Mälardalen University Press Licentiate Theses
No. 238

AN ALARP STOP-TEST DECISION FOR THE WORST-CASE TIMING CHARACTERISTICS OF SAFETY-CRITICAL SYSTEMS

Mahnaz Malekzadeh

2016

School of Innovation, Design and Engineering

ISBN 978-91-7485-279-0
ISSN 1651-9256
Printed by Arkitektkopia, Västerås, Sweden

Distribution: Mälardalen University Press

Populärvetenskaplig sammanfattning

Säkerhetskritiska system är system där fel kan leda till en förlust av människoliv eller katastrofala skador på miljön. Tid är ett viktigt begrepp i säkerhetskritiska system, eftersom dessa system måste kunna reagera på stimuli inom ett visst tidsintervall för att förhindra katastrofer. Det är alltså mycket viktigt att testa dessa system för såväl funktion som timing. Eftersom uttömmande testning i praktiken är omöjlig, är det viktigt att veta när det är rimligt att avbryta testningen. Avbryter man för tidigt så kan eventuella fel kvarstå i systemet. Avbryter man för sent så blir det förlust av både tid och resurser. Hittills har forskning fokuserat på utformning och tillämpning av olika teststrategier, utan att beaktat när det är rimligt att sluta testa.

I den första delen av denna avhandling, föreslår vi en ny metod för att fatta beslut om när det är rimligt att sluta testa i samband med test av svarstid för att hitta den längsta svarstiden för en viss process i systemet. Vi presenterar en konvergensalgoritm som informerar testaren om ytterligare tester skulle kunna ge betydande nya insikter i systemets timing. Om detta inte är fallet föreslår algoritmen att testning avbryts. Vi har utvärderat algoritmen utgående från principen *As Low As Reasonably Practicable (ALARP)* som är en viktigt utgångspunkt i flertalet säkerhetsstandarder och som enkelt uttryckt innebär att fortsatt arbete (testning i detta fall) ska ske så länge ansträngningen står i rimlig proportion till den nytta arbetet ger.

Vi har även trimmat konvergensalgoritmens parametrar, med hjälp av metoden *Design of Experiment (DoE)* för att förbättra algoritmens prestanda och skalbarhet.

Slutligen har vi utvärderat konvergensalgoritmens robusthet, vilket i detta sammanhang definieras som att algoritmen ger giltiga beslut att avbryta testning för en bred uppsättning problem; till exempel att algoritmen ger giltiga beslut för system som består av 10 till 50 processer med perioder som varierar mellan 200 μs till 1000 μs. För utvärdering av robusthet, identifierar vi vilka parametrar som algoritmen är mest känslig för, dvs som om de varieras ger störst påverkan på algoritmens resultat. Sedan använder vi dessa variationer för att stresstesta algoritmen och för att anpassa den så att algoritmen blir robust over erforderliga intervaller.

Abstract

Safety-critical systems are those in which failure can lead to loss of people's lives, or catastrophic damage to the environment. Timeliness is an important requirement in safety-critical systems, which relates to the notion of *response time*, i.e., the time a system takes to respond to stimuli from the environment. If the response time exceeds a specified time interval, a catastrophe might occur.

Stringent timing requirements make *testing* a necessary and important process with which not only the correct system functionality has to be verified but also the system timing behaviour. However, a key issue for testers is to determine when to stop testing, as stopping too early may result in defects remaining in the system, or a catastrophe due to high severity level of undiscovered defects; and stopping too late will result in waste of time and resources. To date, researchers and practitioners have mainly focused on the design and application of diverse testing strategies, leaving the critical stop-test decision a largely open issue, especially with respect to timeliness.

In the first part of this thesis, we propose a novel approach to make a stop-test decision in the context of testing the worst-case timing characteristics of systems. More specifically, we propose a convergence algorithm that informs the tester whether further testing would reveal significant new insight into the timing behaviour of the system, and if not, it suggests testing to be stopped. The convergence algorithm looks into the observed response times achieved by testing, and examines whether the *Maximum Observed Response Time (MORT)* has recently increased,

and when this is no longer the case, it investigates if the distribution of response times has changed significantly. When no significant new information about the system is revealed during a given period of time it is concluded, with some statistical confidence, that more testing of the same nature is not going to be useful. However, some other testing techniques may still achieve significant new findings.

Furthermore, the convergence algorithm is evaluated based on the *As Low As Reasonably Practicable (ALARP)* principle which is an underpinning concept in most safety standards. ALARP involves weighting benefit against the associated cost. In order to evaluate the convergence algorithm, it is shown that the sacrifice, here testing time, would be grossly disproportionate compared to the benefit attained, which in this context is any further significant increase in the MORT after stopping the test.

Our algorithm includes a set of tunable parameters. The second part of this work is to improve the algorithm performance and scalability by (i) determine whether the parameters do affect the algorithm, and (ii) identify and tune the most influential parameters. This process is based on the *Design of Experiment (DoE)* approach.

Moreover, the algorithm is required to be *robust*, i.e., provides valid stop-test decisions across a required range of task sets. For example, if the system's number of tasks varies from 10 to 50 tasks and the tasks' periods change from the range [200 μs, 400 μs] to the range [200 μs, 1000 μs], the algorithm performance would not be adversely affected. In order to achieve robustness, firstly, the task set parameters that influence the algorithm performance the most are identified by the *Analysis of Variance (ANOVA)* approach. Secondly, it is examined whether the algorithm is sound over some required ranges of those parameters, and if not, the situations in which the algorithm's performance significantly degrades are identified. In our future work we will use these situations to stress test the algorithm and to tune it so that it becomes robust across the required ranges.

Our experimental evaluation, so far, has shown that the convergence algorithm provides good results for most considered scenarios, however, there have been experiments in which the algorithm results in a prema-

ture stop-test decision, i.e., it suggests to stop even if further testing is likely to reveal new interesting information. With the aim to improve the situation, we investigate if the the statistical test used in the algorithm may have an adverse effect on it and if another test could improve the premature stop-test decision. The evidence collected indicates that the amount of premature stop-test decision can be reduced (or even eliminated), at the cost of an increased testing effort.

To Mehdi, Oranous, and Nasser

Acknowledgments

"*Be grateful for whoever comes, because each has been sent as a guide from beyond.*" Rumi

First and foremost, I would like to express my highest appreciation to my supervisory team, Prof. Dr. Iain Bate, Prof. Dr. Sasikumar Punnekkat and Prof. Dr. Hans Hansson for their immense knowledge, guidance and patience through the years of my PhD. study. This thesis would not have been possible without your continuous support. I would also like to thank all the colleagues and researchers at Mälardalen University for the great time we have shared during lectures, meetings and coffee breaks. My thanks also go to the administrative staff for always being of great help and support. Many thanks to SYNOPSIS members for all our fruitful discussions and meetings. I am also very grateful for having the opportunity to share my office with Irfan, Omar, Gabriel, Husni and Filip and enjoying our ongoing research collaboration and discussions both on and off the work.

I also had the opportunity to visit the department of computer science, faculty of engineering (LTH), Lund University during my study. My gratitude goes to my supervisors as well as Prof. Dr. Per Runeson at Lund University for arranging my visit. I would also like to thank people whom I met there for the pleasant time we shared.

I am eternally grateful to my parents, Oranous and Mehdi, my elder sisters, Noushin and Nanaz, my brother, Alireza, and my husband, Nasser, whose love and support have been inexhaustible fountain of inspiration through my life.

This work was funded by SYNOPSIS project, from the Swedish Foundation for Strategic Research (SSF).

Mahnaz (Anita) Malekzadeh
Lund ⇌ Västerås, May, 2016

List of Publications

Papers Included in the Licentiate Thesis[1]

Paper A *Making an ALARP Decision of Sufficient Testing.* Mahnaz Malekzadeh, Iain Bate. In Proceedings of the 15th IEEE International Symposium on High Assurance Systems Engineering (HASE 2014), IEEE, January 2014.

Paper B *Using Design of Experiments to Optimise a Decision of Sufficient Testing.* Mahnaz Malekzadeh, Iain Bate, Sasikumar Punnekkat. In Proceedings of the 41st Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2015), IEEE, August 2015.

Paper C *Influential Nuisance Factors on a Decision of Sufficient Testing.* Mahnaz Malekzadeh, Iain Bate. In Proceedings of the 15th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2015), Springer, November 2015.

Paper D *Improving the Stop-Test Decision When Testing Data are Slow to Converge.* Mahnaz Malekzadeh, Iain Bate. MRTC Report, ISRN: MDH-MRTC-310/2016-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, August 2016.

[1] The included articles have been reformatted to comply with the licentiate layout.

Contents

I

Thesis

Chapter 1

Introduction

Safety-critical systems are those in which failure can lead to loss of people's lives or catastrophic damage to the environment, e.g., aircraft flight control, nuclear systems, automobiles and medical devices. Timeliness is an important concept in a safety-critical system and relates to the notion of *response time*, which is the time a system takes to respond to stimuli from the environment. If the response time exceeds a specific time, called *deadline*, a catastrophe might occur.

Stringent timing requirements make *testing* an important process with which not only the correct system functionality must to be tested but also its timing characteristics. However, a key issue for testers is to determine when to stop testing, as stopping too early may result in defects remaining in the system, or a catastrophe due to high severity level of undiscovered defects; and stopping too late will result in waste of time and resources (Figure 1.1). Although the stop-test decision is extremely important and a wrong decision may result in severe consequences, researchers and practitioners have mainly focused on the design and application of diverse testing strategies, leaving the stop-test decision largely an open research issue. In this thesis, we propose a novel approach to make a stop-test decision in the context of testing for the worst-case timing characteristics of systems where our testing is a *black box* approach [1], i.e., without knowledge of the internal structure of

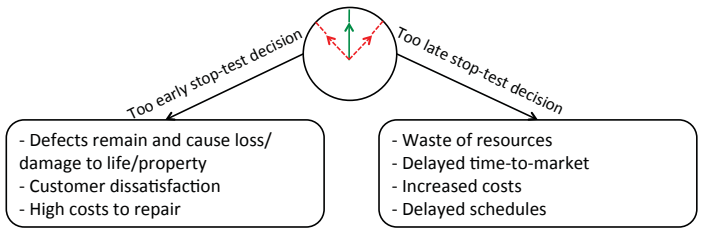

Figure 1.1: Consequences of untimely stop-test decisions

the software under test. Furthermore, the analysis considered this far is based on random testing, i.e., we randomly select input data to the software from the input domain. Random testing can provide confidence over the operational usage of the system. Moreover, it helps us to define system behaviours, e.g., worst-case timing. Figure 1.2 summarises our work focus, which is at system level, to test the timing characteristics using the black box approach.

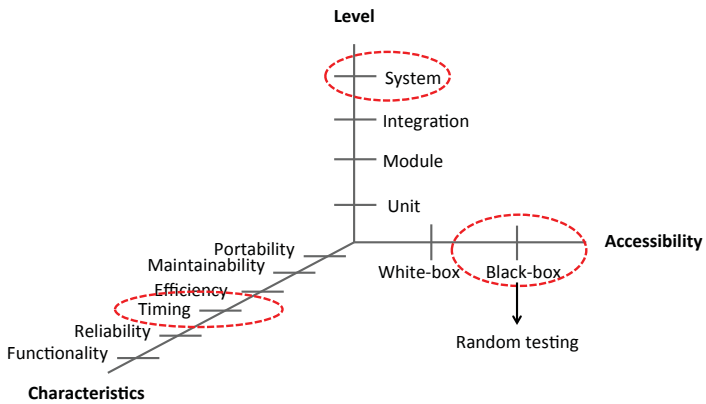

Figure 1.2: Testing: Level, accessibility, characteristics

More specifically, our method is based on a *convergence algorithm* that informs the tester whether further testing would reveal significant new information about timing behaviour, and if not, it suggests testing to be stopped. The main issues concerning the stop-test decision, addressed

in this thesis, are as follows.

