Gaps in design and tests of dependable embedded systems

Iwona Grobelna, Michał Grobelny

Abstract: The paper focuses on gaps in design and test technology for dependable embedded systems. Possible gaps in design may influence system □el□□□cie and as the result final product may not □el□□□ all requirements or some of desired properties. Test phase is also important as it may indicate even subtle errors which occurred in previous phases. The article presents possible solutions to improve the design and test technology.

Keywords: design, tests, dependable embedded systems, model checking, temporal logic

e-□ INTRODUCTION

Dependable embedded systems [1] are currently used even in the today-life domains, like in cars, planes, trains, etc. The desired properties of such systems are high reliability, availability, maintainability, safety and security. Depending on particular usage scenario some of the properties may have more important meaning than the others, but in some degree all of these mentioned properties should be ensured. Dependable embedded systems are also intensive researched by scientific communities, like by *The German Indonesian Tsunami Early Warning System* [8]. The project has to manage several subsystems and has therefore high requirements regarding the reliability.

The development of a system involves couple steps which must occur one after the other (Fig. 1). The process starts with design phase and ends with tests. Both phases are very important.

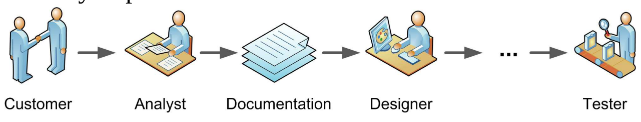

Fig. 1. Dependable embedded system development

The design phase should end with a fully complete and correct design. Possible errors in this phase may influence oncoming phases and the whole venture and generate enormous costs. The later the errors will be detected, the faster the costs of repairing them rise. Additional fact which has to be taken into account especially by dependable embedded systems is that they are supposed to be fully reliable, available, safe and secure all the time. Even a tiny error in design phase may change the total system □el□□□cie and may have tragic effects, i.e. in transport domain (like a plane crash) it may cause a catastrophe and kill hundreds of innocent people. Therefore early discovery of errors may save time, money and even people lives.

Test phase is also important as it may detect some errors which occurred in earlier phases. Tests are quite expensive for companies, but no one should even think about not performing them. The fast technology progress enforces reduction of time and resources spent on tests. Even though in the minimum scenario the most important cases should be covered.

The most frequently occurring gaps in design and test phases in development of dependable embedded systems are presented in the next sections. Additionally there are proposed solutions to solve some of the problems, taking control over one of the possible gaps in design and test phase.

2. GAPS IN DESIGN

Possible gaps in design phase which may have an influence on the dependable embedded system □el□□□cie are connected with human beings. Therefore this phase may be improved by changing organizational aspects of a company.

When starting with a new project the requirements have to be fully formulated. Unfortunately it may happen that the requirements will not be complete. This may cause that the system will operate correctly due to the requirements but it will not be fully reliable and available, because some of the important requirements may have been omitted. Another problem connected with the requirements is the situation when informal specified requirements are ambiguous. Depending on the selection of system designers, they may be correctly or incorrectly interpreted. The second case can have an enormous influence on dependable embedded system □el□□□cie. The requirements list should therefore be complete and include both functional and non-functional requirements (addressing the quality, performance, security, etc.) and be easy to understand for all members of a team.

Similar problems may occur by embedded system specification. It may happen that it will be incomplete. It may also happen that it will be incorrect. Following an incorrect specification, there will be an incorrect implementation and finally there may be an incorrect product! And either the whole development process will start again from the beginning (which causes redundant costs) or the final product will be released what may have catastrophic effects especially by dependable embedded systems.

Another problem is connected with the documentation. It is well known that a project should be fully documented. Lack of documentation may cause the overseen of some organizational and technical aspects or that the development process will last for a long period of time due to these lacks. But even if the documentation is prepared carefully by all team members, it should be ensured that there is a correct and complete flow of documentation. Documentation should be correctly distributed in the team, i.e. if a complete documentation is kept by its owner or is delivered not to the interested team members but only to some part of the team which does not need it, is just useless.

