DRAFT

Full paper published in:

Foundations of Computing and Decision Sciences, Vol. 30 (2005), No. 1, pp. 29-38

EXPERIMENTAL COMPARISON OF UML-HAZOP INSPECTION AND NON-STRUCTURED REVIEW

Aleksander JARZĘBOWICZ[*], Janusz GÓRSKI[†]

Abstract. The paper describes an experiment conducted in the academic environment with participation of students from Gdańsk University of Technology. The aim of the experiment was to compare two review techniques: inspection based on the UML-HAZOP method and a non-structured review. Effectiveness and efficiency of both methods were measured and compared. The paper presents the schedule, realisation and results of the experiment. An introduction to reviews and inspections as well as a brief description of the UML-HAZOP method are also included.

1. Introduction

The general objective of every software project can be defined as: to deliver, within the budget and schedule constraints, a product satisfying the stakeholders' needs. In practice, however, many software projects fail to meet this objective. One of the contributing factors is errors introduced during early development phases (requirements elicitation, analysis, design). Such errors manifest themselves as anomalies in project documentation, especially in documents created early in the system lifecycle. Those anomalies can be of different kinds, e.g. design defects (wrong representation of a problem domain or a solution), failing to satisfy desired metrics or heuristics or violation of specific system attributes like safety or security. An anomaly

[*] Gdańsk University of Technology, Poland. E-mail: olek@eti.pg.gda.pl

[†] Gdańsk University of Technology, Poland. E-mail: jango@pg.gda.pl

which goes undetected reappears in the subsequent representations downstream the development process and the cost of its removal increase rapidly. This creates a strong motivation to detect such anomalies as early as possible within the software lifecycle. The detecting methods are employing the ideas of *inspections* and *reviews* and are focusing on system documentation.

The approach presented in this paper focuses on detecting anomalies in system models like object models, relational models or ontologies. These models are created early in a development lifecycle. They have formalised structure and syntax which provide for defining precise criteria and procedures of the inspection process. The aim of this research is to propose a new inspection method dedicated to detecting anomalies in such models. The work performed so far concentrated on object models expressed in UML and on detecting their design defects. Such defects constitute a common class of anomalies and are difficult to get rid of as their detection is not supported by automatic tools. We proposed the UML-HAZOP inspection method to deal with this class of anomalies. A brief description of the method is presented in section 3. The method was already validated in four case studies. In each of them a model developed (and later implemented) in a real software project was analysed using UML-HAZOP [5]. In each case study a number of significant defects were discovered. Although the experiments supported the claim about the effectiveness of the method we still did not have enough data to compare it with other inspection techniques. We also wanted to learn more about the efficiency of the method, especially when applied by relatively inexperienced users.

To learn more about the method we planned for a new validation experiment. It was a controlled experiment with a larger number of participants divided into two groups. Each group was searching for defects in a given model using a different method (one group was using UML-HAZOP, while the second a more conventional reviewing technique). The objective of the experiment was to compare UML-HAZOP with another technique with respect to their effectiveness and efficiency. A non-structured review (spontaneous process structure, non-specified categories of defects) was selected for comparison with UML-HAZOP.

2. Reviews and inspections

Reviews and inspections have a long history in software engineering. Structured inspections were first introduced in 1976 by Michael Fagan in IBM [1] and, as their effectiveness and efficiency were undoubtedly confirmed, they received much attention and a number of various review techniques were proposed since that time e.g. reviews (also called technical reviews), inspections, walkthroughs, round-robin reviews etc. [2]. Those techniques differ in their scope, aim, organisational structure and process structure. Nevertheless, each of them is based on a single principle that it is most difficult to spot one's own errors and thus it is better to ask someone else to search for them.

In the sequel two techniques: reviews and inspections are discussed in more detail. Both techniques are based on a similar process. The process starts with a kick-off meeting which purpose is to acquaint the participants with the reviewed product, motivate them and (if necessary) to train them how to apply the technique. During the next phase the individual checking is performed: each participant studies the product and the associated documentation and registers detected defects. After that, the logging meeting is conducted. All participants meet together in order to log the found defects and try to find new defects during the meeting. The results of the logging meeting are passed to the owner of the reviewed document (usually the author) who is responsible for corrections. Finally, a check of corrections takes place, to assure that all logged defects were dealt with. Because the whole process involves several people and requires resources, it has to be planned and managed. In our experiment, only some of the abovementioned phases were covered, namely the kick-off meeting and individual checking, plus elements of planning and management.