Q1 When testing the system for the worst-case timing behaviour, how can we judge whether we have gained enough observations and there is no significant gain by further testing?

Q2 How can we evaluate the performance of the approach?

Q3 How can we improve the performance of the approach?

Q4 Which controlled data (input) are the most influential on the observed data (output) and, subsequently, on the convergence metric?

To address (Q1), our proposed algorithm is applied on testing data, i.e., response times of system's tasks and (i) determines whether the *Maximum Observed Response Time (MORT)* has recently increased and, when this is no longer the case, (ii) it investigates if the distribution of response times has significantly changed. When no significant new information about the system is revealed during a given period of time, it is concluded, with some statistical confidence, that more testing of the *same nature* is not going to result in significant new insight. However, some other testing techniques may still achieve significant new findings.

To address (Q2) and to evaluate the algorithm, we use the *As Low As Reasonably Practicable (ALARP)* principle which is an underpinning concept in many safety standards, and involves weighting benefit against the associated cost [2]. In order to evaluate the convergence algorithm, it is shown that the sacrifice would be grossly disproportionate compared to the benefits attained, i.e., any further significant increase in the observed worst-case timing, after stopping the test, needs a disproportionate amount of testing time.

To improve the convergence algorithm performance as stated in (Q3), the following steps are taken:

- The convergence algorithm includes a set of parameters which were initially tuned by means of limited trial and improvement experiments. In order to improve the algorithm performance and scalability, firstly, it is determined whether the parameters do affect the algorithm.

Secondly, the most influential parameters are identified and tuned. This process is based on the *Design of Experiment (DoE)* approach [3]. The reason to use DoE is that we can evenly explore each parameter's input domain to identify the *candidates* leading to better performance of the algorithm. The candidate selection procedure is shown is Figure 1.3.

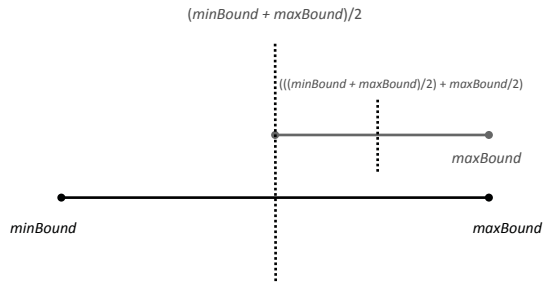

Figure 1.3: Candidate selection procedure by DoE

DoE includes two phases: *phase1* and *phase2*. As shown in Figure 1.3, at *phase1*, the two boundary values of each parameter input domain are examined (*minBound*, *maxBound*). If *maxBound* results in better performance, the range [(*minBound*+*maxBound*)/2, maxBound] will be explored further in *phase2* as it seems that better candidates fall within this range. More candidates are examined in *phase2* which are selected similar to *phase1* procedure. The first phase examines all the algorithm parameters using two candidates while the second phase only focuses on the influential parameters with more than two candidates. This two-phase DoE approach improves the scalability, i.e., more parameters with less candidates at *phase1* and less parameters with more candidates at *phase2* are examined which avoids combinatorial explosion.

- In order to make a stop-test decision, the convergence algorithm uses a statistical test and has shown to provide good results for most considered scenarios, however, there were experiments in which the algorithm results in premature stop-test decisions, i.e., it suggests to stop even if further testing is likely to reveal new interesting information at an acceptable cost. As an alternative, we

focus on the statistical test used in the algorithm and investigate whether any of its properties may have an adverse effect on the algorithm and if a new statistical test could improve the premature stop-test decision.

Finally, to address (Q4), we take the following steps:

- We identify a set of parameters, derived from task set characteristics, which are the most influential on the convergence algorithm performance using the *Analysis of Variance (ANOVA)* approach. Then, we determine ranges of these parameters over which we would like the convergence algorithm to hold, e.g., number of tasks ranging from 10 to 50, task period ranging from [200 μs, 400 μ] to [200 μs, 1000 μs]. Then, we examine whether the algorithm is sound over these required ranges, and if not, the situations in which the algorithm's performance significantly degrades are identified.
- These situations, when the algorithm's performance degrades, will be used in our next piece of work to stress test and to tune the algorithm so that it becomes robust across the required ranges of task set parameters.

1.1 Outline

This thesis is organised in two parts. The first part summarises the research as follows: some basic concepts used throughout the thesis and the related work are presented in Chapter 2 followed by our research methodology and the thesis research goals in Chapter 3. In Chapter 4, we present the concrete thesis contributions in more details. Finally, limitations, conclusions and future work are stated in Chapter 5. The second part of the thesis consists of a collection of papers. Below follows a brief overview of these papers.

Paper A (Chapter 6) Making an ALARP Decision of Sufficient Testing, Mahnaz Malekzadeh, Iain Bate.

Abstract. *ALARP* is an important concept in many safety standards.

It helps in making a decision about how tolerable a risk is. A tolerable risk should be reduced to a point that is *As Low As Reasonably Practicable* (ALARP) which implies further risk-reduction is grossly inappropriate compared to the benefit attained.

To date work has considered the process, safety arguments, and influencing factors of how to make an ALARP decision but not shown how to make a quantified judgement for it. In this paper a method for making an ALARP judgement decision is proposed in the context of testing the worst-case timing properties of systems. The method is based around a *convergence algorithm* that informs the tester when it is believed that testing for longer will not reveal sufficiently important new findings, i.e., any significant increase in observed worst-case timing needs a disproportionate amount of testing time.

Status: Published in the 15th IEEE International Symposium on High Assurance Systems Engineering (HASE 2014), January 2014.

My Contribution: This contribution relates to (Q1) and (Q2) stated earlier in this chapter and I was the main contributer of the work. My contributions include the development and implementation of the convergence algorithm as well as its evaluation using a task set simulator. The work was performed under supervision of the co-author who contributed by constructive comments, discussions, and reviewing of the paper.

Paper B (Chapter 7) Using Design of Experiments to Optimise a Decision of Sufficient Testing, Mahnaz Malekzadeh, Iain Bate, Sasikumar Punnekkat.

Abstract. Testing of safety-critical embedded systems is an important and costly endeavor. To date researchers and practitioners have been mainly focusing on the design and application of diverse testing strategies, but leaving the test stopping criteria as an ad hoc decision and an open research issue. In our previous work, we proposed a convergence algorithm that informs the tester when the current testing strategy does not seem to be revealing new insight into the worst-case timing properties of tasks and hence should be stopped. This algorithm was shown to be successful but its trial and error tuning of parameters was an is-

sue. In this paper, we use the *Design of Experiment (DoE)* approach to optimise the algorithm's performance and to improve its scalability. During our experimental evaluations the optimised algorithm showed improved performance by achieving relatively the same results with 42% less testing cost as compared to our previous work. The algorithm also has better scalability and opens up a new path towards achieving cost effective non-functional testing of real-time embedded systems.

Status: Published in the 41st Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2015), August 2015.

My Contribution: This contribution addresses (Q3) and I was the main contributer of the work. My contributions include the convergence algorithm optimisation based on the DoE approach and implementation in MATLAB. The work was performed under supervision of the co-authors who contributed by constructive comments, discussions, and reviewing of the paper.

Paper C (Chapter 8) Influential Nuisance Factors on a Decision of Sufficient Testing, Mahnaz Malekzadeh, Iain Bate.

Abstract. Testing of safety-critical embedded systems is an important and costly endeavor. To date work has been mainly focusing on the design and application of diverse testing strategies. However, they have left an open research issue of when to stop testing a system. In our previous work, we proposed a convergence algorithm that informs the tester when the current testing strategy does not seem to be revealing new insight into the worst-case timing properties of system tasks, hence, should be stopped. This algorithm was shown to be successful while being applied across task sets having similar characteristics. For the convergence algorithm to become *robust*, it is important that it holds even if the task set characteristics here called *nuisance factors*, vary. Generally speaking, there might be either the main factors under analysis, called *design factors*, or nuisance factors that influence the performance of a process or system. Nuisance factors are not typically of interest in the context of the analysis. However, they vary from system to system and may have large effects on the performance, hence, being very important to be accounted for. Consequently, the current paper looks into a set of nuisance factors that affect our proposed convergence

algorithm performance. More specifically, it targets situations when the convergence algorithm performance significantly degrades influencing its reliability. The work systematically analyzes each nuisance factor effect using a well-known statistical method, further, derives the most influential factors.

Status: Published in the 15th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2015), November 2015.

My Contribution: This contribution relates to (Q4) and I was the main contributer of the work conducting the statistical analysis, simulations and evaluations. The work was performed under supervision of the co-author who contributed by constructive comments, discussions, and reviewing of the paper.

Paper D (Chapter 9) Improving the Stop-Test Decision When Testing Data are Slow to Converge, Mahnaz Malekzadeh, Iain Bate.

Abstract. Testing of safety-critical systems is an important and costly endeavor. To date work has been mainly focusing on the design and application of diverse testing strategies. However, they have left the important decision of "when to stop testing" as an open research issue. In our previous work, we proposed a convergence algorithm, in the context of testing the worst-case timing characteristics of a system, that informs the tester when it is concluded that testing for longer will not reveal sufficiently important new findings, hence, should be stopped.

In order to make a stop-test decision, the convergence algorithm used the *Kullback-Leibler DIVergence (KL DIV)* statistical test and was shown to provide good results for most considered scenarios, however, there were experiments in which the algorithm results in premature stop-test decisions, i.e., it suggests to stop even if further testing is likely to reveal new interesting information at an acceptable cost. As an alternative, we focus on KL DIV and investigate whether any of its properties may have an adverse effect on the algorithm and if a new test called the *Earth Mover's Distance (EMD)* could improve the premature stop-test decision. Our evaluation shows that for EMD none of the considered scenarios suggests premature stop-test decision, however, the algorithm

with EMD has in some cases a slower convergence, leading to an additional cost in terms of testing time compared to using KL DIV.

Status: Published as a technical MRTC report (ISRN: MDH-MRTC-310/2016-1-SE), August 2016.

My Contribution: This contribution relates to (Q3) in which I was the main contributer of the work. My contribution includes improving the convergence algorithm by using a new statistical test. The work was performed under supervision of the co-author who contributed by constructive comments, discussions, and reviewing of the paper.

Chapter 2

Background

This Chapter includes the description of safety-critical systems and testing as an important process in the development of such systems. It also describes timing analysis techniques to predict temporal behaviour of safety-critical systems and ends with a related work on making a stop-test decision.

2.1 Safety-Critical Real-Time Systems

A *system* is an entity that interacts with other entities, called the *environment* [4]. What the system does to implement its function is called its *behaviour* and is described by a sequence of states [4]. The system behaviour as perceived by its user is called *service* [4]. Since a service is a sequence of the system's external states, a service *failure* means that at least one or more external state of the system deviates from the expected correct service state. The deviation is called an *error* and a *fault* is the adjudged or hypothesized cause of an error [4].

A real-time system is a computer system in which the correctness of the system behavior depends not only on the computational results, but also on the time when these results are delivered [5]. Thus, timeliness is an important requirement in real-time systems which relates to the

notion of *response time*. Response time is the time a system takes to respond to stimuli from the environment and to deliver results [6]. A *deadline* is a time instant when results must be produced and is called *soft* if results have utility even after the deadline is passed, whilst it is classified as *hard* if severe consequences occur in case the deadline is missed [5]. For example, a traffic signal at a railway crossing has a hard deadline; if it does not change to red before the train arrives, an accident may occur. A real-time system with no hard deadline is called a soft real-time system and the one with at least one hard deadline is called a hard real-time system or a *safety-critical system* [5]. In other words, when the system's service deviates from correct system functionality and leads to severe consequences, the system is safety-critical, e.g., a flight control system, nuclear reactor or medical devices.

