Something happens later than expected relative to clock time.
BEFORE	Something happens before it is expected, relating to order or sequence.
AFTER	Something happens after it is expected, relating to order or sequence.

Interpretations of deviations suggested by these guidewords with reference to the UML class diagram elements are described below.

In the class diagram, the “connection” (in the sense of HAZOP) corresponds to the *relationship* between classes. According to UML [1] the following kinds of relationships are used in class diagrams:

- *Association* – semantic relationship between classes;
- *Dependency* – relationship stating that implementation or functioning of one or more classes requires the presence of one or more other classes;
- *Flow* – relationship between two versions of the object or between an object and a copy of it;
- *Generalization* – inheritance relationship between classes;

We first concentrated on the most often used kinds of the relationships: *Association* and *Generalization*. The next step was to define the mapping between HAZOP attributes and the elements of the UML diagram. This mapping was not obvious. Both, the relationships and their properties could be considered as HAZOP attributes. The assumed interpretations for the relationships are shown in Tables 2 and 8 and the assumed interpretations for the properties of the relationships are shown in Tables 3 to 7. These properties (which are incidentally also called “attributes” in UML specification) are listed below:

Property of the *Association* relationship*:*

- *name* (name of association)

Properties of *Association* from the point of view of a single class involved in the relationship:

- *AssociationEnd:*
  - *aggregation* (presence of the part-whole relationship)
  - *changeability* (ability of modification by the associated class)
  - *ordering* (ordering of the set of instances)
  - *isNavigable* (ability of transversal from the associated class)
  - *multiplicity* (multiplicity of association)
  - *name* (name of class’ role in association)
  - *targetScope* (specifies if associated is a class or its instance)
  - *visibility* (specifies if the class is visible to its associated class)

Property of the *Generalization* relationship:

- *discriminator* (the criterion of inheritance)

The interpretations assigned to the attribute-guideword pairs are shown in tables below. The tables include only those entries that provided sensible interpretations, the others were omitted.

Table 2. **Interpretations of anomalies related to *Association***

Guideword	**Interpretation**
AS WELL AS	Considered classes are not really associated. The association is wrong and should be removed from the diagram.
OTHER THAN	Wrong type of the relationship. A relationship of another type e.g. *Generalization* should replace the *Association.*
PART OF	*Association* is correct, but some additional relationship or relationships should be introduced between the considered classes.

Table 3. **Interpretations of anomalies related to *Association.name***

Guideword	**Interpretation**
NO	The association has no name though it should be named.
REVERSE	Wrong direction of reading the name of the association between the classes.
OTHER THAN	The name of association is wrong and doesn't represent the meaning of the association. Another name should be used.
AS WELL AS	The name of association is too general, it should be more detailed.
PART OF	The name of association is too detailed, it should be more general.

Table 4. Interpretations of anomalies related to *AssociationEnd.aggregation*

Guideword	**Interpretation**
NO	The aggregation that should be present is not marked in the diagram.
AS WELL AS	The aggregation is wrong and should be removed.
OTHER THAN	Wrong type of aggregation is marked i.e. strong instead of weak or inversely.
MORE	Too many classes are aggregated, some of them shouldn't be included in this aggregation.
LESS	The aggregation doesn't include some classes (present on the diagram or not) that should be aggregated.

Table 5. Interpretations of anomalies related to *AssociationEnd.ordering*

Guideword	**Interpretation**
NO	No ordering though it should take place.
AS WELL AS	The ordering that shouldn't take place is marked.

Table 6. Interpretations of anomalies related to *AssociationEnd.multiplicity*

Guideword	**Interpretation**
MORE	The marked multiplicity of association is too high (it can concern upper or lower bound of multiplicity).
LESS	The marked multiplicity of association is too low (it can concern upper or lower bound of multiplicity).