Despite similarities, there are significant differences between reviews and inspections. An inspection can be considered as a highly formalised version of review, based on statistical process control and focusing on optimisation of the results. The formalisation of the process includes the following [3]:

- Focusing attention of inspectors on certain issues by assigning them to defined roles and by the use of checklists which contain suggestions of defects that the inspectors are supposed to search for;
- Adding the entry and exit phases during which a decision about allowing the product to enter/exit the inspection process is made. The decision is based on a defined and measurable criteria (for instance, assessing the defects density in the reviewed product);
- Imposing a strictly defined structure of the process and related quantitative controls (e.g. the checking rate: a number of pages studied per hour);
- Gathering metrics during each inspection phase and using them for a continuous improvement of the inspection process and the software development processes.

3. UML-HAZOP method

HAZOP (*Hazard and Operability Study*) method [6] is widely applied in the safety critical systems domain to detect *hazards* – dangerous events/states which may lead to harm in system environment. UML-HAZOP [5] is an adaptation of HAZOP which focuses on design defects that are present in UML models. It uses the original HAZOP approach by adopting HAZOP's key mechanisms: guideword-based constructing of checklists and systematic inspection process.

The idea of constructing checklists is rather simple (see Fig. 1). UML notation defines a number of elements used to express different aspects of modelled system: classes, use cases, associations, generalisations and so on. These elements are further described by their attributes. While defining the elements and their attributes we can introduce defects. HAZOP guidewords are used to classify such possible defects. Each guideword suggests a potential deviation from the desired state e.g. NO means complete lack of something that should be included in the model, OTHER THAN – another state, different from the correct one, PART OF – a correct state is achieved only partially, LESS – a quantitative decrease of some value, etc. A combination of a particular guideword with a particular attribute of the UML element gives a suggestion of a defect related to that element. In the example presented in Fig. 1, such an element is the association. Each association has a number of attributes: its name, multiplicity, aggregation, ordering and so on. The figure shows an example of the attribute-guideword combination: applying OTHER THAN guideword to the attribute 'aggregation' results in the suggestion of the following defect: *use of wrong aggregation type*. Not all such combinations give a sensible result. However, those which do make sense and are likely to occur in the models should be included in the checklist related to a given UML element. Such checklists are then used to define the scope of UML-HAZOP inspection.

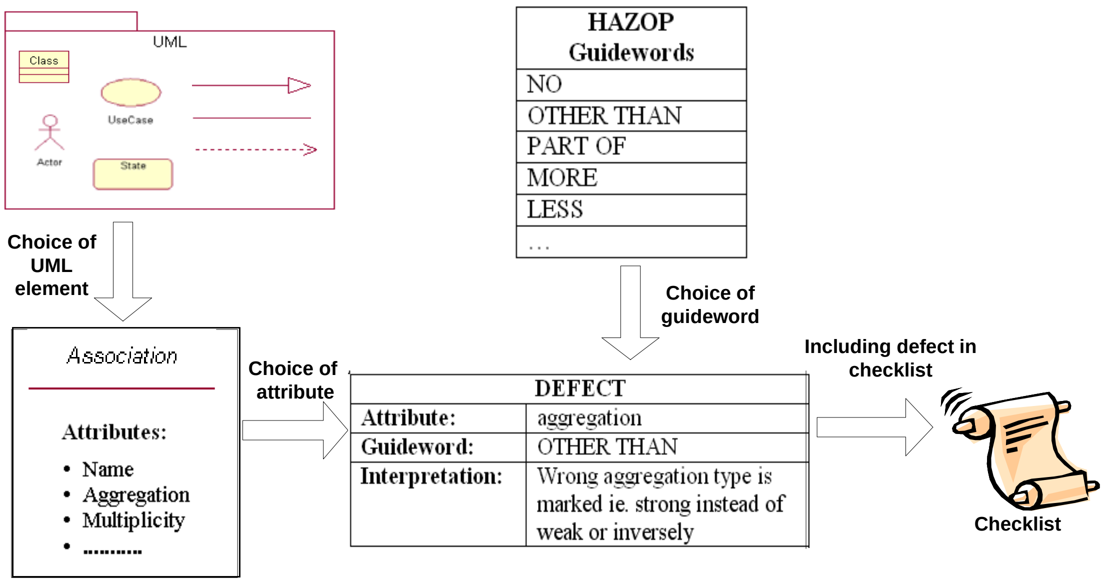

Figure 1. Construction of UML-HAZOP checklists

UML-HAZOP inspection follows a defined process. For the model under inspection, each instance of an element which occurs in the model is analysed using the checklist generated for this element type (for instance, each association is analysed using the Association Checklist). The task of the inspector is to decide if the defects suggested by the checklist are actually present and if so, to log this decision.