A safety-critical system usually includes a set of applications running on the target hardware. Each application has a set of tasks and a release *period* where a task is a piece of code that is to be run within a single thread of execution. A task generates a sequence of jobs to the processor which are queued and executed. Scheduling refers to determining the exact order of tasks' execution in the system and provides: (*i*) an algorithm for ordering the use of system resources and in particular the CPU, (*ii*) a means to predict the worst-case timing characteristics of the system when the scheduling algorithm is applied. The predictions can be also used to assure that the temporal requirements of the system are met.

The job execution time is the time it spends using processor resources which is likely to vary between different jobs of the same task due to sources such as path data dependencies; as well as complex hardware features, e.g., caches and pipelines. The job response time is the time between when it is released and the time it finishes which may become longer than its execution time. There are several factors that result in the job response time to be longer than its execution time. For example, consider a system with jobs in the queue being scheduled by *Deadline Monotonic Scheduling Theory (DMST)*, i.e., the shorter the task deadline, the higher priority is assigned to the jobs associated to that task to be executed [6]. The scheduling is preemptive which means tasks can interrupt one another so that the higher priority task suspends the lower priority one and runs until completion. Then, the lower priority task

resumes its execution.

Figure 2.1 shows how the execution time and response time of a medium priority job τ_{medium} can differ due to blocking and interference imposed by a lower priority job τ_{low} and a higher priority job τ_{high} respectively. At the beginning, τ_{low} prevents τ_{medium} from execution by locking a shared resource which is called priority inversion. When τ_{low} completes and the shared resource becomes available, τ_{medium} runs. However, it is suspended when τ_{high} is released and selected by the *Real-Time Operating System (RTOS)* scheduler for execution. When τ_{high} finishes, τ_{medium} resumes and completes. There are also RTOS overheads for context switches and preemptions that delay τ_{medium} execution and result in the response time to become longer than the execution time.

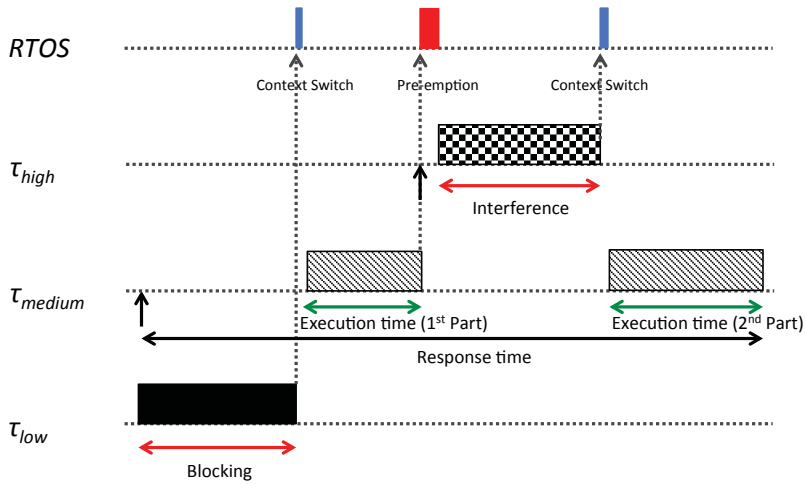

Figure 2.1: Factors that delay job τ_{medium} execution

Worst-case timing behaviour including *Worst-Case Execution Time (WCETs)* and *Worst-Case Response Times (WCRTs)* are of great importance in safety-critical systems and can be calculated using *Response Time Analysis (RTA)* techniques where the RTA technique is fed by a scheduling policy and WCETs as inputs. Moreover, strict timing requirements in such systems make testing an important and necessary process

with which not only the correct system functionality in the given environment must be tested, but also the timing characteristics.

2.1.1 Dependability, Reliability and Safety

One of the main requirements for a safety-critical system is facilitating the construction of a highly dependable system. *Dependability* is a property of the system which allows reliance to be justifiably placed on the service it delivers [6], and includes attributes as presented in Figure 2.2. Among dependability attributes, *reliability* and *safety* requirements are usually more stringent in safety-critical systems compared to other computer systems, as safety-critical systems may endanger human lives or the environment if they fail to deliver their intended service.

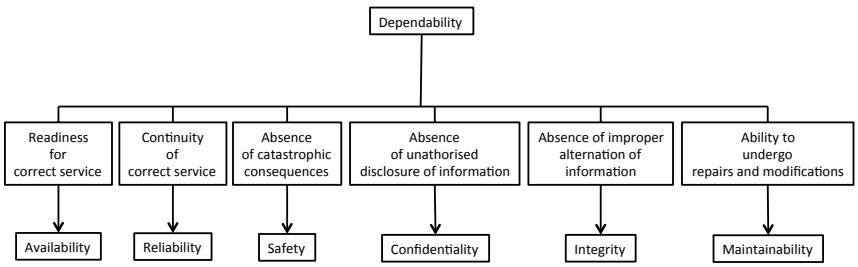

Figure 2.2: Dependability attributes [6]

Reliability is a measure of success with which a system conforms to some authoritative specification of its behaviour [6]. This specification should be complete and unambiguous including *response times* of a system as an important part. In order to create a reliable system all faults and resulting failures must be prevented. Several approaches have been proposed to estimate software reliability and can be broadly categorised as: software *Reliability Growth Models (RGMs)* and statistical models [1]. RGMs try to predict a software reliability based on its error history. Statistical models try to estimate the reliability based on the software success or failure response to random tests without correcting the discovered errors.

Safety can be defined as freedom from conditions that can result in

death, injury, occupational illness, damage to equipment, or environmental harm [6]. Software safety is considered in terms of *mishap*, which is an unplanned event or series of events that can cause death, injury and damage to equipment or environment [6]. Safety and reliability have a difference in their emphasis. Reliability is the probability that a system will deliver correct service in a given time interval, while safety is the probability that conditions resulting in mishaps do not occur regardless of whether the intended function is performed or not.

Safety is reliability regarding *critical failure modes*. In a critical failure mode, the cost of failure can be orders of magnitude more than the utility of the system during normal operation, e.g., an aircraft crash due to a failure in the flight control system. Thus, safety-critical (hard) real-time systems must have a *failure rate* with respect to critical failure modes conforming to the reliability requirement where the failure rate is the probability that a system will fail providing the correct service in a given time interval.

ALARP Principle

ALARP stands for *As Low As Reasonably Practicable* and its core is the concept of "reasonably practicable" which involves weighting a *risk* against the associated trouble, e.g., time or money to control the risk. Thus, ALARP defines the level to which we expect the risks to be controlled. A risk is the *likelihood* that a *hazard* will actually cause its adverse *effects*, including a measure of the effect. A hazard is a state or condition of a system that, combined with certain environmental conditions, could lead to accidents [7]. For example, loud noise is a hazard as it can result in hearing loss. Likelihood can be defined as probabilities and frequencies. The effect can be described in many ways, e.g., yearly (likelihood), about 1500 workers in Great Britain suffer a non-fatal injury (effect) caused by contacting a moving machinery (hazard).

The "reasonably practicable" concept originates from the British health and safety system. It is an important part of the general duties of the Health and Safety at Work etc. Act 1974 and its definition by the Court of Appeal is as follows. "*Reasonably practicable* is a narrower term than *physically possible* ... a computation must be made by the owner in which the quantum of risk is placed on one scale and the sacri-

fice involved in the measures necessary for averting the risk (whether in money, time or trouble) is placed in the other, and that, if it be shown that there is a gross disproportion between them – the risk being insignificant in relation to the sacrifice – the defendants discharge the onus on them."

In essence, in order to assure that a risk has been reduced ALARP, it has to be weighted against the sacrifice required to further reduce it. To avoid further trouble, it has to be shown that the sacrifice would be grossly disproportionate compared to the benefits of the risk reduction. For example, spending €1m to prevent bruised knees of 5 staff is grossly disproportionate while €1m to avoid an explosion capable of killing 200 people is obviously proportionate.

In order to evaluate our approach using ALARP, it is shown that the sacrifice, i.e., testing time, would be grossly disproportionate compared to the benefit gained, which is any further increase in the MORT after stopping the test.

2.2 Testing

Testing is one of the important phases in the software development procedure which can be used for two other processes called verification and validation (Figure 2.3). Verification evaluates a software system or component to examine whether products of a specified development phase meet conditions imposed at the start of that phase. Activities such as inspections and reviews of software deliverables are usually part of the verification process [1]. Validation evaluates a software system or component during or at the end of the development process in order to determine whether it meets specified requirements. Validation usually includes traditional execution-based testing, i.e., exercising the code with *test cases* [1].

A test case, according to IEEE description, includes at least the following information: (*i*) a set of inputs received from an external source, e.g., software, hardware, human, by the software under test, (*ii*) execution conditions required to run the test, e.g., a configuration of hardware device, (*iii*) expected outputs which are results to be generated by the

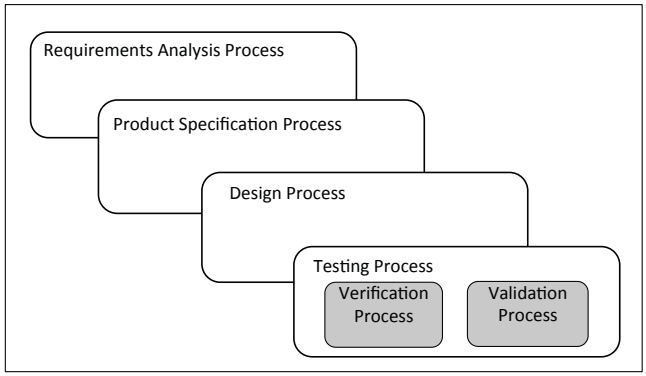

Figure 2.3: Different phases in software development process

software under test [1]. Testers use test cases to execute the software under test, with a set of input data, under a specified set of conditions to reveal faults which are defects in the system, e.g., a mistake in the code, and may manifest themselves as errors. An error is a system state in which faults become apparent, e.g., an incorrect operation when a faulty code is executed [1]. Moreover, testers compare the delivered results with the correct ones, provided by a test case, to examine whether the software has passed or failed the test. Thus, testing can be described as a process to exercise a software or a software component using a set of test cases in order to (*i*) reveal faults, and (*ii*) to evaluate the *quality* while quality relates the degree to which a system meets specified requirements. During software development process failures are observed by testers, faults are located and repaired by developers.

There are two basic strategies to design test cases: *black box* and *white box* test strategies [1], [8].

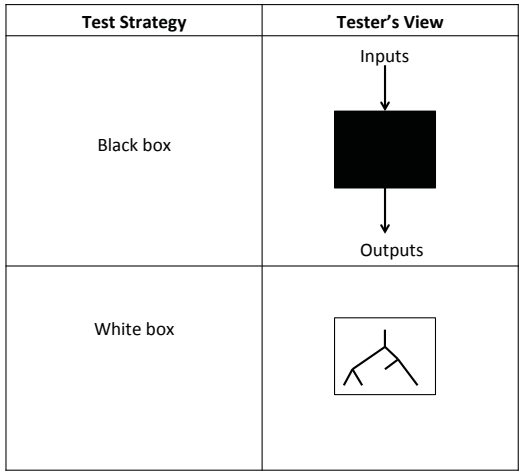

Figure 2.4: The two basic test strategies

As shown in Figure 2.4, in the black box approach, testers know what the software under analysis does. However, they have no knowledge of the inner structure of the software. The black box strategy can be applied on a wide range, from a simple module to a subsystem or a complete software system. In the white box approach, testers know the inner structure of the software, i.e., the code or a suitable pseudo code representation must be available. Testers select test cases to exercise specific internal structural elements in order to examine whether their functionality is correct, e.g., test cases could be defined to exercise all statements or branches that occur in a module. As designing, executing and analysing results in the white box strategy is very time-consuming, it is usually applied to smaller pieces of software such as a module or a member function.