Table 7. Interpretations of anomalies related to *AssociationEnd.name*

Guideword	Interpretation
NO	The side of the association has no name though it should be named.
OTHER THAN	The name of the association end is wrong, another name should be used.
AS WELL AS	The name of the association end is too general, it should be more detailed.
PART OF	The name of the association end is too detailed, it should be more general.

Table 8.* Interpretations of anomalies related to *Generalization

Guideword	Interpretation
AS WELL AS	Considered classes are not really related with *Generalization* relationship. The relationship is wrong and should be removed from the diagram.
MORE	Some additional classes are marked as subclasses, they shouldn't take part in this relationship.
LESS	Some classes (present on the diagram or not) that should take part in this relationship are not marked as subclasses.
OTHER THAN	Wrong type of relationship. A relationship of another type should be defined e.g. *Association* instead of. *Generalization.*

3. UML-HAZOP process

UML-HAZOP analysis process is shown in Figure 1.

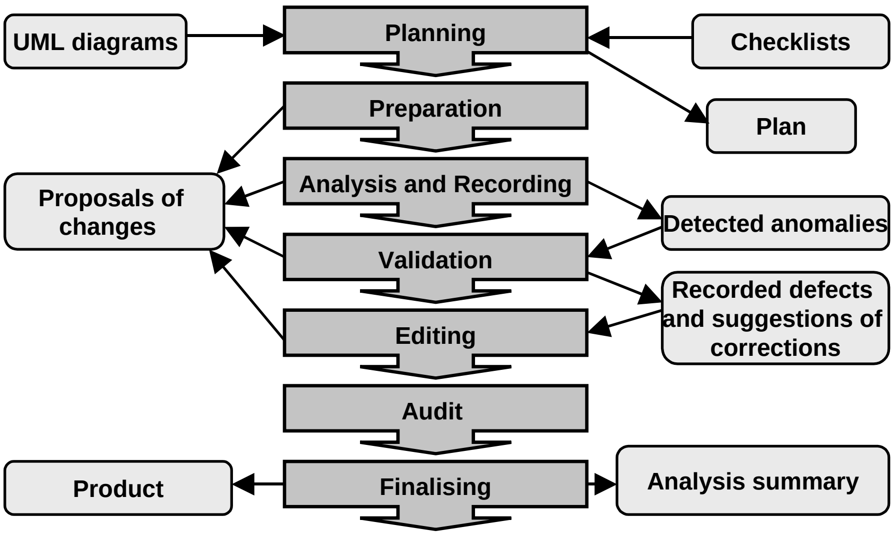

Figure 1. The structure of UML-HAZOP analysis process

The first step of the process is *Planning*, where the decisions about the subject and the scope of the analysis are being made, the proper checklists are being prepared, the participants are selected and the schedule of the next steps is established. *Preparation* includes activities necessary to the acquaint participants with the method and with the subject of the analysis (the analysed system and its models).

In the *Analysis* and *Recording* steps, the UML diagrams are systematically reviewed using the checklists. Detected anomalies are recorded in the previously prepared data structures shown in Table 9.

Table 9. Structure for recording the results

Relationship: <identification of relationship>		
Attribute	**Guideword**	**Result**
<attribute name>	<HAZOP guideword>	<description>

While filling in the “Result” field the following symbols are used:

D – an anomaly, initially classified as a defect,

Q – a question, an ambiguous element that needs to be further discussed,

C – a change (made because an anomaly in another model element was found e.g. if the analyst decided to remove an aggregation from the association end, a name should be given to that association)

N – no anomaly, everything is OK,

X – not analysed, skipped.

All the analysed cases are registered, including these ones, for which no anomaly was found.

In the *Validation* step the final decision is undertaken on which anomalies are classified as defects and the importance of those defects is being assessed. This step requires the participation of experts who are familiar with the considered system and its domain. As the result, the list of detected defects and the qualification of their importance (negligible, significant, critical) is created. Although some suggestions of changes to the model (to eliminate the detected defects) could also be made, it is not the main purpose of the analysis but rather a (desirable) side effect of the process. The participation of the author of the analysed model is highly recommended in this step. If more experts are involved in the validation, their meeting should be properly moderated to prevent long discussions and to assure its convergence. The techniques supporting achievement of these goals are beyond the scope of this paper.