The present state of the UML-HAZOP method includes: the systematic method of constructing checklists, the library of checklists for selected UML elements, the definition of the inspection process and the tool supporting the method application.

The supporting tool inputs a UML model previously created by a CASE tool and automatically generates checklists for all instances of elements in the model. The checklists can be used during the inspection and the detected defects are logged right in these checklists. The system supports simultaneous work of multiple inspectors. It is based on client-server architecture and is accessible via Internet.

The detailed description of UML-HAZOP method can be found in [5] and the presentation of the supporting tool in [4].

4. The experiment

4.1. Objectives

The main objective of the experiment was to compare UML-HAZOP with a non-structured review. The comparison focused on effectiveness and efficiency of those methods. This determined the scope of the raw data collected during the experiment: the time of the individual analysis and the number and types of detected defects.

In addition to this general objective, the experiment was expected to provide answers to more specific questions:

- To which extent the lack of focusing reviewer's attention decreases the effectiveness of the review?
- If the formalisation of the process and checklists restrict the inspector?
- If the higher formalisation of inspection compensate its higher cost?

We also wanted to use this opportunity to collect subjective opinions of the participants about the usability of UML-HAZOP and the quality of support of the tool.

4.2. Scope

Both methods: UML-HAZOP and non-structured review were applied to search for defects in the same documentation. This was the documentation of a system developed as a case study for Group Project run during the software engineering courses offered to the industry. This documentation was extensively used in a number of courses and improved several times so we could make an assumption that it is of a high quality. The participants of the experiment were not familiar with those documents. We selected an UML class diagram with additional descriptions extracted from other project artefacts as the document where the participants of the experiment were supposed to search for defects.

The participants of the experiment were carefully selected. Because of the good modelling and UML skills required they were the last year students of the Software Engineering specialisation. The total group consisted of 15 students.

4.3. Implementation

The experiment took place in October 2003. Its subsequent phases are described below.

4.3.1. Preparations

In this phase a wide range of activities related to the detailed planning and preparation of the infrastructure, software tools and documentation was performed. The documentation given to each experiment participant included:

- Source documentation – it specified the objectives, context and requirements on the system being the subject of experiment. It provided a reference while deciding what is really the defect and what is not;
- Target documentation – a class diagram (with descriptions) to be checked for detects.

The source documentation was based on the system requirements specification. Some parts of the specification were removed to reduce the volume (acceptance tests, appendices) respecting the consistency of the remaining information. The extracted source document size was 21 pages in English.

The target documentation contained an UML class diagram. The diagram was slightly modified but without changing its meaning (removal of the container classes). Before giving it to the participants, the document was carefully analysed to find defects present in the diagram. The primary focus was on the defects within the scope of the UML-HAZOP checklists. A number of defects from outside the UML-HAZOP scope were detected as well and registered. However, this was not done in an exhaustive way, which turned out to have some impact on the results of the experiment. I addition, a number of new defects were injected to the diagram. Those injected defects were inside and outside the UML-HAZOP scope. The target documentation consisted of 16 pages in English and included a diagram (16 classes and 16 relationships) and 15 pages of description of its contents - a report generated from a CASE tool.

4.3.2. Training

The training took place one week before the proper part of the experiment and consisted of three parts. During the training, all participants were informed about the objectives, schedule and the way of work and were introduced to the discipline of reviews and inspections. The participants were divided into two groups: Review Group (8 persons) and UML-HAZOP Group (7 persons). The division was made in a way to assure that the groups have equal human potential (a mean value of participant's grades was used as the criterion).

The next two steps of the training were conducted separately for each group. Each group learned about the particular technique supposed to be used during the experiment. The training of UML-HAZOP Group took 45 minutes and included presentation of the method as well as its supporting tool. The participants could apply learned skills in practice by inspecting a simple example diagram.