2.2.1 Stop-Test Decision

Testing is an important phase in the development process of a software product. However, it is also one of the most expensive parts. Let us imagine that we are running tests on a system and at one point we decide to stop testing to save time and resources. However, there might

be still uncovered faults in the system which may result in customer dissatisfaction with our product or damages if being of high severity level. On the other hand, if we continue testing, there might be no further faults of high severity left in the system. By further testing in this situation, we would waste time and resources and risk our position in the market. Thus, an important part of testing is to decide when to stop under conditions of risk and uncertainty as it is not possible to determine with certainty that all faults have been identified. There are some criteria and methods to decide when to stop the *whole* testing process. The weakest stop-test decision is to stop testing when the project runs out of time. There are other stop-test criteria which are more based on quantitative approaches, including the following:

1. All planned tests that were developed have been executed and passed. This is also a weak decision as the clues from analysis of test cases and faults found so far may suggest that there are faults still remaining in the system which may be ignored if using this criterion in isolation.

2. All specified coverage goals have been met, e.g., when 100% branch coverage in unit testing [1] is achieved. Testing can be stopped if coverage goals specified in the test plan are met. Coverage goals such as branch coverage are based on white box testing and need knowledge of the software internals.

3. The detection of specific number of faults has been accomplished. This approach needs faults data from past releases or similar projects, i.e., the fault distribution and total faults are known for past projects which are used to estimate the number of faults and their types for the current project. This approach is risky as it assumes the current system will be built, tested and behave like the past projects. However, this is not always true and using this criterion on its own might be too risky.

4. The rates of fault detection for a certain time period have fallen below a specified level. The number of faults discovered per time unit, achieved from past projects, can be used. When the rate of fault detection of a specified severity level falls below a defined threshold, testing can be stopped, e.g., stop testing when less than 5 faults with severity level of 3 are discovered weekly.

5. Ratio of detected faults among seeded ones is favorable. Fault (defect) seeding, first proposed by [9], is based on intentionally inserting known set of defects into a program which helps to make a stop-test decision. These intentional defects can be obtained using historical defects from past releases or similar projects. The number of undiscovered seeded defects during testing gives an indication of the number of total defects (actual + seeded) remaining in the code. For example if 100 defects were seeded and 50 are found by now, it indicates that 50% actual defects are still in the code, then, testing effort would continue. There is clearly a weakness of seeding as it only covers known types of faults.

There are also more advanced stop-test criteria, e.g., based on reliability and confidence level as follows.

1. RGMs are based on observed data and accurate usage model and are mainly applied during testing. Failure rate over time can be observed as the software is being tested and be used to make a decision of when to stop testing. Operational profiles, statistical testing and RGMs can provide a stop-test decision with regard to a specified reliability goal. Reliability models help testers to predict how reliability should grow or improve over time as the software is executed, faults are identified and repaired. The software is run using an operational profile, statistical testing is applied and reliability is measured over time. When a number of reliability measurements have been reached they are compared to a selected growth model and reliability predictions are made to estimate when reliability objectives can be reached.

2. Testers may make estimations about the confidence level of the software. Confidence gives testers the likelihood that the software is fault free and is usually expressed as a percent, e.g., a 95% fault free system means the probability that the system has no faults is 0.95. Confidence levels are achieved using fault seeding technique, i.e., if S faults are planted in a system and we believe that there are N actual faults in the system, testing continues until S seeded faults are found.

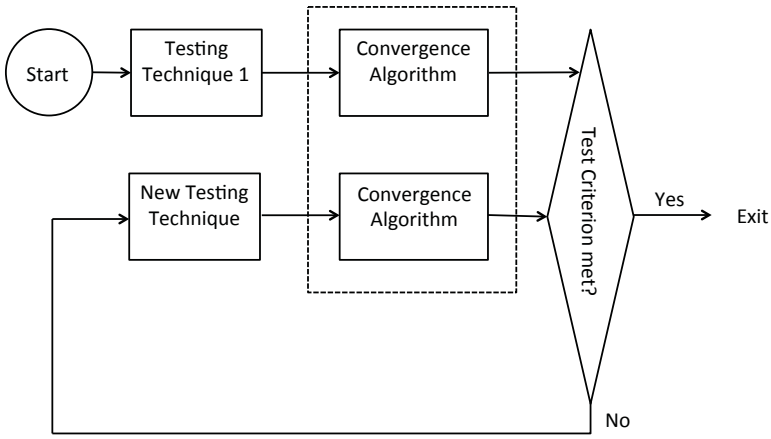

Figure 2.5: The convergence algorithm applicability within the whole testing process

Most of the stop-test criteria mentioned above need the knowledge of the system internal structure, e.g., to seed, locate, repair faults; branch coverage. These criteria help to decide when to stop the whole testing process with regard to an overall system goal, e.g., coverage or reliability. Moreover, the stop-test decisions based on fault seeding very much depend on the past projects, i.e., the known defects, and does not account for the unknown. Our method, on the other hand, is based on black box testing, hence, does not require the exact knowledge of software internals. Moreover, the decision is based on statistical analysis of testing data to decide whether we will achieve significant new information by further testing of the same nature without acquiring knowledge of the past projects. Figure 2.5 shows how our algorithm may fit into the overall testing process, i.e, our convergence algorithm can be used during the same testing technique which continually checks whether testing data has converged. Once, the convergence occurs, a new testing technique could be used. The whole testing process, however, would stop once a specified test criterion is met.

2.3 Timing Analysis Techniques

A key step in verification and validation of a safety-critical system involves reliable timing analysis in the form of WCET estimation and schedulability analysis to assure that the tasks in the system meet their timing requirements. However, variations in software execution times, which are caused by some software and hardware characteristics, make timing analysis a complex and difficult process. Sources such as varying input sets to the software, the software layout in the memory for the code and data may cause execution times variation on the software side. Whereas, features that improve the average performance by exploiting properties of execution history, e.g., caches and pipelines, may cause execution times variations on the hardware side. If all characteristics of software and the underlying hardware are thoroughly understood, accurate WCET estimation would be possible. However, the dependence of hardware and software timing characteristics on the history of execution is one of the main factors that makes acquiring knowledge needed to perform timing analysis challenging.

WCET estimation is carried out using two main approaches: *Deterministic Timing Analysis (DTA)* and *Probabilistic Timing Analysis (PTA)*. As stated in Section 2.1, WCET estimation also provides the necessary information, i.e., WCET estimates, to perform *Response-Time Analysis (RTA)*. DTA and PTA approaches are performed through two approaches: static and measurement-based methods [10]. Figure 2.6 presents different WCET estimation approaches.

2.3.1 Deterministic Timing Analysis (DTA)

Conventional static timing analysis techniques require the system under analysis to be *time deterministic*. In a time deterministic system, for a specified input set, the sequence of events that will occur is completely known, as well as the time after which the output of an event will be generated. This analysis depends on a very detailed model of the system as well as accurate information of dependence between significant events. The growing complexity in safety-critical systems, at hardware and software level, makes the extent of required knowledge, time, effort and cost to acquire and understand all the relevant details, unaffordable.

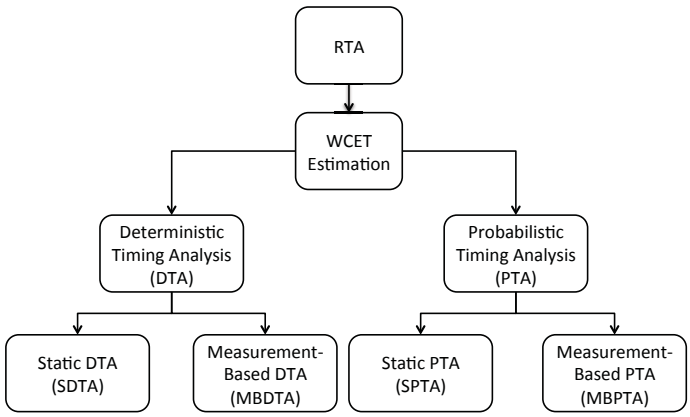

Figure 2.6: Timing analysis techniques

For example, when trying to enumerate all possible execution histories for even a simple piece of code, the large amount of state carried out by a modern processor results in combinatorial explosion. Therefore, as long as current analysis techniques are incapable of covering challenges within the increased hardware complexity, there will be a significant degradation in the quality of the resulting products.

To sum up, industry demands higher level of performance with reduced cost and power dissipation which can be provided by advanced hardware features. However, complex hardware features are difficult to deal with by DTA techniques as the amount of required information is becoming huge and complex to be analysed.

2.3.2 Probabilistic Timing Analysis (PTA)

The cost of acquiring knowledge to carry out correct timing analysis can be significantly reduced by a hardware/software architecture in which the execution time behaviour does not depend on execution history. This can be achieved by introducing *randomisation* in the timing behaviour of the hardware and software while functional behaviour is kept unchanged which is coupled with new PTA techniques [11].

A PTA model matches the reliability behaviour of the hardware

which may fail with a, though very low, probability. In fact, not only the computing system, but also other mechanical parts of a safety-critical system have a distinct probability of failure. In that sense, timing failure can be considered just another type of failure. PTA techniques aim to provide sufficiently safe WCET estimates for an application while keeping the overall system failure rate below a domain-specific threshold.

Static Probabilistic Timing Analysis (SPTA)

SPTA statically derives a-priori probabilities from a model of the system [12]. Tight WCET estimates made with SPTA have less dependence on complete knowledge than conventional DTA methods, and lack of information in analysis has less impact on the WCET estimation. In fact, WCET estimates provided by SPTA react much more smoothly to the lack of knowledge, with a gradual shift towards a maximum value as knowledge is reduced. Although the amount of information required about the hardware/software under analysis is reduced, SPTA still needs a lot of information about the internal behaviour of the architectural components.

Measurement-Based Probabilistic Timing Analysis (MBPTA)

MBPTA requires much less information rather than SPTA which makes it more attractive for industry. MBPTA derives probabilities from end-to-end runs of the software under analysis on the target platform. These runs provide data about the timing behaviour of the software when executing on the proposed hardware with randomised timing behaviour.

MBPTA derives probabilities by intensively stress testing the real system through high-coverage input data, then, recording the longest execution time, called *High Water Mark (HWM)*; and adding to it an engineering margin to make safety allowances for the unknown [10].

A MBPTA technique based on the *Extreme Value Theory (EVT)* [13] is proposed by Cucu-Grosjean et al. [11] which addresses the problems with the way EVT is applied [14], [15]; and derives *probabilistic WCET (pWCET)* estimates. The authors collect execution times achieved by end-to-end runs of the system under analysis on the target platform.

In order to apply EVT, they collect from the original distribution, the values which fit into the tail while discarding the rest of observations. Then, by means of a convergence algorithm, they assure whether enough number of observations is gained to obtain statistical significance; and for EVT to provide high quality *pWCET* estimates. Their convergence algorithm compares tails of two distributions depicted by $N_{current}$ and $N_{current} + N_{delta}$ where N determines the number of runs and decides to stop observing WCETs if the difference between the distributions' tail becomes below a given threshold for some specified consecutive iterations.

