The Journal of Systems and Software 146 (2018) 26–41

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

ESPRET: A tool for execution time estimation of manual test cases

Sahar Tahvili[a,b,*], Wasif Afzal[b], Mehrdad Saadatmand[a], Markus Bohlin[a], Sharvathul Hasan Ameerjan[b]

[a] RISE SICS Västerås AB, Sweden
[b] Mälardalen University, Västerås, Sweden

ARTICLE INFO

Article history:
Received 8 November 2017
Revised 21 August 2018
Accepted 5 September 2018
Available online 6 September 2018

Keywords:
Software testing
Execution time
Test specification
Optimization
Manual testing
Regression analysis

ABSTRACT

Manual testing is still a predominant and an important approach for validation of computer systems, particularly in certain domains such as safety-critical systems. Knowing the execution time of test cases is important to perform test scheduling, prioritization and progress monitoring. In this work, we present, apply and evaluate *ESPRET* (EStimation and PRediction of Execution Time) as our tool for estimating and predicting the execution time of manual test cases based on their test specifications. Our approach works by extracting timing information for various steps in manual test specification. This information is then used to estimate the maximum time for test steps that have not previously been executed, but for which textual specifications exist. As part of our approach, natural language parsing of the specifications is performed to identify word combinations to check whether existing timing information on various test steps is already available or not. Since executing test cases on the several machines may take different time, we predict the actual execution time for test cases by a set of regression models. Finally, an empirical evaluation of the approach and tool has been performed on a railway use case at Bombardier Transportation (BT) in Sweden.

1. Introduction

As is well known, an efficient and effective software testing process is important for delivering high quality software products (Casey, 2008; Afzal et al., 2016). Software testing is overall costly and a time-consuming process. This is even more true for manual testing which requires a tester to perform the test operations on the system under test (SUT) without the help of test automation (Itkonen et al., 2007). Test automation has therefore become a popular approach to reduce the cost of software testing; a computer can follow a sequence of steps more quickly than a tester and can run the tests overnight to present the results in the morning (Dustin et al., 2009; Strandberg et al., 2016). On the other hand, the labor that is saved in actual testing through automation must instead be spent on writing the test program. Depending on the type of application to be tested and the automation tools that are chosen, this may itself require more labor than a manual testing approach, at least in the beginning of a testing process (Flemström et al., 2018). In addition, test automation in its most current form is neither able to cover human intuition, inference, inductive reasoning, nor can it change course in the middle of a test run to examine something that had not been previously considered (Casey, 2008). Therefore, manual testing still has a critical role in the software testing process and cannot be easily replaced.

Over the past few years, the interest in industry to find ways to improve effectiveness and efficiency of software testing (Garousi et al., 2017b; Afzal et al., 2015) has grown. Effective software testing processes and decision supports for these should reduce the total cost of testing and result in earlier fault detection (Kasurinen et al., 2010; Tahvili et al., 2016a; Engström and Runeson, 2011), and several factors such as coverage, test strategy, test implementation, planning and analysis of test results should be considered (Felderer and Ramler, 2014; Strandberg et al., 2018). In this context, test case selection and prioritization are quickly becoming an inseparable part of an overall test strategy (Singh and Sahib, 2014). There are several methods proposed for test case selection, prioritization and scheduling (Li et al., 2007; Engström and Runeson, 2011; Kasurinen et al., 2010; Hao et al., 2016), where the initial problem of test prioritization is recognized as a multi criteria decision making problem, meaning that a set of criteria must be identified before preparing test cases for execution. The execution time of test cases is one of the important criteria that impacts test scheduling, pri-

* Corresponding author.
E-mail addresses: sahar.tahvili@ri.se (S. Tahvili), wasif.afzal@mdh.se (W. Afzal), mehrdad.saadatmand@ri.se (M. Saadatmand), markus.bohlin@ri.se (M. Bohlin), san15014@student.mdh.se (S.H. Ameerjan).

https://doi.org/10.1016/j.jss.2018.09.003
0164-1212/

oritization and progress monitoring. In our previous works (Tahvili et al., 2016b; Tahvili, 2016), we proposed a multi criteria decision support system (DSS) for selecting and prioritizing manual test cases at integration testing level, considering some criteria such as execution time, dependency and requirement coverage.