2.1. POSSIBLE SOLUTION: MODEL CHECKING FOR FORMAL VERIFICATION OF SPECIFICATION

Errors in the specification can be detected using formal verification methods [2] [5]. They can be used when the real system does not physically exist yet, preventing from possible errors on an early stage of system development. One of them is *Model Checking* [3] which will be further presented, not forgetting about *Theorem Proving* [5] as another formal verification method with all its advantages and disadvantages. Model checking allows fully automated system verification and error detection in □el□□□cie□e system specification by computer deduction tools (*model checkers*). The technique can be used to verify the whole system or just some part of it (*partial verification*). It is especially valuable by large systems where the design process lasts for a long period of time and is a complex and difficult task. Then the verification can be performed step-by-step during the design phase, each time considering only a subset of requirements.

It is important to remember that model checking can not prove that the system model is completely correct. It can just prove that the model does or does not satisfy specified requirements.

It verifies whether the requirements are satisfied in defined system model. User has to formally specify requirements for the embedded system. It is also needed to model the system using description language of a chosen model checker. Computer deduction tool automatically verifies the system and gives an answer whether the model satisfies the specification or not (model checking technique is schematically presented in Fig. 2). If not – some errors must have been detected and appropriate counterexamples are generated.

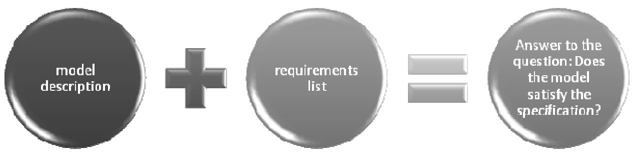
model description
requirements list
Answer to the question: Does the model satisfy the specification?

Fig. 2. Model checking technique

Discussed formal verification method can indicate some errors either in requirements specification or in model description. In the first situation some requirements may have been incorrectly formulated. In the second situation the system design itself may be incorrect. System designer has to carefully analyze received counterexamples, find the source of an error and solve the problem either by changing system model or the requirements list. Without human interaction the received results do not localize the problem, they just mention that there is a problem.

Required properties of dependable embedded system are defined using temporal logic [4] formulas. They are coded using the specification language of a chosen model checker. User has to specify as many desired properties as possible. This is due to the fact that only the defined properties will be checked.

2.2. A LITTLE BACKGROUND ON TEMPORAL LOGIC

Temporal logic [4] [6] [7] derives from modal logic and was introduced into computing science by Amir Pnuelli in 1977 when he proposed to use temporal logic in concurrent and re-active systems. Nowadays it is used as well in program specification as in its verification, synthesis and even logical programming. Using temporal logic and its operators it is possible to formally specify embedded system functionality.

Linear Temporal Logic (LTL) is a classical temporal logic and describes relations in the system specifying state sequences. In some states a given formula can be satisfied, while it can not be satisfied in the others. Basic temporal logic operators with their meanings and examples are listed in Table 1.

Table 1.
Basic temporal logic operators

Sign	Meaning	Example	Explanation
□	always	□ p	p is true in all states
◊	sometimes	◊ p	p is true in some states
o	next	o p	p is true in the next state

Temporal logic with time branches is Computation Tree Logic (CTL). Time is here presented as a tree branching out into the future with present moment as the root. Basic temporal connectors are presented in Table 2.

Table 2.
Temporal connectors

Quantifier	Explanation
Path quantifiers	
E	for some path
A	for all paths
State quantifiers	
F	for some state
G	for all states

Path quantifiers are characteristic for branching time temporal logic and are meant for paths beginning from a given state. State quantifiers are for states in that path. Combining path and state quantifiers it is possible to describe complex dependencies, like for example:

- Afp — In every path there is some state where formula p is true

- Efp — In some path there is some state where formula p is true
- Agp — In every path in every state formula p is true
- Egp — In some path in every state formula p is true

Model checking technique uses temporal logic and allows to formally verify specifications of dependable embedded systems.