Cucu-Grosjean et al.'s work [11] is similar to ours in the sense that both are based on a convergence algorithm to decide whether enough observations from testing have been attained to predict future timing behaviour of the system. However, the two methods are different when it comes to the following aspects:

- Our observations relate to *response times* of tasks in the system without knowledge of exact task WCET, thus, is based on a *black-box* approach. While their method is based on *WCETs* obtained from end-to-end runs of the software using a *white-box* technique.
- To decide the sufficiency of observations, our method looks into the *whole distribution* of testing data while theirs just considers the *distribution tail*. In our view, it would not be sufficient to look into the distribution tail as it might happen that while there is no significant change in the distribution tail, the rest of the distribution is still significantly changing which indicates that there is a high chance of observing new and important information by further testing. On the other hand, if the whole distribution is converged, future timing behaviour is not expected to significantly change and even if there would be such a change, its cost is not justified.
- In the work by Cucu-Grosjean et al. [11], the correct use of EVT imposes that observed execution times can be described by *independent and identically distributed (i.i.d.)* random variables. This also places strong requirements on the processor hardware and how end-to-end execution times are taken such that they can be modeled by *i.i.d.* random variables. Our work is neither based on EVT

nor assumes *i.i.d.* for observed data. However, we should highlight the fact that no hardware dependency is considered in our work. Hardware dependencies are interesting to be included as a part of our future work as they affect system timing characteristics, i.e., WCETs, and may influence the convergence algorithm as well.

Chapter 3

Problem Description

This chapter starts with the definition of research, research methodology and the steps we have taken to conduct our own work. Then, the broad research problem and research questions and goals within this thesis are presented.

3.1 Research Methodology

Research according to Advanced Learner's Dictionary of Current English is defined as "a careful investigation or inquiry specially through the search for new facts in any branch of knowledge". It involves defining and refining problems, formulating hypothesis or proposed solutions; collecting, organising and evaluating data; making deductions and conclusions; finally, carefully testing the conclusions to determine whether they fit the formulated hypothesis [16]. *Research methods* can be defined as all methods and techniques being used by a researcher for conducting the research, e.g., theoretical procedures, statistical and numerical approaches, and experimental studies. Research methods help researchers to collect data and find a solution to a problem. Scientific research methods are based on not only reasoning but also explanations with respect to collected facts, measurements and observations. *Research methodology* is a way to systematically solve the research problem and includes

various steps that are generally adopted by researchers in their research problem along with the logic behind each step [16], i.e., researchers need to know not only how to apply specific research methods, but also which of these methods are suitable for their research and why; and what are the assumptions underlying different methods.

The *research process* includes necessary steps, in desired order, to effectively carry out the research. Figure 3.1 illustrates the research process to conduct the research within this thesis which is an adapted version of the process described in [16].

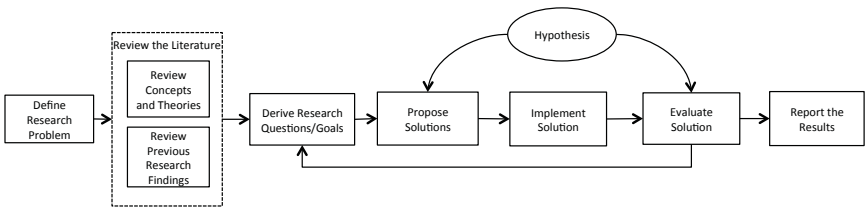

Figure 3.1: Research Process

The steps taken within our research process are as follows. Initially, in order to bridge the identified gap in the research topic, we state the problem in a broad general format and resolve its associated ambiguities. Our main research problem is when to stop testing for the timing behaviour of safety-critical systems. Then, we examine the available literature to get familiar with the research problem. Two types of literature are reviewed: (i) conceptual literature which includes related concepts and theories, (ii) empirical literature which is concerned with previous similar studies. The basic outcome of the literature review phase is the knowledge about available data and materials which enables us to specify our own research problem in a meaningful context. After this step, the research problem is refined into a couple of unambiguous research questions and goals. The next step is proposing a solution to each research question with respect to a working hypothesis. The working hypothesis is a tentative assumption made in order to derive and to test its logical consequences. It delimits the area of research and keeps us on the right track; and implies the type of data and analysis methods required. Then, the proposed solution is implemented and evaluated against the research

question. At the evaluation phase, the formulated hypothesis is tested to determine whether facts support the hypothesis or they happen to be contrary. Finally, the results are reported as a series of research papers which present our work to the research community.

3.2 Problem Statement and Research Goals

As stated in Chapter 2, timeliness is an important requirement in safety-critical systems, i.e., the system must respond to stimuli from the environment within specified time intervals, otherwise a catastrophe may occur. Stringent timing requirements make testing an important process with which not only the correct system functionality has to be tested but also the timing requirements must be catered for. However, a critical issue for testers is to determine when to stop testing as stopping too soon may result in defects remaining in the system, or even catastrophic consequences if the defects are of high severity level; and stopping too late may be waste of time and resources. Researchers and practitioners, so far, has focused on the design and application of diverse testing strategies rather than the critical decision of when to stop testing. More specifically, to the best of our knowledge, there is no previous work on making such a decision for timing characteristics of systems. Thus, the main research challenge defined in this thesis is:

How to make a stop-test decision when testing for the worst-case timing characteristics of safety-critical systems?

The main idea to tackle this challenge is to make a stop-test judgment by using testing data, i.e., MORTs of the system's tasks, achieved from running test cases on the system; then, analysing those data to make predictions about future timing behaviour where the analysis is based on statistical testing. If it is predicted, with some specified statistical confidence, that further testing of the *same nature* does not reveal significantly new information, then, it is suggested to stop testing. Eventually, the main research challenge is refined into the following research questions.

Research Question 1

When testing the system for the worst-case timing behaviour, how can we judge whether we have gained enough observations and there is no significant gain by further testing?

This research question equivalents the following question: *Which convergence metric identifies whether the observed testing data, i.e., MORTs, are revealing new information about timing characteristics of the system under test?*

More specifically, such a metric uses MORTs to determine whether they are increasing at a sufficiently fast rate; and if not, examines whether the most recent samples are significantly changing. If the metric shows that no new significant information is achieved from testing for a specific period of time, i.e., the MORTs have converged, this implies that further testing is not going to achieve significantly new insight into the timing behaviour.

The main research challenge also includes a couple of sub-challenges which are refined into further research questions as follows.

Research Question 2

The second research question relates to the ALARP principle and is defined as follows: *How could the stop-test decision be evaluated with respect to the ALARP principle?*

In terms of the work for the first research question, i.e., when to stop testing, compliance of the convergence metric to the ALARP principle is to reduce the amount of observed data needed for the analysis, as this equates to the testing time, without stopping too early. A secondary objective is to decrease the number of convergence tests needed, e.g., tasks with lower priorities are examined for convergence as their MORTs are likely to be the last to converge. Another alternative could be to not running the convergence test too early when we just start testing as testing data is unlikely to converge at the beginning.

Research Question 3

How can we improve the performance of the approach?

The third research question relates to the optimisation of the convergence algorithm such that its performance and scalability are improved. The algorithm has a set of parameters that were initially tuned using limited trial and improvement experiments. The optimisation process systematically tunes the algorithm parameters based on the DoE approach.

Moreover, the work associated with this research question includes improvement in the algorithm focusing on the statistical KL DIV test. Our experimental evaluation has revealed that KL DIV provides good results for most considered scenarios, but that there are cases when KL DIV results in a premature suggestion to stop testing, i.e., suggests to stop even if further testing is likely to reveal new interesting information. As an alternative to KL DIV, we then investigate the properties of another test (EMD).

Research Question 4

Once an appropriate convergence metric is achieved, what is the range of its validity?, i.e., for which ranges of task set characteristics is the stopping criteria still correct with respect to the ALARP principle?

In our approach, MORTs are achieved by testing the system using the black box technique, which means that no knowledge about the internal system structure is available except the system input and output, called controlled and observed data respectively. Controlled data is the task set parameters including number of tasks, period, execution time and offset. The offset is defined as the time between the task arrival and its actual release time.

Information about the most influential controlled data can be used to generate test cases with respect to the specific testing strategy being used. For example in a *stress testing* technique, we can use those values of the influential parameters that lead to circumstances under which the system timing behaviour is likely to exceed the requirements. Stress testing is when the system is tested with a load that makes it allocate

its resources in maximum amounts [1]. It can discover defects as well as weak areas, resulting in unavailability of the system's service, which may not be revealed under normal testing conditions. It is also of particular importance in safety-critical systems where unpredictable events may occur and cause input loads that exceed those in the system requirements. Stress testing can be used towards making the convergence algorithm robust.

In order to make our algorithm robust, it needs to hold across some specified ranges of a system's task set characteristics. For example, let us imagine that we have a convergence metric that makes a timely stop-test decision in a system consisting of 10 tasks with periods ranging from 200 μs to 400 μs. Then, the question could be if the number of tasks increases to 50 and periods vary from 200 μs to 1000 μs, is the metric still able to make a stop-test decision of the similar quality? Thus, it must be shown that the metric holds for some specified ranges of task set parameters, e.g., number of tasks from 10 to 50 and periods from 200 μs to 1000 μs.

Chapter 4

Thesis Contributions

This Chapter presents our contributions to answer the research questions and achieve research goals stated in Chapter 3, i.e., the first contribution proposes a convergence algorithm to decide whether further testing would reveal new significant insight into the timing behaviour of a system, and if not, suggests testing to be stopped. The convergence algorithm is tuned so that its performance and scalability are improved, where scalability relates to testing time and physical memory to attain test results. This is accomplished through the second contribution. The third contribution is an approach to identify which task set parameters affect the algorithm and whether they result in significantly degraded performance of the algorithm. The results could be used later to stress test the algorithm, i.e., through stress testing the algorithm is directed towards its worst-case performance and is tuned such that it eventually holds against the worst-case circumstances and becomes robust. Finally, the fourth contribution is to improve the algorithm when testing data tends to converge late such that premature stop-test decisions can be avoided. More details on the contributions follow in the subsections below.

4.1 A Convergence Algorithm to Make an ALARP Stop-Test Decision

This contribution relates to the first and second research questions:

Which convergence metric identifies whether the observed testing data, i.e., response times, are revealing new information about timing characteristics of the system under test?
and

How could the stop-test decision be evaluated with respect to the ALARP principle?

In order to decide when to stop testing timing characteristics of a system and to address the first research question, we propose a convergence algorithm in Paper A that analyses testing data observed from the system under test. In the following sections, firstly, the system model is described, then, the convergence algorithm and the evaluation metrics, based on ALARP, are explained.

4.1.1 System Model

Testing data, in this work, is generated using an in-house task set simulator which represents a system based on a modified version of a ball and beam example used in [17]. The plant, i.e., the ball and beam example, is shown in Figure 4.1 where the variable to be controlled (r) is the position of a free-rolling ball on a beam; and θ is the angle of the gear. The overall scheme of the simulator is shown in Figure 4.2. The simulator executes a randomly generated task set for a given duration ($SimDur$).