This paper is an improved version of Tahvili et al. (2017), which presents a novel automated approach for estimating and predicting the execution time of manual test cases based on test specifications and available historical data on previously executed test cases. The execution time of test cases is categorized by us into two main groups: 1- the maximum execution time (MT) and 2- the actual execution time (AT), where MT is supposed to be a constant and AT is a variable time value. Later in this paper, the formal definitions of MT and AT are provided. Our proposed approach works by extracting timing information for various steps in manual test cases. This information is then used to estimate the maximum time for test steps that have not previously been executed, but for which textual specifications exist. As part of our approach, natural language parsing of the specifications is performed to identify word combinations, which in turn are used to check whether existing timing information on various test activities is already available or not. Additionally, a set of regression models is used to predict the actual execution time (AT) for manual test cases.

ESPRET (EStimation and PRediction of Execution Time) is the tool implementation of our proposed approach for estimating and predicting the execution time of test cases. *ESPRET* estimates the execution time of manual test cases by recognizing some critical and key elements in textual test specifications and analyzing the historical test data. Since the actual execution time for test cases is in essence a system dependent time (variable value), the polynomial and spline regression models have been applied for providing a prediction of this time. Moreover, the prediction error of the regression models has been measured in order to evaluate the prediction algorithms. An empirical study at Bombardier Transportation (BT) is also done to evaluate the proposed approach, as part of our industry-academia collaboration strategy to facilitate knowledge transfer and innovation (Garousi et al., 2017a).

The organization of this paper is laid out as follows: Section 2 provides a background of the initial problem and also an overview of research on execution time prediction, Section 3 describes the proposed approach. An industrial case study has been designed in Section 4, threats to validity and delimitations are discussed in Section 5. Section 6 is about the discussion and some future direction of the present work and finally Section 7 concludes the paper.

2. Background and related work

A typical goal in software testing is to detect as many defects as possible given the allocated testing resources. Therefore, recognizing the required time and budget for creating and executing test cases plays a vital role in an effective testing strategy (Tahvili et al., 2016a; Afzal and Torkar, 2008). Computing the overall time for executing manual test cases, however, is a challenging task. Building a time estimation model entails data analysis and most organizations lack sufficient historical data (Angelis et al., 2001). Without the experience data, it will be difficult to create an accurate estimation of the execution time. Moreover, manual test specifications are written in natural text by a group of testers with different level of writing and testing skills, which makes the initial problem more complicated.

Estimation and prediction of software resources has been an interesting area of investigation in software engineering for several years. The overwhelming goal of such investigations is to assist in planning of resources and to complete various development tasks effectively and efficiently. Some examples of estimation and prediction research in software engineering includes software fault/defect prediction, software effort/cost estimation and software maintenance effort prediction. A variety of model building techniques have been investigated including regression analysis, neural networks, case-based reasoning and soft computing techniques.

Prediction of software testing effort has received less research as compared to other prediction and estimation tasks in software engineering. For software test effort estimation and prediction, one factor of importance relates to the ability of making accurate predictions early in the development process. It is a common understanding that models developed using lines of code would enable predictions to be available much later than, for example, models developed from requirements. Similarly, predictions based on test specifications would also enable timely decision-support when compared with predictions based on lines of code. Below we discuss representative relevant papers with a discussion of their pros and cons. We end this section with a discussion on our approach, while the details of our approach are presented in Section 3.

Nageswaran (2001) presented an use case points (UCP) approach for estimation of software testing effort. UCP identifies the number of actors, use cases, software requirements, technical and environmental factors for estimating the test effort. Thus, UCP is not dependent on lines of code (LoC) and function points (FP). The required test effort can be estimated in the early stages of software development process. Although UCP can estimate the test effort as a whole, including test plan, design, execution, monitoring and reporting, it is not able to capture each and every instance of test activity such as one test case execution (Zhu et al., 2008). Srivastava et al. (2012), proposed a similar approach to UCP for test effort estimation, called Cuckoo search method. This method also utilizes use cases to estimate the effort for testing. The principal difference between Cuckoo search and UCP is the way that they assign a value to their parameters. UCP assigns a fixed value (or weightage) to the parameters (e.g. actors, use cases) while a range of static or dynamic values for each parameter would be assigned by Cuckoo search, depending on the target system. Additionally, Cuckoo search takes into account the expertise of development and testing team. In order to apply Cuckoo search for estimating the required effort for a new project, historical data on at least on one project is required.