The list of defects is the input to the *Editing* step. Its purpose is the improvement of analysed models. This step is conducted by the author (or authors) of the model. After that, a formal review takes place to assure that all detected defects were dealt with.

When the process ends, the analysis is formally closed and the summary data (resources consumed, numbers of anomalies and defects detected, effectiveness metrics etc.) is collected and recorded.

During the analysis process, especially in its preparation, analysis, recording, validation and editing steps, suggestions of possible improvements of the analysis process may be formulated and collected. These suggestions can then be used to launch activities aiming at improving the effectiveness and efficiency of the process in a long-term prospects.

4. Experimental validation of UML-HAZOP method

To assess effectiveness of the UML-HAZOP method we planned a series of experiments. In these experiments we used UML models resulting from early phases of software development processes. The main purpose of experiments was to assess the effectiveness of the method while analysing real models. Concentration on the models from the early development phases is essential, because if defects introduced in these phases go undetected are passed to the next representations of system, in most cases the cost of their correction increases significantly. An additional goal (if the good performance of method were confirmed) was to look for such a structure of the analysis process that minimises the costs-to-benefits ratio.

At present, three experiments have been completed. Their brief description is given below.

4.1. Experiment 1: Billing System STACT – a "retrospective" analysis

The subject of analysis was a billing system for a small telephone exchange. The system calculates charges for phone connections and issues bills to the subscribers. It stores information about subscribers, lines, connections, charges etc.

The STACT system is presently over 6 years old. It was developed as a didactic example for Postgraduate Study of Software Engineering at the Technical University of Gdańsk. The documentation of STACT and the system itself are used as a case study by the student groups. The UML models of STACT were reviewed many times in the course of six subsequent editions of the Study.

The UML-HAZOP analysis was performed with respect to two documents: the class diagram representing the results of analysis phase and the system design document. The analyst had no prior insight into the rest of system's documentation, including the requirements specification. The most interesting results obtained regarded the class diagrams and state diagrams. The analysis of use-case and sequence diagrams provided no significant results (most probably because the considered system is rather static and its dynamic diagrams are very simple). The analysed class diagram consisted of 17 classes and 14 relationships.

4.2. Experiment 2: Management Support System SZAFIR – a "retrospective" analysis

The experiment was conducted on models developed for a real software application. The system SZAFIR is an integrated, accessible via Internet system supporting the management of a small firms. It is under development with new functions being added following the evolutionary development paradigm. At present, its functionality includes: accountancy, logistics, selling, invoicing, pricing and others.

The system is developed by Prokom Software S.A., a leading Polish software house, that became interested in our research in UML-HAZOP and offered its co-operation during the case study. The delivered documentation comprised UML models and specifications of two recently developed subsystems. The analysed models included several defects already known to the developers but those defects were not communicated to the people performing the UML-HAZOP analysis.

Only class diagrams were analysed in this experiment. There were two diagrams: the main and supplementary. Altogether the diagrams consisted of 56 classes and 79 relationships.

4.3. Experiment 3: Management System SZAFIR – an "in-line" analysis

This experiment was conducted in cooperation with Prokom Software S.A. and its subject was another subsystem of the SZAFIR system described above. In contrary to the previous experiments, this time the analysis built in the development process, i.e. it was performed before the models were used in following development phases. This time the subject of the analysis were the final versions of the models that had previously passed all the Quality Assurance checks.