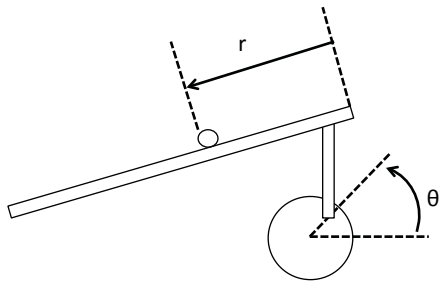

Figure 4.1: Ball and beam

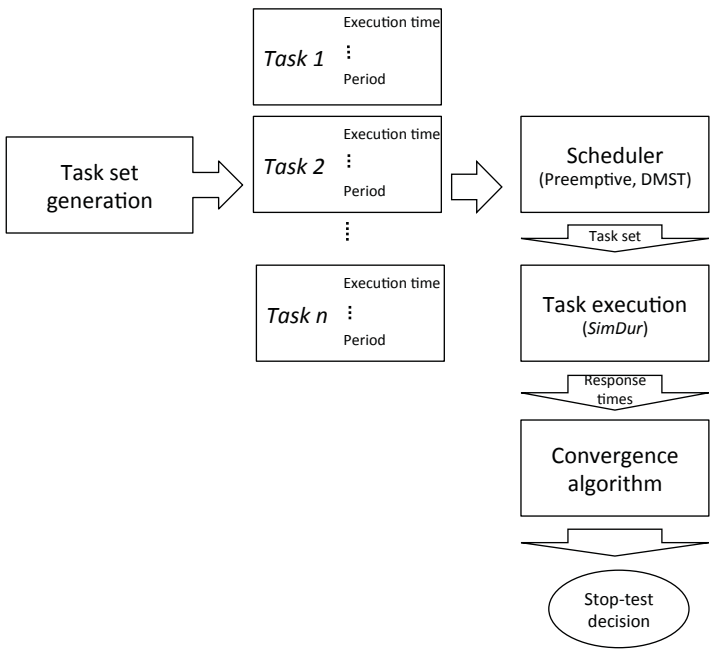

Figure 4.2: Overall scheme of task set simulation experiments

No task dependency and overhead, e.g., context switch time, is considered in the simulator. The simulator neither involves the underlying hardware characteristics. The simulation environment includes the following key parts:

- *Task set Generation* - A random number of tasks are generated with a *Best-Case Execution Time (BCET)* and WCET before the simulation starts. The BCET and WCET are the shortest and the longest execution times of a task respectively. For all the tasks, except the control scheduling tasks, a period is randomly chosen within a defined range. The deadline of the tasks is then set to be equal to the period.
- *Task execution* - Each time a task is released it is given a random execution time, according to a uniform distribution, in the range [BCET, WCET].
- *Scheduler* - A scheduler monitors each task's status: *delayed*, *in run queue* or *executed* and performs scheduling according to the preemptive *Deadline Monotonic Scheduling Theory (DMST)* based on the tasks' fixed priorities, i.e., the task with highest priority is executed first. Tasks are prioritized based on the *Deadline Monotonic Priority Ordering (DMPO)*, i.e., the shorter task deadline, the higher priority.

The simulator outputs testing data, i.e., response times, that are analysed by the convergence algorithm introduced in Chapter 1 in order to make a stop-test decision. Figure 4.3 shows the observed response times of an arbitrary task within a task set for a specified duration (each point in the graph represents a specific response time obtained for the task in the simulation).

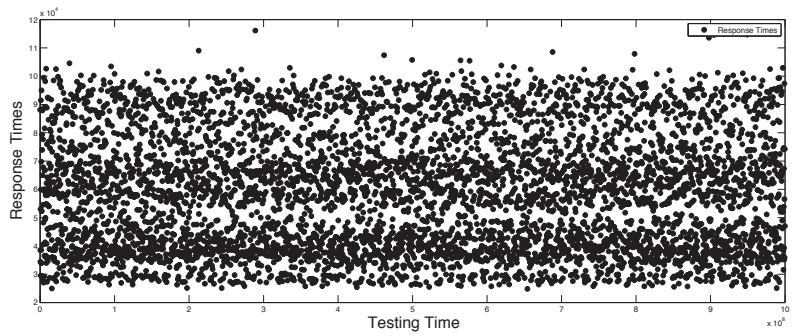

Figure 4.3: Simulator output for a given duration

The simulator also generates static WCRT for each task set which is based on the standard response time analysis for fixed priority scheduling [18]. The timing analysis for calculating the WCRT of task i, R_i, in a preemptive system of the type we consider is given by Equation 4.1 assuming no blocking by the lower priority tasks.

$$R_i^{n+1} = C_i + \sum_{j \in hp(i)} \lceil \frac{R_i^n}{T_j} \rceil C_j \tag{4.1}$$

with $R_i^0 = C_i$ which terminates when $R_i^{n+1} = R_i^n, or R_i^{n+1} > D_i$. R_i^n is the n^{th} iteration of calculating R_i, and C_i is the WCET for task i.

The reason to use the simulator is that, in this way, we have careful control over the task set characteristics and a *ground truth* for evaluations is established. Two ground truths are provided by the simulator: (i) WCRT achieved by static analysis which provides an exact safe result, and (ii) a *High Water Mark (HWM)* achieved by significantly longer simulation, far beyond the stop-test decision. Longer simulation is possible due to the nature of the simulator used. However, such increased testing would be prohibitively expensive in a real system. These two ground truths are shown in Figure 4.4 for a task in the simulator. The HWM is achieved by running the simulator for 10^{13} μs. However, it may miss the actual WCRT. The static analysis provides a safe upper bound for the WCRT.

4.1.2 Convergence Algorithm

In order to decide when to stop testing for timing characteristics of a system and to address the first research question, we propose a convergence algorithm. The algorithm analyses testing data, achieved by testing the system introduced in 4.1.1, to determine whether further testing would reveal new useful insight into the timing behaviour and, if not, it suggests testing to be stopped. The main idea behind the algorithm is that the system is tested for a while, e.g., $T_{current}$ where T determines the amount of testing time, and the response times are observed, then it is tested a little longer, e.g., $T_{current} + T_\Delta$, and it is examined whether we have seen significant new information at $T_{current} + T_\Delta$ compared to $T_{current}$. If this is the case, it indicates that further testing may still reveal new useful insight, thus, we test further and check again; if not,

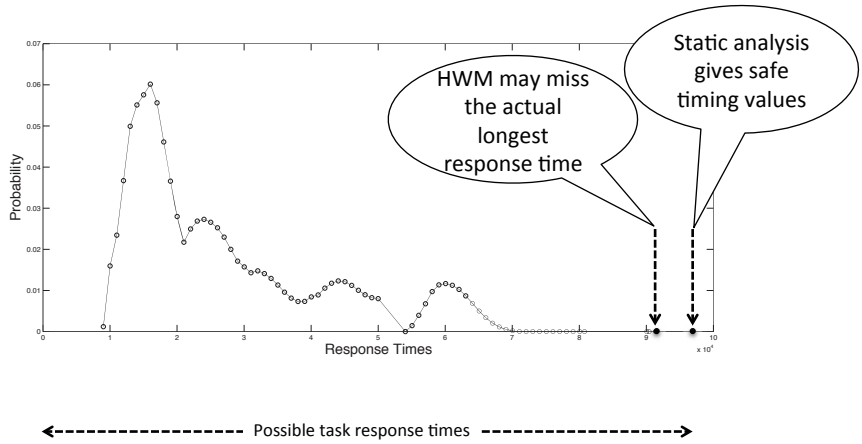

Figure 4.4: HWM vs. static analysis results

we conclude, with some statistical evidence, that further testing is not going to be significantly different, hence we should stop testing. To accomplish this idea, the algorithm looks into two successive distributions of response times, achieved at $T_{current}$ and $T_{current} + T_{\Delta}$ respectively (Figure 4.5), and determines whether they are significantly different. If this is not the case, it is concluded that the distributions have converged and further testing would not reveal significant new information. However, other testing techniques may still reveal significant information.

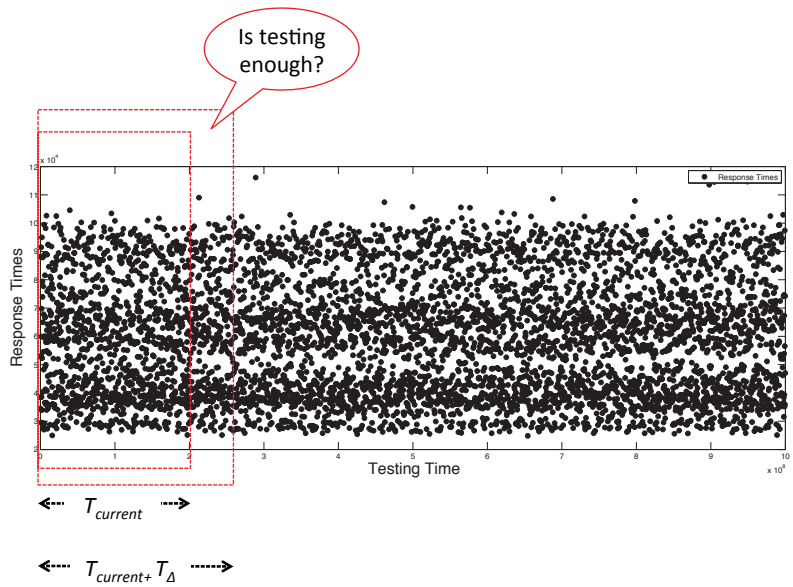

Figure 4.5: Convergence algorithm and analysis of observed response times

Since the amount of observed response times is huge, it is expensive to be collected and analysed in its raw format. Therefore, testing data is transformed into (i) the MORT values during testing and (ii) the histograms of response times, such that the scalability of the analysis is improved. The algorithm then looks into the transformed testing data and, firstly, determines whether the MORT has recently increased. If this is no longer the case, it secondly examines whether the distribution of response times has significantly changed. If the distribution does not change for a specific time, the algorithm concludes they have converged and further testing of the same nature would not be useful. To do the analysis, the algorithm is based on the following tests:

- *High Water Mark (HWM)*, which stands for the MORT at some specific points during testing, is used to determine whether the MORT has recently increased. As stated in Chapter 3, to decrease the number of convergence tests, we do not apply it early when

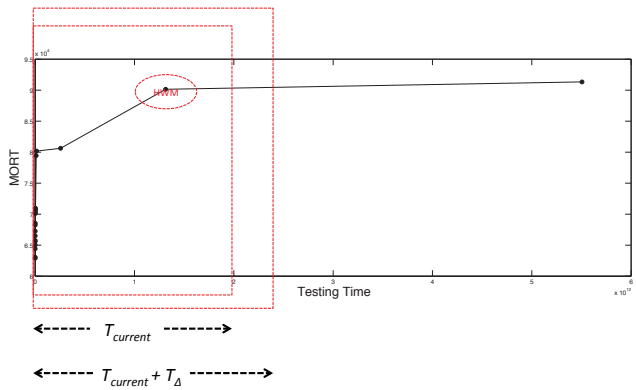

Figure 4.6: HWM within specified testing time

we just start testing but look into the HWMs. If we have seen no increase in the HWMs for a specific time, then we start applying the convergence metric. Figure 4.6 shows the MORT values for a specific task in the system during testing and the MORT depicted by the red oval corresponds to the HWM observed within $T_{current}$ = 2*10^{12} μs.

In order to run the HWM test, the algorithm compares HWMs corresponding to two successive response times distributions, depicted by $T_{current}$ and $T_{current} + T_{\Delta}$ in Figure 4.6, over a couple of iterations. If the HWM does not increase for a specific number of consecutive iterations, the algorithm analyses the distributions in more details using the convergence metric which is based on a statistical test. For example in Figure 4.6, HWM has not increased at $T_{current} + T_{\Delta}$ compared to $T_{current}$. The HWM test increases the scalability of the convergence algorithm by reducing the number of convergence tests that needs to be performed.

- When the MORT values have no increase for a specific time, i.e., the HWM test is passed, the algorithm applies a statistical test called *Kullback-Leibler DIVergence (KL DIV)* [19, 20] to examine whether the distribution of response times has significantly changed. The reason to use KL DIV is that we initially examine our testing data to see whether they belong to a normal distribution using the statistical *Kolmogorov-Smirnov (KS)* test [21]. The

KS test result shows that our data does not have a normal probability distribution and no specific assumptions can be made while we test the distributions for convergence. The KL DIV test, which measures the difference between two probability distributions, is a *nonparametric* test and makes no assumptions about the probability distributions of the variables being assessed, thus, it suits our analysis. Moreover, traditional inferential statistics show if there is a significant difference between distributions while the KL DIV shows the presence of such a difference either if it is significant or not.

