DRAFT

Full paper published in the proceedings of

3rd World Congress for Software Quality
26-30 September 2005,
Munich, Germany

Development and validation of a HAZOP-based inspection of UML models

Janusz Górski , Aleksander Jarzębowicz

Gdańsk University of Technology,
Narutowicza 11/12, 80-952 Gdańsk, Poland
{jango, olek}@eti.pg.gda.pl

Abstract. The paper presents an inspection technique dedicated to detection of defects in UML models. The technique was developed by adoption of HAZOP – the method widely used in safety-critical systems domain. The main idea of HAZOP - application of generic guidewords to find all potential anomalies related to a given element of a model – has been applied to generate checklists for UML diagrams in a systematic way. The checklists together with the process of their application constitute the core of the UML-HAZOP inspection. The paper introduces UML-HAZOP and presents results of its validation through a series of case studies and controlled experiments.

1. Introduction

Professionally developed software has many different representations in addition to the code, for instance requirements, architecture, design, user manuals, etc. Most of them are not executable, so their verification and validation cannot be addressed by testing. They are logically dependent which results in that they are also temporally ordered (even in various forms of iterative development processes the requirements are documented before they are reflected in the design). If such a representation contains a defect which goes unnoticed, this defect will be reflected in all later representations downstream the process up to the code. If this defect is then detected (by testing the code) its correction requires changes in all the upstream representations and the related cost can increase exponentially [1]. This dramatic cost increase creates a strong motivation for finding defects as early as possible in software representations with the particular focus on those, which occur early in the development process. The process of finding such defects can be automated only to a limited extent as those representations are usually expressed in non-formal languages, often in just a natural language. Therefore, a human participation in such processes is inevitable and takes a form of various reviews of documents.

Review techniques are widely applied in software assurance since their effectiveness was proven [2]. Aiming at further optimization of review results or customization for a particular purpose, many various forms of review techniques were developed afterwards, just to enumerate a few [3, 4]: technical reviews, inspections, walkthroughs, round-robins. It should be noted that this taxonomy is not precise and often the same name is used in different meaning. Therefore, to avoid ambiguities we introduce the following definitions:

– *Technical review* – an ad-hoc form of review technique, which does not have a defined process structure and does not direct the inspector's attention during the review.

– *Inspection* – a formalized review imposing a strictly defined structure of the process and related quantitative controls (e.g. the checking rate: a number of pages studied per hour) and focusing attention of inspectors by assigning them to defined roles and by the use of checklists and measurable criteria (for instance, assessing the defects density in the reviewed product).

The application of inspection to a given project artifact (e.g. requirements specification, design model, source code module) is supported by suitable tools (checklists, process structure, metrics and criteria).

In this paper we focus on a particular inspection technique, which we propose for analyzing models [5]. We assume that those models are expressed in UML, the Unified Modeling Language [6], which is becoming increasingly popular in the software domain. An important tool supporting such inspection are checklists suggesting possible defects. To provide for systematic generation of such checklists we used HAZOP, a technique originally used to detect hazardous situations during system safety analysis. In our approach we adopted HAZOP for detection of general anomalies present in UML models, which makes it applicable to a broader range of commercially developed systems. The resulting technique is called UML-HAZOP.

In the subsequent sections we introduce UML-HAZOP in more detail and then we present the results of a series of case studies and experiments aiming at assessing the effectiveness and efficiency of this technique. In conclusions the research contribution is summarized and the plans for further research are outlined.

2. UML-HAZOP inspection

HAZOP (Hazard and Operability Study) is widely used in the safety-critical systems domain to seek for possible hazards, the events or states that, if occur, can have dangerous consequences. The technique is focusing on system models and analyses them by a systematic application of HAZOP *guidewords* to the model components to identify potentially dangerous situations in the system. HAZOP was first introduced in chemical industry and later was adapted for software safety [7, 8].

Table 1 gives an example set of HAZOP guidewords [7]. Application of the guidewords to the elements of the analyzed model results in suggestions of potential hazards, for instance the guideword MORE applied to the temperature of a pipe results in “too high temperature in the pipe” and applying LATE to a transition between states suggests “the transition is fired later than expected”. Such suggestions can then be analyzed if they constitute system hazards.