5. Results of experiments

The quantitative results of the experiments are given in separate tables (Table 10, 11 and 13). „Number of analysed cases" means number of the cases in which the decision of a human had to be made. This number does not include cases that could be removed automatically, e.g. for an unnamed association we checked for the anomaly "The association has no name though it should be named" only, because the anomalies referring to the accurateness of the name, direction of its reading etc. made no sense in the absence of the name. "Number of significant defects" gives the number of defects that were classified as significantly influencing the further development (it did not include e.g. defects referring to names).

Table 10. Results of experiment 1

Number of people involved in analysis	1
Time of checklists preparation	2 man-hours

Time of studying documentation	4 man-hours
Time of analysis	2,5 man-hours
Time of validation	3,5 man-hours
Number of analysed cases	approx.170
Number of detected anomalies	11
Number of confirmed defects$^{(*)}$	8
Number of significant defects	6

5.1. Experiment 1: Billing system

The results of the analysis are summarised in Table 10. Table 11 gives an example of the data structure containing the results of the analysis.

$^{(*)}$ Anomaly was classified as a confirmed defect when at least two out of three experts who participated in the validation phase admitted that it really is a fault in analyzed model.

Table 11. The example result of analysis

Relationship: is_part_of (Charge,Bill)		
Attribute	**Guideword**	**Result**
Association	AS WELL AS	N
Association	OTHER THAN	N
Association	PART OF	N
Association.name	NO	N
Association.name	REVERSE	N
Association.name	OTHER THAN	N
Association.name	AS WELL AS	N
Association.name	PART OF	N
End 1 - Charge		
AssociationEnd.aggregation	NO	N
AssociationEnd.aggregation	AS WELL AS	N
AssociationEnd.aggregation	OTHER THAN	N
AssociationEnd.aggregation	MORE	N
AssociationEnd.aggregation	LESS	N
AssociationEnd.ordering	NO	D – Elements of bill that inform about charges for particular phone lines should be ordered. In case of large firm or institution that leases many lines it is important.
AssociationEnd.ordering	AS WELL AS	N
AssociationEnd.multiplicity	MORE	N
AssociationEnd.multiplicity	LESS	D – The multiplicity should be 1..*. Every bill consists of at least one charge (or more if the subscriber leases more lines).
AssociationEnd.name	NO	N
AssociationEnd.name	OTHER THAN	N
AssociationEnd.name	AS WELL AS	N
AssociationEnd.name	PART OF	N
End 2 – Bill		
AssociationEnd.aggregation	NO	N
AssociationEnd.aggregation	AS WELL AS	N
AssociationEnd.aggregation	OTHER THAN	D – The charge is included in one bill only, a composition should be marked here.
AssociationEnd.aggregation	MORE	N
AssociationEnd.aggregation	LESS	N

Table 11. The example result of analysis (continued)

Attribute	Guideword	Result
AssociationEnd.ordering	NO	N
AssociationEnd.ordering	AS WELL AS	N
AssociationEnd.multiplicity	MORE	N
AssociationEnd.multiplicity	LESS	N
AssociationEnd.name	NO	N
AssociationEnd.name	OTHER THAN	N
AssociationEnd.name	AS WELL AS	N
AssociationEnd.name	PART OF	N

The analysed class diagram is shown on Figure 2. The elements in which anomalies were detected (and confirmed as defects) are marked with thick circles.