As stated earlier, large amount of response times is expensive to be collected and analysed in its raw format. Therefore, testing data is transformed into the histograms of response times. Figure 4.7 shows $T_{current}$ histogram gained from the raw response times. Two histograms of $T_{current}$ and $T_{current} + T_{\Delta}$ distributions are generated to undergo the KL DIV test as shown in Figure 4.8 and the result shows the difference between these two. If the test result falls below a specified threshold at a required confidence level, it is concluded that $T_{current}$ and $T_{current} + T_{\Delta}$ have converged. The convergence indicates that no new significant information has been recently observed, thus, future timing behaviour is expected not to be very much different. At this point, the convergence algorithm proposes to stop testing.

Each bar in the histogram corresponds to the frequencies of the data falling between the ranges that the bar shows, also called a *bin*. In the case of our data, we observe and collect the frequencies of response times within each bin rather than collecting every single response time which significantly improves the scalability of the approach in terms of physical space allocated to store the data.

4.1.3 Evaluation

The algorithm is designed and evaluated with a particular focus on the ALARP principle which is a cost-benefit argument. Three criteria are defined for evaluations as follow.

- The first criterion relates to the closeness of the HWM when the algorithm suggests testing to be stopped (depicted by *HWM at*

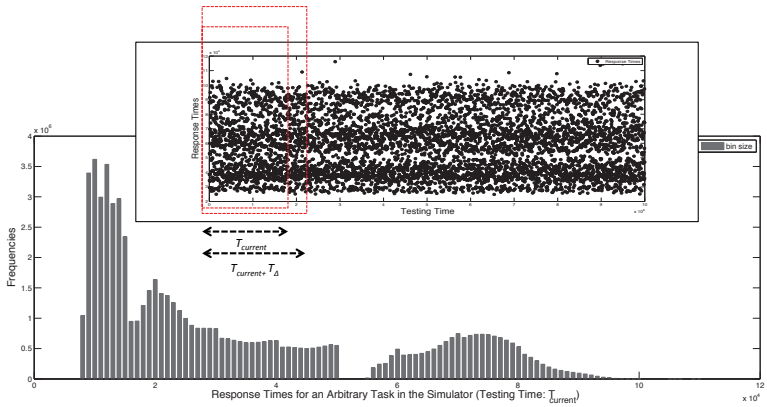

Figure 4.7: Distribution of response times and its histogram

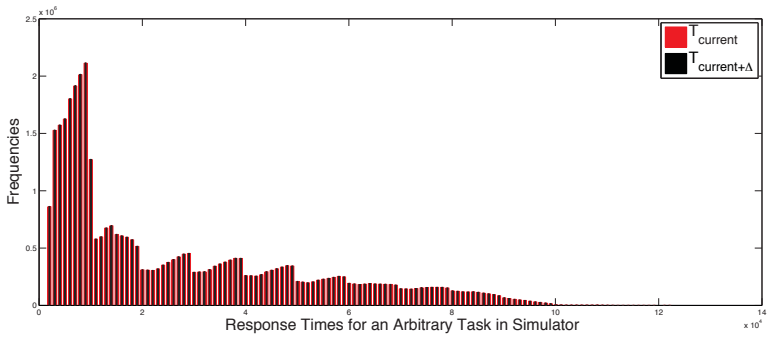

Figure 4.8: Histograms of response times and KLDIV test

StopPoint) to the last observed HWM, i.e., the ground truth (depicted by *GTHWM*), and is formulated as follows.

$$\frac{\text{GTHWM - HWM at StopPoint}}{\text{GTHWM}} \tag{4.2}$$

The smaller the equation result, the better performance in terms of MORT is achieved at *StopPoint*.

- The second criterion relates to testing effort at *StopPoint* relative to testing effort at GTHWM which is as follows.

$$\frac{\text{TestingEffort at StopPoint}}{\text{TestingEffort at GTHWM}} \tag{4.3}$$

Similar to Equation 4.2, smaller result indicates better performance of the algorithm in terms of testing effort. The first and second criteria help to compare the gain at *StopPoint* (closeness of the HWM at *StopPoint* to GTHWM) with the associated cost, i.e., cost-benefit analysis.

Figure 4.9 compares multiple algorithm's suggested *StopPoint*s and shows how such criteria help to evaluate the algorithm performance. Let us, firstly, start comparing *StopPoint1* with *StopPoint2*. The MORTs at both *StopPoint1* and *StopPoint2* are equal, thus, according to Equation 4.2, their performance in terms of MORT are also equal. However, testing effort at *StopPoint 2* is more than *StopPoint1* which results in higher value of Equation 4.3 at *StopPoint 2* rather than *StopPoint1* (0.64 vs. 0.5 respectively). So, it is concluded that *StopPoint1* results in better performance than *StopPoint2*.

Let us, this time, compare the algorithm performance at *StopPoint1* with *StopPoint3*. Both HWM and testing effort at *StopPoint3* is less then *StopPoint1* which means that the algorithm has better performance in terms of cost and poorer performance in terms of benefit at *StopPoint3* rather than *StopPoint1*. In order to compare the performance in such a situation, a quantified MORT, here called *ALARPHWM*, is defined so that if the HWM at *StopPoint* falls between *ALARPHWM* and *GTHWM*, it is considered as an acceptable performance. As all MORT values in the range [*ALARPHWM*, *GTHWM*] are acceptable in terms of benefit, the *StopPoint* with less effort is more desirable.

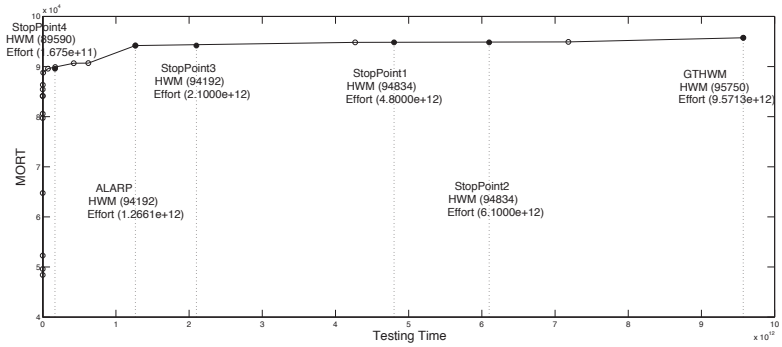

Figure 4.9: Evaluation of the convergence algorithm

- The *ALARPHWM* is defined as follows.

$$\text{ALARPHWM} = \text{GTHWM} - \xi\% * \text{GTHWM} \tag{4.4}$$

The value of ξ can be set according to requirements which is 5% in our evaluations (shown as ALARP in Figure 4.9). Consequently, the third criteria is defined as in Equation 4.5.

$$\frac{\text{HWM at StopPoint - ALARPHWM}}{\text{HWM at StopPoint}} \tag{4.5}$$

Based on this criterion, the algorithm has an acceptable performance in terms of benefit at both *StopPoint1* and *StopPoint3*. However, *StopPoint3* is associated with less cost compared to *StopPoint1*, thus, is a better stop-test decision. The performance at *StopPoint4* in Figure 4.9 is not acceptable as it results in the HWM being less than ALARPHWM.

These criteria help to evaluate the performance of the algorithm in terms of benefit at *StopPoint* (closeness to the GTHWM) against the associated cost (testing time) whilst the least acceptable benefit has to be met (ALARPHWM).

The Experimental Setup

Once the criteria for evaluating the algorithm performance are defined, the evaluation phase starts. In our case it is based on the trials of 20 tasks, of which 3 (sensor, calculator, actuator) are associated with

the control software in the simulator [22]. As stated earlier, the simulator is based on the ball and beam example. The simulator runs for $SimDur = 10000000$ sec in each trial and generates the following task set characteristics according to the pseudo code presented in Algorithm 4.1 (All timing values are in microseconds unless otherwise stated):

- Non-harmonic *periods* which are randomly generated within the input domain [50000, 200000] for tasks,
- Execution times which are randomly generated between the BCET and the WCET where BCET and WCET are calculated as shown in Algorithm 4.1, so that the overall task set utilisation is within [80%, 100%],
- *Deadlines* which are equal to periods,
- *Priorities* which are assigned based on DMPO.

Algorithm 4.1: The *Task Set Generation Pseudo Code*

Input: $NumberOfTasks, MinUTIL, MaxUTIL, UTILStep, MinPeriod, MaxPeriod, PeriodStep, MaxBCET, BCETStep$

Output: $TaskSetCharacteristics$

```
1  NumberOfTasks = 20;
2  foreach Task ∈ {TaskSet} do
3  |  TaskPeriod ← Rand(MinPeriod, MaxPeriod, PeriodStep);
4  |  TaskDeadline ← TaskPeriod;
5  |  TaskBCET ← Rand(1, MaxBCET, BCETStep);
6  |  TaskSetUTIL ← Rand(MinUTIL, MaxUTIL, UTILStep);
7  |  TaskUTIL ← UUniFast(TaskSetUTIL);
8  |  WCET ← TaskPeriod * TaskUTIL;
9  |  TaskPriority ← DMPO(TaskSet);
10 end
```

The *Rand* function in Algorithm 4.1 is initialized by a random seed and returns a value in the range [MinValue, MaxValue] with coefficient *ValueStep* in which *Value* can stand for period, BCET or UTIL. UTIL corresponds to the utilisation and the UUniFast algorithm generates each task's utilisation with uniform distributions [23]. The *DMPO* function returns each task priority based on the DMPO.

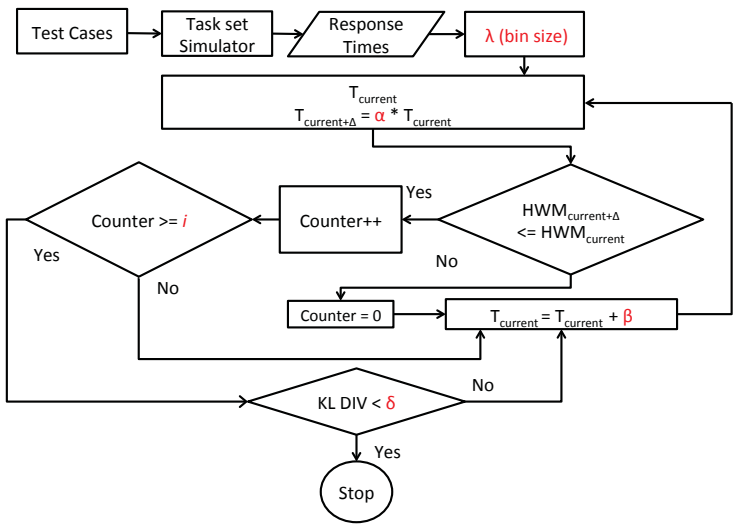

Figure 4.10: Convergence algorithm

4.2 Optimisation of the Convergence Algorithm

This contribution relates to the third research question as it deals with the optimisation of the convergence algorithm so that a better ALARP decision is made and the analysis becomes more scalable. The algorithm includes a set of parameters and the optimisation is done by tuning those parameters.

Figure 4.10 shows the steps within the convergence algorithm along with its parameters. The convergence algorithm takes each task's response times (testing data) as input, analyses them and proposes a *Stop-Point*. In order to improve the scalability of our method, a parameter called λ is introduced that keeps track of frequencies of response times falling between time t and $t + \lambda$. This process is called *binning* where λ defines the *bin size*. Consequently, the algorithm compares histograms of binned response times rather than distributions of individual values which saves us memory space. In each round of analysis, two histograms $T_{current}$ and $T_{current} + T_{\Delta}$ are compared where $T_{current} + T_{\Delta}$ is equal to α * $T_{current}$ and α is another parameter in the algorithm. At the

next step, the HWM test is applied on $T_{current}$ and $T_{current} + T_{\Delta}$ and if there has been no increase in the HWM for i successive iteration, then, the KL DIV test would be applied. If the KL DIV test's result falls below a threshold, δ, the algorithm concludes that the distributions of response times have converged, thus suggests to stop testing. i and δ are the other two parameters of the convergence algorithm. If neither test passed, $T_{current}$ is increased by β μs and the next analysis round starts. These steps are taken for each task in the simulator and the algorithm would stop when the last task in the task set passes the HWM and KL DIV tests.