An important observation, which can be made with respect to HAZOP is that it comprises two phases that can be considered in separation. The first phase focuses on detecting *anomalies*, i.e. the possible (but unwanted) states or events that can occur in a considered system. The second phase concerns the assessment of those anomalies and is driven by a set of chosen criteria. One possibility is to consider the system wide safety consequences of the detected anomalies (this is what the ‘classical’ HAZOP is all about) but there are other possibilities, like for instance seeking for security consequences [9]. We can go even further and adopt HAZOP to an arbitrary analytical objective just by changing the criteria applied while considering the detected anomalies.

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.

We have exploited the above idea and adopted HAZOP in two respects:

– to analyze UML models that are developed early in the software lifecycle,

– to analyze if the detected anomalies can have negative consequences downstream the development process (we call them defects). The defects of particular interest were design defects (wrong mapping of real world concepts into a model), violation of "good practices" (for instance too much responsibility assigned to a class, too many levels of inheritance) or violation of some system attributes, like performance or security.

Our primary focus was on class diagrams of UML. Each application of a HAZOP guideword to an UML entity or attribute results in a suggestion of a model anomaly. Such candidate anomalies are subjected to a preliminary analysis with respect to their credibility and are eventually inserted in the checklists. Some examples of such checklists are given in tables 2-4.

Table 2. HAZOP table for element ClassSet

CLASS SET		
Attribute	**Guideword**	**Interpretation**
Class Set	PART OF	The diagram does not include a class or a number of classes that should be present.

Table 3. HAZOP table for element Association

ASSOCIATION		
Attribute	**Guideword**	**Interpretation**
Association	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.
Association. Name	NO	The association has no name though it should be named.
	OTHER THAN	The name of association is wrong and does not represent the meaning of the association. The name should be changed or completely removed.
AssociationEnd. Multiplicity	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).
AssociationEnd. Aggregation	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 aggregation.
	LESS	The aggregation doesn't include some classes (present on the diagram or not) that should be aggregated.
AssociationEnd. Name	NO	The side of the association has no name though it should be named.
	OTHER THAN	The name of association end is wrong or superfluous. The name should be changed or completely removed.
AssociationEnd. Ordering	NO	No ordering though it should take place.
	AS WELL AS	The ordering that should not take place is marked.

Table 4. HAZOP table for element Class

CLASS		
Attribute	**Guideword**	**Interpretation**
Class	AS WELL AS (1)	The class should not be included in a diagram and it is necessary to remove it.
	AS WELL AS (2)	The class is too complex and should be split into two or more classes.
	PART OF	The class is uniform with other class present on a diagram. These classes should be merged.
Class.Name	OTHER THAN	The name of the class does not reflect its meaning. Another name should be used.
Class. AttributeSet	PART OF	The class misses an attribute or a number of attributes. Necessary attributes should be added.
Class. Attribute	AS WELL AS	The attribute is superfluous and should be removed from this class.
	OTHER THAN	The name of the attribute does not reflect its meaning. Another name should be used.
Class. OperationSet	PART OF	One or more operations are missing and should be added.
Class. Operation	AS WELL AS	The operation is superfluous and should be removed.
	OTHER THAN	The name of the operation does not reflect its meaning. Another name should be used

The checklists comprise the main asset of the UML-HAZOP inspection. The other components of UML-HAZOP include the process of inspection [5] and a supporting software tool [10]. The process, controlled by inspection leader, is similar to classical inspection process with exclusion of logging meeting and brainstorming session; defects found by inspectors are passed to author of the model who assesses them and if necessary introduces corrections to the model. The tool imports UML models from the Rational Rose CASE tool and automatically generates checklists for all model elements.

3. Validation: Case Studies

UML-HAZOP was validated in a number of case studies during which the technique was applied to UML models developed in real software projects. The metrics collected included model size, number of detected defects and the effort spent.

The four case studies are briefly described below.