3. Gaps in tests

Test phase should find all existing errors so that the final product is as much reliable as possible. However it is a difficult task. Embedded systems require huge amount of rigorous tests which ensure appropriate quality [9]. The main problem is the fact that exhausting testing is often impossible. There are white-box (structural), black-box (functional), module, and integration tests necessary to prove that designed product □el□□□c previously assumed requirements. It may not be possible to check all configurations of input data with all configurations of the parameters of the surrounding environment. This indicates possible gaps in the test phase which may cause that some errors which could be handled will not be found.

One of them is the situation where tests do not cover all important fields and technical aspects. Often tester concentrates on particular field of dependable embedded system usage and does not foresee that something other can happen. Tester teams should include innovative and creative testers which foresee even the so-called *not-possible-to-happen-situations*. This is especially important by embedded systems which have to be fully reliable [9]. Test quality depends though on the appropriate selection of test cases.

In many industrial projects prototypes are tested either by designers or engineers strongly connected with particular project. Frequent reason is cost minimization and short development time. Nowadays the very fast evolution of technology enforces short term of products design and development. Time is often to short to □el□□□ all test cases. To reduce the test time companies may assign designers to testing phase. It has the economical advantages that they already know the product and its desired functionality, so that they do not waste time on getting to know the documentation. However in many situations designers are confident or just assume that something operates correctly and do not concentrate on obvious system properties. Unfortunately, even in obvious properties there may be a tiny problem which would be easy found during a simple test, but someone has to perform it.

Not only manual functional testing is taken into account. Automated testing plays a strong role in latest projects. Considered are here software as well as hardware tests. In test environment an appropriate preparation of such procedure is even more important. Let us assume that there is an error in test procedure, which omits one of key system properties / features. Such test can false ensure the system quality which can have tragic results. Therefore the design of test environment and automated test procedures must be prepared even more carefully than tested system. That means that sometimes test procedure preparation can last longer than the product design phase. Moreover, the cost of tests can highly increase. Here appears a possible huge gap. Currently cost reductions are popular, and following further with such reasoning, the reduction of test costs is considered. Therefore nowadays there is a trend in turning end-user into a tester which ends with purchasing partly tested products to the market.

3.1. Possible solution: Independent tester teams

Dependable embedded systems should be carefully tested so that as many errors as possible are found. The dreamed test phase indicates all errors, but in reality some of them may be easily overseen. Testing phase should involve additionally independent tester teams (Fig. 3) – one team with engineers and the second with future ordinary users.

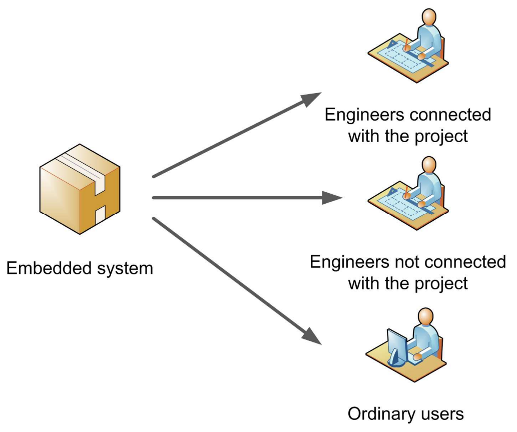
Engineers connected with the project
Embedded system
Engineers not connected with the project
Ordinary users

Fig. 3. Proposed tester teams

Engineers understand how everything operates and may concentrate on more technical aspects, like i.e. representation of data or imperfect randomness. It is important that no one of tester team took part in product design or development. Then they have fresh look and may spend much energy on testing.