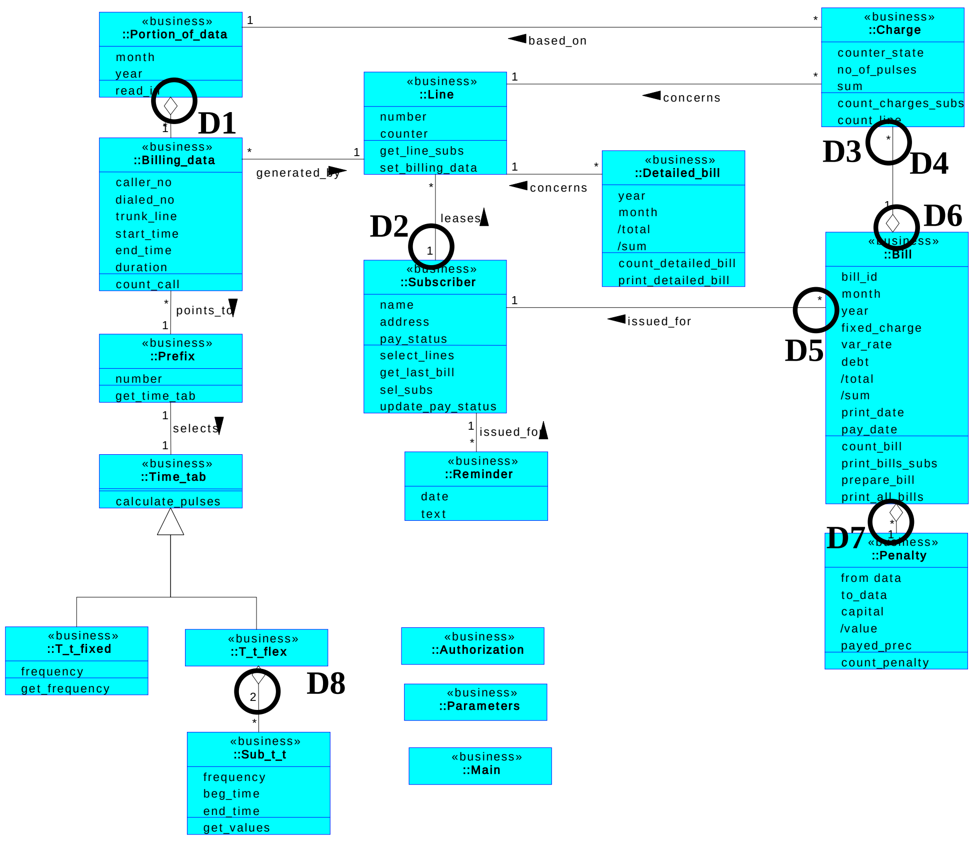

Figure 2. The analyzed class diagram

Description of the defects marked on the diagram:

D1 – There should be a composition instead of aggregation.
D2 – It should be 0..1 multiplicity (there may be "free" lines without associated subscriber).
D3 – The elements of a bill informing about charges for particular a line should be ordered.
D4 – It should be 1..* multiplicity. In every bill is at least 1 charge (or more if the subscriber has more lines).
D5 – Bills viewed from subscriber's side should be ordered according to time.
D6 – The charge is part of only one bill, a composition should be marked here.
D7 – At most 1 penalty (for last month) is included in bill. 0..1 multiplicity should be marked.
D8 – There is no reason for 2 multiplicity, it should be 1.

5.2. Experiment 2: Management support system

The results of the analysis are summarised in Table 12.

Table 12. Results of experiment 2

Number of people involved in analysis	1
Time of checklists preparation	4,5 man-hours
Time of studying documentation	16 man-hours
Time of analysis	8,5 man-hours
Time of validation	16 man-hours
Diagram „Invoices"	
Number of analysed cases	approx. 750
Number of detected anomalies	94
Number of confirmed defects(*)	71
Number of significant defects	35
Diagram „Pricelists"	
Number of analysed cases	approx. 140
Number of detected anomalies	10
Number of confirmed defects(*)	9
Number of significant defects	3

5.3. Experiment 3: Management system – "in-line" analysis

The results of the analysis are summarised in Table 13.

(*) Analysis results were validated by project team members who developed analyzed model.

Table 13. Results of experiment 3

Number of people involved in analysis	1
Time of checklists preparation	1,25 man-hours
Time of studying documentation	9 man-hours
Time of analysis	3,75 man-hours
Time of validation	7 man-hours
Number of analysed cases	approx. 360
Number of detected anomalies	44
Number of confirmed defects*	40
Number of significant defects	5

6. Conclusions

The experimental results confirm a considerable potential of UML-HAZOP method in detecting defects in real applications. It is even more evident when we notice that the method is able to detect defects in the models that were already used in implementation of real systems and were previously subjected to extensive QA checks.