A training of the Review Group was shorter (15 minutes) because this group did not have to master any defined method. An overall idea of review and a number of example defects in class diagrams were presented during this training.

4.3.3. Studying documentation

Studying the source documentation was separated from the individual checking phase to allow a more precise measurement of time spent exclusively for detecting and registering defects. Each participant (of both groups) received a separate printed copy of source documentation to study (excluding the target documentation). No communication between participants was allowed. A 2 hour period of time was initially planned for studying, but after 100 minutes when everybody declared that they had already finished, this phase was closed.

4.3.4. Individual checking

This phase took place right after the studying phase, except a short break between them. At the beginning the participants received the printed version of the target documentation (and they were allowed to use the source documentation as well). The task was to detect and register as many defects as possible. Only the defects in the diagram were sought for (not the ones in the description of the diagram). The work was done individually.

The Review Group could check the diagram in any way they wished, no process formalisation was imposed on them. The detected defects were logged in a Word file, in table with the following fields: defect number, location in the diagram, description (explanation of the defect) and classification ('major' or 'minor' depending on its estimated impact).

The UML-HAZOP Group used the supporting tool to register defects. The participants were instructed to focus on the defects suggested by the checklists and not to perform any additional checks. However, had they spotted any defects from outside the UML-HAZOP scope they were allowed to log them in.

The individual checking phase was limited to 140 minutes, the same for both groups. It was possible to finish it earlier though: for UML-HAZOP Group, after going through all checklists and for Review Group if for the period of 15 minutes no new defects were found. For each participant, the real time of the checking effort was registered.

4.3.5. Questionnaires

One week after the individual checking phase, all participants of the experiment were asked to fill in a short anonymous questionnaire. The questions were partially different for each of the two groups and concerned the following issues:

- Organisation of experiment (both groups) – if there were any organisational shortcomings which could affect the results (e.g. insufficient time for checking);
- UML-HAZOP Group: the method difficulty and usability;
- UML-HAZOP Group: the supporting tool usability.

4.3.6. Analysis of results

The raw data gathered directly from the experiment included: registered defects, individual checking time and opinions from questionnaires. The processing and analysis of the results comprised the following steps:

- verification of the detected defects;
- statistical processing (mean values, standard deviations);
- derivation metrics from the raw data;
- analysis of the questionnaires.

5. Results

Before presenting the results it is helpful to describe in more detail the issue of defects present in checked diagram. The overall number of (known) defects was 72. These defects can be classified according to the following criteria:

- origin: 15 injected, 57 made by the diagram developers;
- relation to the UML-HAZOP scope: 26 within scope, 46 outside scope;
- knowledge about them before the individual checking : 37 known, 35 unknown.

The third criterion requires more explanation. During verification of defects registered during the experiment, it turned out that the checked diagram contained many more defects that authors of experiment were aware of. This lack of awareness resulted from the fact that primarily check concentrated mainly of the defects within UML-HAZOP scope. UML-HAZOP Group detected almost all of these defects. However, they did not look for defects from outside this scope. These defects could be (and were) detected by participants of Review Group. The authors of the experiment assumed the presence of such defects in the diagram but their actual number exceeded expectations.

The results of experiment concerning the number of detected defects and the checking time are summarised in Table 1. These results were considered very surprising. First, in contrary to the expectations, more defects were detected by the Review Group than by UML-HAZOP Group. Second, the checking time for was much longer for Review than UML-HAZOP although the latter process was far more formalized.

Table 1. Experiment's results (per participant)

	Review		**UML-HAZOP**	
	Mean value	Standard deviation	Mean value	Standard deviation
Number of registered defects	27,8	12,0	20,4	9,4
Number of confirmed defects	20,9	10,0	13,0	8,6
Number of confirmed defects from UML-HAZOP's scope	6,1	3,5	11,6	6,7
Time of checking (minutes)	132,9	3,5	97,9	16,8

Explanation of this result requires a closer look at the experiment conditions. Participants from UML-HAZOP Group focused (as they were asked for) on defects suggested by the checklists. Checklists covered only some classes of defects (those from the scope of the method), so the inspectors did not have the opportunity to detect remaining defects, which turned out to be more numerous than expected. Relatively short checking time was caused by suggested rule of finishing inspection after going through all checklists. The conditions for Review Group participants were different: they were free to look for any kind of defects. This is why the Review Group took all the assigned time for finding defects. With such a great number of defects outside the UML-HAZOP scope they achieved a high number of detected defects. For defects within the UML-HAZOP scope however, their result was significantly worse than the result of the other group though (see the third row of Table 1). Table 1 also shows the difference between the numbers of registered defects and numbers of ones confirmed during verification – participants often reported presence of defects in elements which in fact were correct (rows 1 and 2).