In Paper A, the parameter set, which includes $\{\lambda, \alpha, i, \delta, \beta\}$, were tuned by means of limited trial and improvement experiments. In Paper B, firstly, it is determined whether the parameters do affect the algorithm. Secondly, the most influential parameters are identified and tuned. This process is based on the *Design of Experiments (DoE)* approach. The DoE approach tunes the parameter set such that the performance and scalability of the convergence algorithm are improved.

There are five parameters involved in the optimisation which may lead to combinatorial explosion if being optimised in an inappropriate fashion, e.g., assume we are interested in assessing the quality of four candidates for each parameter within its input domain. Then, $4^5 = 1024$ combinations of these candidates have to be assessed. If each candidate is assessed by running three simulations, it means 3072 simulations in total, which may be prohibitively expensive. DoE, however, is based upon a two-phase method to systematically explore each parameters's input domain. At the beginning, DoE determines the simulation duration such that the scalability is improved. Then, it identifies insignificant parameters with minor or no effect on the performance and discarded them in the first phase called *factor screening* before digging deeper the most significant parameters. This phase is based on the *Analysis of Variance (ANOVA)* [3] approach and it avoids the combinatorial explosion and enhances the scalability. In the second phase each significant parameter is sampled at high resolution, i.e., more candidates from parameter's input domain are examined. The high resolution phase is feasible as the number of candidates with polynomial experimental cost growth can be increased while the number of parameters with exponential experimental cost growth has been already dropped in the factor screening phase.

This phase eventually derives a candidate resulting in the best algorithm performance and scalability.

4.3 Identification of Task set Parameters Influencing the Convergence Algorithm

In Paper C, which relates to the fourth research question, we firstly define a set of task set parameters, including {Period, Offset, Number of tasks, Harmonic vs. nonharmonic period}, as they do affect the timing behaviour of tasks. *Harmonic vs. nonharmonic* period relates to the way each task set's periods are generated. Harmonic periods require that every task's period evenly divides every longer period which is not the case for non-harmonic periods. Non-harmonic periods result in more complex timing behaviour.

Secondly, we define new ranges of the parameter set: {Period, Offset, Number of tasks, Harmonic vs. nonharmonic period} over which we would like to examine the performance of the algorithm, e.g., in order to see what would happen if the number of tasks increase from 10 to 50, the algorithm performance is analysed over the range [10, 50] for this parameter. Finally, we identify which parameters are the most influential on the convergence algorithm using the ANOVA approach.

To make the convergence algorithm robust, it is important that the stop-test decision is sound over a required range of task sets. However, some values within the required ranges of task sets significantly degrade the algorithm performance and its robustness according to our results in Paper C. Therefore, our next piece of work will include tuning of the algorithm against conditions that adversely affect it.

4.4 Improving the Convergence Algorithm When Testing Data Tends to Converge Late

The last contribution in this thesis relates to the third research question and tries to improve the convergence algorithm performance. The

convergence algorithm, based on KL DIV, has shown good results for most considered scenarios. However, there have been experiments in which the algorithm results in a premature stop-test decision, i.e., it suggests to stop even if further testing is likely to reveal new interesting information. Therefore, in Paper D, we focus on KL DIV and investigate whether any of its properties may adversely affect the algorithm. For example, one of the issues with the KL DIV test is that large quantities of testing data can affect the test results, i.e, subtle changes are harder to see with lots of data. At the other hand, as stated in Section 4.1, our algorithm analyses a large amount of testing data. It compares two successive histograms $T_{current}$ and $T_{current} + T_{\Delta}$ while a big part of these two histograms intersects and only subtle changes are observed in $T_{current}+T_{\Delta}$ compared to $T_{current}$. Thus, the more sensitive a statistical test is to such changes, the more accurate dissimilarity measure between histograms is achieved. Therefore, we use a new statistical test called *Earth Mover's Distance (EMD)* [24] which is sensitive to such delicate changes and evaluate it by means of experiments.

Chapter 5

Conclusions, Limitations and Future Work

Testing is an important step in the development process of safety-critical systems. However, it is also one of the most expensive parts. Therefore, it is essential for testers to determine when to stop testing the system. This thesis proposes a convergence algorithm to make such a decision in the context of the worst-case timing characteristics of systems. Our simulation-based evaluations show that the algorithm performs well, that is, it stops after the point at which a human with clairvoyance would stop testing. The evaluations also show that the Maximum Observed Response Time (MORT) values and the amount of testing (cost) which is avoided at the algorithm's *StopPoint* are reasonable based on the ALARP principle. We also identify which task set parameters are the most influential on the algorithm and whether they adversely affect it. We can use these influential parameters later to stress test the algorithm towards its worst-case performance, then tune it such that it becomes robust against such stressed conditions. Our evaluation also shows premature stop-test decisions in some cases when using the KL DIV test. Therefore, we investigate whether an alternative test called EMD could improve premature stop-test decisions.

5.1 Discussions

In order to make a stop-test decision in the context of the worst-case timing behaviour of safety-critical systems, we propose a convergence algorithm and tune its parameters to improve the performance and scalability. We also focus on controlled data (test input) to examine which task set parameters are the most influential on the algorithm and to determine which ranges of these parameters significantly degrade its performance. At the final step, we investigate whether the algorithm can suggest a better stop-test decision when testing data tends to converge late focusing on its statistical test. On a more specific level our contributions are the following.

1. We define convergence metrics that look into the test data (response times) and decide whether further testing would reveal new significant information about the timing behaviour. If the metric indicates that there is no significant findings by further testing, then this suggests testing to be stopped. The algorithm proposed in this thesis is based on such a convergence metric. It also includes a set of parameters which are tuned to improve its performance and scalability.

2. The algorithm is evaluated based on the ALARP principle which is a cost-benefit argument, i.e., we show that further testing after the algorithm *StopPoint* is not justified as the amount of information obtained is not significant in relation to the cost that it would incur.

3. In order to make the algorithm robust, first, we define task set parameters over which we would like the algorithm to hold. We also examine whether these parameters can adversely affect the algorithm to further use them in stress testing.

4. Finally, in the last piece of work, we try to improve the premature stop-test decisions caused by KL DIV and use a new statistical test called EMD.

Table 5.1 shows how the contributions (included papers) relate to the research questions defined in Chapter 3.

Table 5.1: Thesis Contributions

	RQ1	RQ2	RQ3	RQ4
Paper A	✓	✓		
Paper B			✓	
Paper C				✓
Paper D			✓	

5.2 Limitations

As stated in Chapter 4, our experiments and evaluations are based on an in-house task set simulator which gives us the control over the task set complexity and provides two ground truths: HWM and static WCRTs. HWM is achieved by significantly longer simulation time which is prohibitively expensive in a real system. The simulator so far provides simplified task sets and it has been useful to evaluate our initial hypothesis. However, in order to have a thorough evaluation of the algorithm, more complex and realistic scenarios have to be considered. This helps us to determine to what extent our convergence algorithm would be applicable, e.g., in a real system. Examples of such scenarios may include simulations dealing with task complex timing behaviour, task dependency and underlying hardware characteristics.

5.3 Future Work

Some possible directions of our work for future are as follows.

- In order to deal with more complex timing behaviour, the main part of our future work would be to investigate more general task models and lifting some of our current restrictions in the simulator. In other words, we would like to investigate how the convergence algorithm can be applied in practice and real world scenarios. Our strategy would be to gradually increase task set complexity, e.g., using a more advanced simulator, and get as close as possible to

a real system characteristics. Then, we determine whether the convergence algorithm can still operate in the new circumstances.

- Our work has been based on random test cases, i.e., we simulate random task sets. One interesting direction would be considering other testing techniques than *random* to determine to what extent our statistical results hold for such scenarios. For example, what happens if we specifically use test cases which are focused on the worst-case. Also, testing data, so far, has been the worst-case timing characteristics of a system. Another possible future work can be considering other characteristics than the worst-case timing.

- Another possible direction could be the support of *mixed criticality* as we anticipate it may cause tasks get more complex temporal behaviour especially for the tasks of low criticality. Criticality is a designation of the level of assurance against failure needed for a system component and a *mixed criticality system (MCS)* is the one that has two or more distinct levels, e.g., safety critical, mission critical and low critical [25].

Bibliography

[1] I. Burnstein. *Practical Software Testing: A Process-Oriented Approach*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[2] Health and safety at work act 1974. Technical report, The National Archives, 1974.

[3] L. Eriksson. *Design of Experiments: Principles and Applications*. Umetrics, 2008.

[4] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1(1):11–33, January 2004.

[5] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer Publishing Company, Incorporated, 2nd edition, 2011.

[6] A. Burns and A. J. Wellings. *Real-Time Systems and Programming Languages: ADA 95, Real-Time Java, and Real-Time POSIX*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 2001.

[7] C. Anderson and M. Dorfman. *Aerospace software engineering: a collection of concepts*. Progress in astronautics and aeronautics. American Institute of Aeronautics and Astronautics, 1991.

[8] B. Beizer. *Black-box Testing: Techniques for Functional Testing of Software and Systems*. John Wiley & Sons, Inc., 1995.

[9] H. D. Mills. *Software Productivity*. Little, Brown Computer Systems Series. Little, Brown, 1983.

[10] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):36:1–36:53, May 2008.

[11] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiñones, and F. J. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *Proceedings of the Euromicro Conference on Real-Time Systems*, pages 91–101, 2012.

[12] F. J. Cazorla, E. Quinones, T. Vardanega, L. Cucu-Grosjean, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim. PROARTIS: Probabilistically Analysable Real-Time Systems. Research Report RR-7869, INRIA, January 2012.

[13] E. J. Gumbel. *Statistics of Extremes*. Dover books on mathematics. Dover Publications, 2004.

[14] S. Edgar and A. Burns. Statistical analysis of WCET for scheduling. In *Real-Time Systems Symposium, 2001. (RTSS 2001). Proceedings. 22nd IEEE*, pages 215–224, Dec 2001.

[15] J. Hansen, S. Hissam, and G. A. Moreno. Statistical-Based WCET Estimation and Validation. In Niklas Holsti, editor, *9th International Workshop on Worst-Case Execution Time Analysis (WCET'09)*, volume 10 of *OpenAccess Series in Informatics (OASIcs)*, pages 1–11, Dagstuhl, Germany, 2009. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. also published in print by Austrian Computer Society (OCG) with ISBN 978-3-85403-252-6.

[16] C.R. Kothari. *Research Methodology: Methods and Techniques*. New Age International (P) Limited, 2004.

[17] I. Bate, P. Nightingale, and J. McDermid. Establishing timing requirements for control loops in real-time systems. *Microprocessors and Microsystems*, 27(4):159–169, 2003.

[18] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):284–292, Sept 1993.

[19] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics (AMS)*, 22(1):79–86, 1951.

[20] S. Kullback. *Information Theory and Statistics*. Wiley, 1959.

[21] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley, January 1968.

[22] I. Bate. Utilising application flexibility in energy aware computing. In *Proceedings of the 2008 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, RTCSA '08, pages 285–290, Washington, DC, USA, 2008. IEEE Computer Society.

[23] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Syst.*, 30(1-2):129–154, 2005.

[24] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *Int. J. Comput. Vision*, 40(2):99–121, November 2000.

[25] A. Burns and R. I. Davis. Mixed criticality systems - a review. University of York, 2015.