- CS-1: A telecom billing system developed at the university for educational purposes. The system was already in use for several years.
- CS-2: A subsystem of a management support system developed by a software house. A retrospective analysis (the subsystem was already implemented) was conducted.
- CS-3: Another subsystem of the same system as in CS-2. This time, however, the inspection was integrated with development and its results were fed back into the process. Before UML-HAZOP inspection the model was subjected to a quality assurance process.
- CS-4: Selected models developed during a EU 5th FP project aiming at providing IT support for healthcare application [11]. UML-HAZOP was conducted for models describing essential concepts and solutions of the project.

The results of all case studies are summarized in Table 5. The number of defects concerns only those defects that were confirmed during further analyses with participation of software developers. The total effort includes studying the project documentation, checking models and the assessment of the detected anomalies with participation of system developers.

Table 5. Case studies results

Case study number	Size of inspected models	Number of defects found	Total effort (man-hours)
CS-1	17 classes, 14 relationships	8	12
CS-2	55 classes, 79 relationships	80	45
CS-3	27 classes, 34 relationships	39	21
CS-4	39 classes, 42 relationships	81	not available

4. Validation: Experiments

To learn more about UML-HAZOP we have also arranged for two experiments aiming at comparison of UML-HAZOP with less structured reviews (more details can be found in [12]).

4.1. Experiment 1

The participants were 15 last year students of Informatics divided into two groups. Then one group received a brief training in UML-HAZOP whereas the other was trained in unstructured reviews.

Both groups received the same model to be analyzed together with the supporting documentation. One group was asked to apply UML-HAZOP for detecting defects while the other was told to conduct the unstructured review. All participants were working individually. The raw data gathered from the experiment included the detected defects and the analysis time by each participant. The following metrics were computed using the data:

- Effectiveness - the number of defects found divided by the number of all defects present[1],
- Efficiency - the number of found defects divided by the checking time (number of defects per checking hour).

As the result, the unstructured review found more defects than the UML-HAZOP. A closer analysis showed that the reason was the limited scope of checklists used, not covering most of defects present in analysed diagram. This was a motivation to extend the UML-HAZOP scope to cover not only the relationships (as was inherited from original HAZOP) but the other UML elements as well.

4.2. Experiment 2

The first experiment provided some information that was later used to improve UML-HAZOP, particularly by extending its scope and improving the clarity and unambiguity of the checklists.

The second experiment involved a different group of 17 students who analyzed the same models as were used in the first experiment. The primary focus was on verifying the defect coverage of the modified UML-HAZOP. In addition to the data collected in the first experiment we also registered the exact time of defect detection.

The results of UML-HAZOP were much better comparing to the results from first experiment and slightly better than those from the unstructured review. The new checklists proved to be complete – there were no defects discovered outside the scope of the checklists. However, the expanded scope resulted in bigger checklists and this required more time than had been allocated. As the result only 65% of the checklists were examined during the inspection. It leaves open the question of how many defects would have been detected if enough time were available. The answer to this can be formulated if we take into account that the detection rate was almost constant throughout the whole experiment, what is presented in Fig.1 (which is quite obvious if we observe that the inspectors were just going through the pre-prepared checklists).

Then by extrapolation we can postulate that if the checklists were run to the end, the number of detected defects would be increased by almost 50% (this is illustrated in Fig. 2).

[1] This was the number of all different defects discovered during the analysis, including also some defects that were already known before.

4.3. Experimental results

The following figures summarize the overall results of both experiments with respect to the effectiveness and efficiency of the analysed techniques.

Fig. 2 presents the effectiveness results from both experiments. We can see that in the first experiment the full scope (measured with respect to all known defects) effectiveness of UML-HAZOP was lower than for the review although with respect to the limited scope (measured with respect to the defects addressed by UML-HAZOP checklists). UML-HAZOP was doing significantly better than the unstructured review. In the second experiment the effectiveness of UML-HAZOP improved and would probably have been significantly better if the checklists were analysed to the very end (which is depicted by the last stripe).