Using the gathered data (number of defects, checking time) the following two metrics were computed for each of the applied techniques:

- Effectiveness – the number of confirmed defects divided by the number of all defects;
- Efficiency – the number of confirmed defects divided by the checking time (number of defects per checking hour).

The metrics are presented in Table 2. The Effectiveness metrics uses the number of all defects and can assume different values depending on if we refer to all defects in the diagram or to all defects in the diagram within the UML-HAZOP scope. For review technique it does not matter as its scope is not limited in any way, but for UML-HAZOP the difference is significant (72 versus 26). Both cases are reflected in Table 2, respectively in the row "UML-HAZOP" and "UML-HAZOP (within its scope)". It should also be noticed that an average reviewer found only about 1/3 of available defects and an average inspector half of them. In efficiency terms, review proved better – the longer checking time was overbalanced by a higher number of defects found.

Table 2. Comparison of characteristics

Effectiveness		**Efficiency**	
Review	0,29	Review	9,4
UML-HAZOP	0,18	UML-HAZOP	7,7
UML-HAZOP (within its scope)	0,5		

For UML-HAZOP a qualitative analysis of the results was also performed. The objective was to assess to what extent the results were dependent on the skill and perceptiveness of an individual. The present results suggest that the method is not free of such dependency. Some of the defects went completely undetected. In some other cases additional problems appeared: inspectors correctly identified an incorrect element of the diagram, but failed to report what exactly the defects is or reported defect in other checklist item than it was supposed to be.

The supplementary source of information about the experiment could be found in the questionnaires. Regarding the organizational issues no significant drawbacks were pointed out. The members of the UML-HAZOP Group estimated the usability of the method and the supporting tool as “rather high”.

6. Conclusions

At the first glance the comparison of the numbers of defects detected by non-structured review and UML-HAZOP inspection suggests that the former was doing better. A closer look however, reveals the fact that the majority of defects present in the checked diagram were from outside of the scope of UML-HAZOP. Within its scope UML-HAZOP inspection performed significantly better. This observation clearly suggests expanding scope of the method to include in it additional UML elements.

The experiment revealed some problems with applying UML-HAZOP by inexperienced users. Despite the fact that the participants generally considered the training as sufficient, the observed problems with correct identification and classification of defects suggest that more thorough training procedures and more communicative checklists could be helpful.

In the future we plan for improving the UML-HAZOP exploiting the conclusions from the present experiment as well as running some new experiments. They will aim at evaluating how broadening the scope of UML-HAZOP and changing structure of the inspection process influence the effectiveness and efficiency metrics. We also plan for a comparison of UML-HAZOP with a more formally defined review technique (e.g. a one focusing on verification of object models).

References

[1] Fagan, M., *Design and Code Inspections to Reduce Errors in Program Development*, IBM Systems Journal 8, 1976.

[2] Freeedman D., Weinberg G., *Handbook of Walkthroughs, Inspections, and Technical Reviews: Evaluating Programs, Projects, and Products (Third Edition)*, Dorset House, 1990.

[3] Gilb T., Graham D., *Software Inspections*, Addison-Wesley, 1993.

[4] Górski J., Jarzębowicz A., Leszczyna R., Miler J., Olszewski M., *Tool support for detecting defects in object-oriented models*, Proc. of 10th International Multi-Conference on Advanced Computer Systems, 22-24 X 2003, Międzyzdroje.

[5] Górski J., Jarzębowicz A., *Detecting defects in object-oriented diagrams using UML-HAZOP*, Foundations of Computing and Decision Sciences, Vol. 27 (2002), No 4.

[6] UK Ministry of Defence, *Defence Standard 00-58, HAZOP Studies on Systems Containing Programmable Electronics (Part 1&2)*, Issue 2, 2000.