A quite surprising result is a relatively small amount of work put in activities that can not be automated i.e. preparation, analysis and validation. In case of Experiment 1 the overall cost was 10 man-hours, which in comparison with 8 confirmed defects gives 1,25 man-hour for detecting one defect. Under assumption that the average cost of a single defect correction in the late development phases is 10 man-hours (which comparing to the data presented in e.g. [9] is not an overestimated value) the effectiveness of the method amounts 8 (the effectiveness is understood here as the proportion of amount of the saved work to the amount of spent work).

In case of Experiment 2, in which the analysed models were not previously subjected to QA, the effectiveness amounts 19,75 (80 confirmed defects, 40,5 man-hours). For Experiment 3, in which the analysis was performed on finished, verified models, 40 confirmed defects were detected with overall cost of 21 man-hours. It means the effectiveness of the method amounts 19,05.

The results related to the number of detected defects look quite interesting, especially after taking into consideration that the models analysed in Experiments 1 and 3 were previously verified and declared as „finished”. This in particular refers to the billing system because of the long period of its usage and the number of people involved in “reading” its models.

UML-HAZOP is a partial analysis method i.e. it focuses the attention of the analyst on selected parts of the model and is not expected to detect all possible defects. It can’t therefore be treated as the only method to be applied; instead it should rather be used together with other, complementary methods. However, because of its limitation of scope, the method does not require large amount of work spent on preparation and can be applied even in cases when the analyst does not have deep understanding of the whole model but concentrates on its parts only (according to UML-HAZOP checklists). Thanks to that, the costs related to application of UML-HAZOP can be kept relatively small and would not increase fast with the increase of

* Analysis results were validated by project team members who developed analyzed model.

the size of the analysed models. At present it is just a hypothesis and its confirmation requires more experimental works.

It is also worth noticing, that the preparation of the UML-HAZOP checklists can be entirely automated. If the models are stored in an electronic repository (which is becoming a typical situation nowadays), it is not difficult to write a program that automatically generates UML-HAZOP checklists for a chosen model. Such a supporting tool is now presently under development within the EU IST-DRIVE 1999-12040.

References

[1] Object Management Group, *OMG Unified Modeling Language Specification*, Version 1.4, September 2001.

[2] Interim Defense Standard 00-58, *HAZOP Studies on System Containing Programmable Electronics*, UK Ministry of Defense 1996.

[3] McDermid J., Pumfrey D., Development of hazard analysis to aid software design, COMPASS '94: *Proceedings of the Ninth Annual Conference on Computer Assurance*, pp. 17-25.

[4] Fencott C., Hebbron B., The application of HAZOP studies to integrated requirements models for control systems, *Proceedings of Computer Safety, Reliability and Security*, 13th International Conference SAFECOMP 1994.

[5] Kim S., Clark J., McDermid J., The rigorous generation of JAVA mutation operators using HAZOP, *12th International Conference Software and Systems Engineering and their Applications* (ICSSEA '94).

[6] Hussey A., *Safety analysis of the Druide user-interface*, SVRC DCS, The University of Queensland, Technical Report No. 99-13, March 1999.

[7] Winther R., Johnsen O., Gran B., Security assessments of safety critical systems using HAZOPs, *Proceedings of Computer Safety, Reliability and Security*, 20th International Conference SAFECOMP 2001, Springer Lecture Notes in Computer Science 2187, pp. 14-24.

[8] Department of Applied Informatics, Technical University of Gdańsk, *The billing system for a telephone exchange (STACT) – UML Model*, Internal Technical Report, 2000.

[9] *Case Study II: Finding Defects Earlier Saves Big $$$*, http://www.cigital.com/solutions/roi-cs2.html (page checked 29.09.2002).