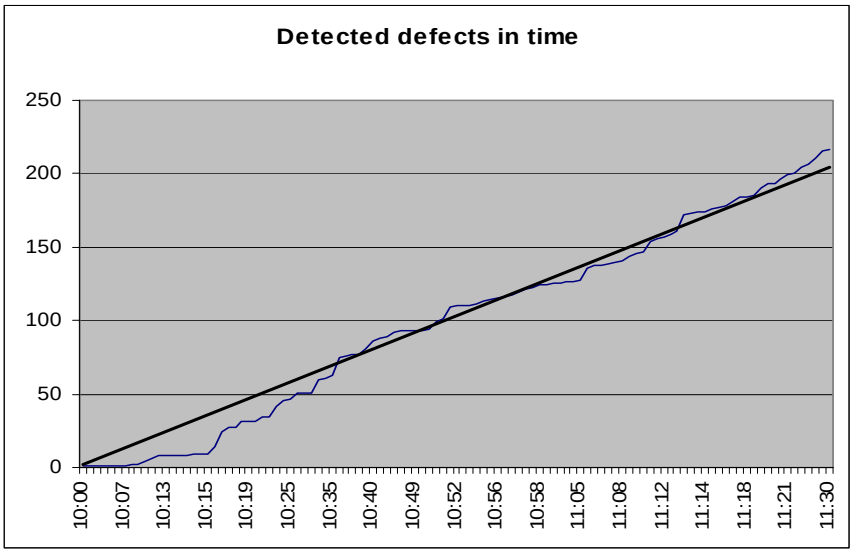

Figure 1. UML-HAZOP: the detected defects in time.