Accumulated efficiency is another method proposed by e. Silva et al. (2009), which focuses on team efficiency to estimate the execution effort of a test suite. Accumulated efficiency captures and analyses LoC and therefore cannot be utilized in the early stages of software development. However, this method does not require historical analysis and natural language processing techniques. Accumulated efficiency estimates the whole testing effort instead of estimating effort for individual test case executions and has been designed for the systems with a web interface.

de Almeida et al. (2009) proposed an alternative method, called Weighted Nageswaran, for estimating the required effort to test a software based on use cases. Since different test cases have different influence in their efforts, the Weighted Nageswaran method classifies test cases into simple, medium and complex groups, based on the required efforts.

Sharma and Kushwaha (2013, 2010) proposed a test metric for the estimation of software testing effort, using software requirement specification (SRS) document. The proposed method works by extracting the requirements from the SRS document and assigning complexities to them based on their level. With the help of the weightage assigned to the requirements based on complexity, test effort is estimated. Since this method is based on a SRS document, the required effort can be estimated in the early stage of software development. On the other side, using requirement specification instead of test specification might be less reliable as one needs

Table 1
Summary of related work.

Reference	Purpose of paper	Drawback
Nageswaran (2001)	An use case points (UCP) approach for estimation of software testing effort	A gross estimate of testing effort and not on individual test activities
Srivastava et al. (2012)	An use case points (UCP) approach for estimation of software testing effort using Cuckoo search	Historical data on at least one project required
e. Silva et al. (2009)	Uses team efficiency to estimate the execution effort of a test suite	Use of LoC make it not unfit for estimation in early stages
de Almeida et al. (2009)	Uses use case information to estimate test effort	A gross estimate of testing effort and not on individual test activities
Sharma and Kushwaha (2013) Sharma and Kushwaha (2010)	Uses SRS to estimate testing effort	Needs to ensure traceability between requirements and test specification
Aranha and Borba (2007)	Uses test steps and execution points for estimating test execution effort	Use of controlled natural language may impose restrictions
Torkar et al. (2010)	A dynamic Bayesian network is used for predicting test effort in an iterative software development environment	A number of variables are used including test team size, length of iteration and number of identified faults to name a few
Nguyen et al. (2013)	Measures the size of the test case based on its checkpoints, preconditions and test data, as well as the type of testing.	The test cases have to be available at the time the estimation is performed.

to ensure the traceability between requirements and test specifications. Aranha and Borba in Aranha and Borba (2007) overcame the mentioned limitation in Sharma and Kushwaha (2013) by using test specification instead of requirements specification. The proposed approach is based on an estimation model, which extracts test steps of a test specification document. It works based on test case size (the number of steps) and execution points for estimating the test execution effort. The execution points are fixed based on the historical data of previously executed test cases. The proposed method requires a controlled natural language (CNL) text, which means a standard format of language should be used in the test specifications. In other words, testers have some restrictions in the usage of lexicon and grammar for creating test cases, otherwise the proposed method is not able to estimate the required time for execution. There is a limitation of writing an action in many different ways. This approach identifies that each and every test step has one main verb and zero or many arguments supporting the main verb. Each and every verb identified in a test step represents the action and arguments add more information to the action. For using this method for effort estimation, test cases should be readily available. Moreover, this method uses test steps for execution time estimation. Therefore, the cost of estimation is high. Each time, the relation between test execution time and execution points has to be shown.

A number of test effort estimation methods are applicable for iterative software development. Silva et al.'s approach (e. Silva et al., 2009) was discussed earlier in this section and belongs to one such methods. Their approach is applicable in cases where several test cycles are performed. A similar approach is presented in Torkar et al. (2010) where a dynamic Bayesian network is used for predicting test effort in an iterative software development environment. A number of variables are used to make the predictions, including test team size, length of iteration and number of identified faults to name a few. Nguyen et al. (2013) named their approach as qEstimation, which measures the size of the test case based on its checkpoints, preconditions and test data, as well as the type of testing. A checkpoint, precondition, data, and type of the test case is used to evaluate a test case's complexity. The size estimate is then translated into effort and a feedback mechanism improves the estimation process for the next testing cycle. Table 1 presents a summary of the related work.

In our proposed approach, we use test specifications to estimate the execution time, without using execution points (as in Aranha and Borba (2007)). A test or a test case specification refers to the documentation of a set of one or more test cases (ISO/IEC/IEEE 29119-1:2013, E). In our context, a test case consists of a set of pre-conditions, inputs (called as test steps), and expected results. We make use of several elements in a manually written test case such as its size in terms of number of test steps, number of "waiting" steps and the number of pre-conditions. Moreover, scripted versions of the manual test specifications as well as test logs are utilized (more details on our approach are given in Section 3). Our approach does not require CNL in test cases, which makes the model more general. The concept of actual execution time has been proposed by us in Tahvili et al. (2017) and the continuous comparison between actual time (AT) and maximum (MT) helps us to estimate a more accurate value for MT. We have also estimated the error rates for our method. Additionally, *ESPRET* has been implemented as a tool for aiding testers and test managers in estimating the test case execution time.