Product prototypes should also be tested by ordinary users who the product is addressed to. They concentrate on its functionality and test both the desired properties and the situations often not taken into consideration by engineers. They may i.e. push multiple times the same button or the same combination of buttons to force some reaction of dependable embedded system. Sometimes it is based on lack of experience in using the device. Such □el□□□cie can help in discovering bugs of the device caused by not foreseen user □el□□□cie or invalid data input. Designer who is testing self-made device often can not know all possible □el□□□cie of an not experienced simple end-user. Therefore some part of test has to be realized with testers outside of a project.

4. Conclusion

As shortly presented in the previous sections the design and test technology of dependable embedded systems involve many traps and there are numerous places where something can go wrong.

Human being is always a possible gap in design and test technology. Both phases can not be performed without people but this fact makes it also possible to control the gaps. People involved in the dependable embedded system development should consider potential possible situations and think one more time, before they do anything. The technology itself can not provide high product quality, it can support it, but intelligent human interaction is here always needed.

There exists an assumption that there is no technical system which is 100% reliable and which has no errors. However, the number of them should be minimized so that the reliability, availability and maintainability rises. Fault-tolerant systems have such advantage that they can try to keep the reliability of computation even in the presence of faults. This is achieved mainly by redundancy of data, instructions, software, hardware and time. Dependable embedded system should operate correctly over time, and even if it is not hundred percent correct or some unpredicted circumstances occur then the potential effect of a failure should be avoided or just minimized.

REFERENCES

[1] A. Avizienis, J. Laprie, and B. Randell, *Fundamental Concepts of Computer System Dependability*, IARP/IEEE-RAS Workshop on Robot Dependability: Technological Challenge of Dependable Robots in Human Environments, Seoul, Korea, May 2001

[2] E.M. Clarke, J.M. Wing et al., *Formal methods: State of the Art and Future Directions*, ACM Computing Surveys, Vol. 28, No. 4, 1996

[3] E.A. Emerson, *The Beginning of Model Checking: A Personal Perspective*, Lecture Notes in Computer Science, 25 Years of Model Checking: History, Achievements, Perspectives, 2008, pp. 27 – 45

[4] M. Huth, M. Ryan, *Logic in Computer Science. Modelling and Reasoning about Systems*, Cambridge University Press 2004

[5] C. Kern, M.R. Greenstreet, *Formal Verification in Hardware Design: A Survey*, ACM Transactions on Design Automation of Electronic Systems (TODAES), Vol. 4, Issue 2, April 1999, pp. 123 – 193

[6] L. Lamport, *"Sometime" is sometimes "not never"*, On the Temporal Logic of Programs, Proceedings of the Seventh ACM Symposium on Principles of Programming Languages, ACM SIGACT-SIGPLAN 1980, pp. 174 – 185

[7] M.V. Rice, M.Y. Vardi, *Branching vs. Linear Time: Final Showdown*, Proceedings of the 2001 Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2001, LNCS Volume 2031, Springer-Verlag 2001, pp. 1 – 22

[8] B.Schnor, *Dependable and Fault Tolerant Distributed Systems*, DAAD-DEDIS Summer Academy, Cottbus 5.9.2008

[9] Wei-Tek Tsai, Lian Yu, Feng Zhu, Paul, R., *Rapid embedded system testing using verification patterns*, Software, IEEE Volume 22, Issue 4, July-Aug. 2005 pp. 68 – 75

mgr inż. Iwona Grobelna
Uniwersytet Zielonogórski
Wydział Elektrotechniki, Informatyki i Telekomunikacji
Instytut Informatyki i Elektroniki

ul. Podgórna 50
65-246 Zielona Góra

e-mail: i.grobelna@iie.uz.zgora.pl

mgr inż. Michał Grobelny
Uniwersytet Zielonogórski
Wydział Elektrotechniki, Informatyki i Telekomunikacji

ul. Podgórna 50
65-246 Zielona Góra

e-mail: m.grobelny@weit.uz.zgora.pl