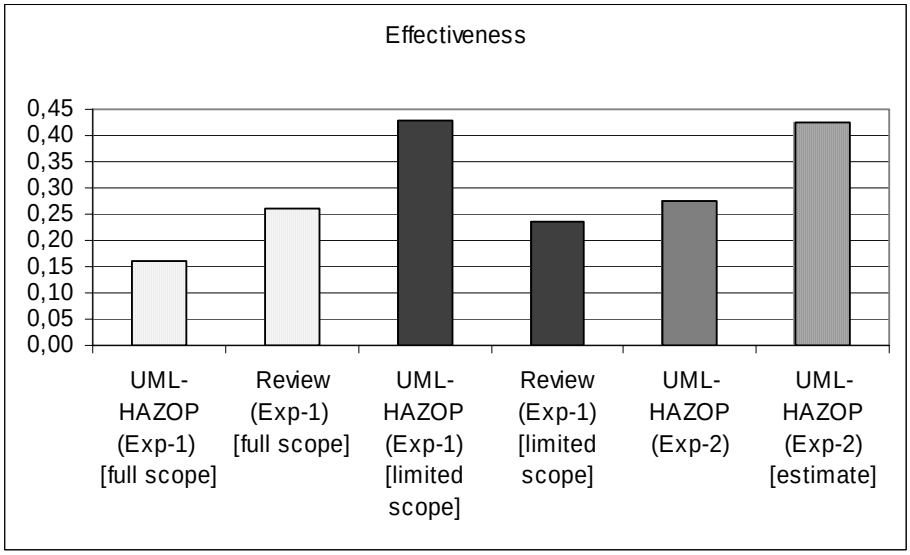

Figure 2. Comparison of effectiveness

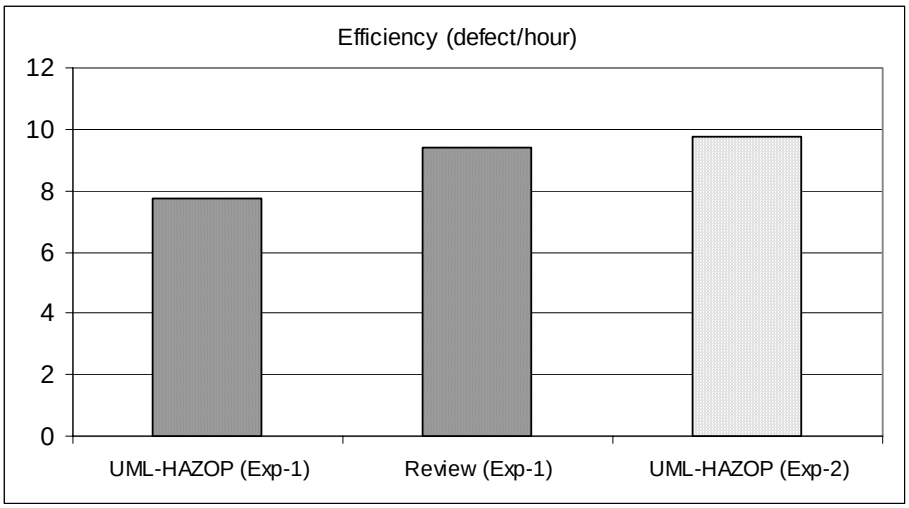

Figure 3. Comparison of efficiency

Fig. 3 presents the efficiency results obtained from the experiments. The lower efficiency of UML-HAZOP results from the limited scope of the checklists. This has then been improved in the second experiment. Assuming the linear increase of the detected defects (see Fig. 1) we can expect that the number obtained in the second experiment is representative for the UML-HAZOP inspections.

5. Conclusions

Software inspection is one of the industry best practices to deliver quality software. The inspections that focus on the software representations that are produced early in the lifecycle are of particular interest as they prevent the exponential growth of defect repair costs.

In the article we introduced a new technique of software inspections. Its distinguishing feature is that it focuses on the UML models that are produced early during software development. Another distinguishing feature is that the method is based on checklists that can be generated in a systematic way using a set of HAZOP guidewords. This process of checklists generation has been automated if the UML models are imported from a common CASE tool.

The validation experiments and case studies delivered promising results demonstrating that the effectiveness of UML-HAZOP can be significantly better then the unstructured review if the scope of the checklists is properly calibrated. The method has a constant defect detection rate (see Fig. 1), which differs from the unstructured review as symbolically illustrated in Fig. 4. If applied in short time the method can detect less defects than the unstructured review (the part on the left hand side of the crossing point in Fig. 4) whereas if the time of inspection is long enough it delivers significantly better results.

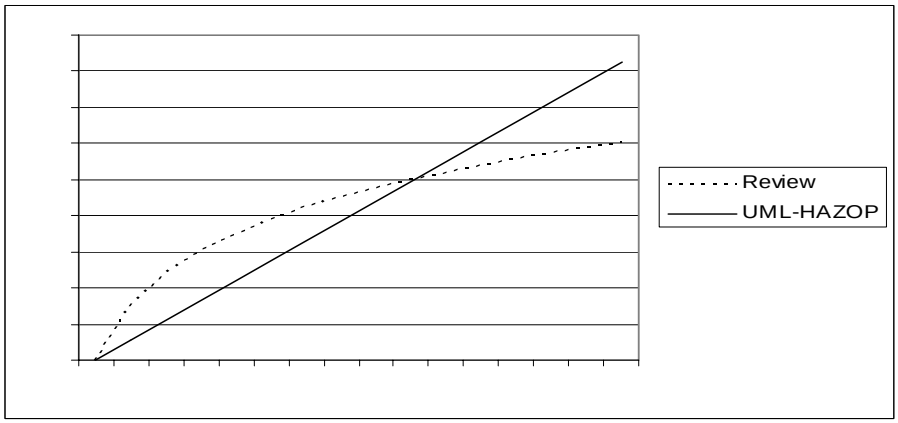

Figure 4. Number of detected defects in time

In the nearest future we plan for further experiments aiming at identification of an optimal time allocated to UML-HAZOP (as a function of the length of the checklists) and at better identification of the crossing between

the lines shown in Fig. 4 in order to learn more about the resources needed to exploit the advantages of the proposed method.

References

[1] Boehm B. W., *Software engineering economics*, Prentice-Hall, Englewood Cliffs, 1981

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

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

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

[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] Object Management Group, *OMG Unified Modeling Language Specification, Version 1.5,* March 2003.

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

[8] Redmill F., Chudleigh M., Catmur J., *System Safety: HAZOP and Software HAZOP*, J. Wiley & Sons, 1999

[9] 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.

[10] 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.

[11] *IST-1999-12040 DRug In Virtual Enterprise (DRIVE) D11.4 deliverable version 1.1*, January 2003

[12] Jarzębowicz A., Górski J. *Experimental comparison of UML-HAZOP inspection and non-structured review*, Foundations of Computing and Decision Sciences, vol. 30 (2005) no. 1.