3. Description of the proposed approach

In this section, we describe the details of our proposed approach for estimating the maximum execution time (MT) and predicting the actual execution time (AT) of manual test cases. This is done by determining the MT and AT values of test steps constituting a test case. Our approach takes multiple test process artifacts (e.g., test specification, requirement specification, test scripts, test logs) as input and produces the output in the form of a time value representing test execution time of test cases (as a summation of execution time of their test steps). It is important to note that there can be other factors which can contribute to the overall time required for executing a test case, such as skill of testers, setup times, and other human factors. However, in this work, we do not take such factors into account.

Generally, there is a difference between the maximum length of time that a test case is allowed to execute and the actual time that the same test case could take for the execution on a specific machine. Therefore, the concept of execution time can be categorized into two main groups:

- **Maximum execution time (MT):** is the maximum time that a test activity (or a test case) is allowed to take for execution. Assuming the maximum execution time for a test step is t seconds, if the system takes more time than t, the test step is considered as a failure. MT is independent from the system properties.

The maximum execution time is a non-negotiable deadline, which describes the reliability or the correct functional behavior. In the most cases, the MT value is explicitly explained in the software requirements specifications, such as: *Auxiliary compressor is permanently blocked within 3 s when requested*. The blocking process for the auxiliary compressor can be tested by one (or more) test cases, where the maximum length of time that the test case(s) is allowed to take is 3 s. This blocking process might take less than 3 s on different machines, but if this process takes more than 3 s (even one millisecond), the mentioned requirement is not fulfilled.

- **Actual Execution Time (AT):** is the real time (t') taken by the system executing a test activity (e.g., a test step or a test case), which is dependent on the system properties. The actual execution time per test step is equal or less than maximum execution time ($t' \leq t$).

The actual execution time of a test case is a machine dependent variable, meaning that having multiple machines running the same test case may take different times (Bush et al., 1989).

The MT value can be used as a timeout value when developing test scripts. Therefore, estimating a good margin for MT is important from the time-saving aspect. In other words, a too low MT value will result in test cases to fail due to timeout issue and not necessarily a fault in the system. However, estimating a too large MT value for a test case will lead to unnecessary waiting between steps and thus wasting of testing time. In fact, the MT and AT values are related to each other, where AT can be predicted based on MT and a safe margin for the value of MT can be determined via monitoring of several ATs on different machines. Based on the above definitions, our approach consists of the following. The details of each phase will be elaborated in subsequent subsections.

1. *Parsing and historical data collection:* in this phase, the AT value of previously executed test steps (test logs), as well as the MT value of test steps in existing test scripts (if any) are collected (*Phase 1*).
2. *The estimation phase*: the MT values are estimated for test cases in this phase. Several approaches such as NLP and log analysis are utilized (*Phase 2*).
3. *The prediction phase*: the AT values are predicted for test cases through applying the regression models (*Phase 3*).

Moreover, in this paper, we introduce m_t as the system baseline reaction time, which represents the response time that system under test (SUT) takes to react to a given input. The baseline time value is assigned as a maximum execution time for the steps of newly designed test cases, which have never been executed before and there exists no similar and matching activity for the steps of such test cases. The initial m_t value in the present work is assumed as 3 s in consultation with the testers and engineers at Bombardier Transportation. However, the baseline value can be determined in any other environment by monitoring the response time on several machines over a period of time. Fig. 1 provides a summary and illustrates the start and end points, the different steps and the phases of our proposed approach.

3.1. Parsing and historical data collection

For some test steps, MT and AT values can already exist. Therefore, in this phase, different sources such as test logs, test scripts, test specifications, and even requirement specifications and already known timing constraints (e.g., by consulting test experts) are used to build a database of test steps with MT and/or AT values. Whenever a test step is executed for the first time, its actual execution time will also be stored as its maximum time (i.e., MT_{stored}=AT_{stored}). In future executions of the same test step, the new actual execution time will replace the stored one (i.e. AT_{stored}=AT_{new}). Also, if the stored *maximum* execution time for this step is less than the new *actual* execution time, we store the new actual execution time as maximum execution time of this step (i.e. MT_{stored}=AT_{new}).

Test specifications are usually written in a natural language, thus the natural language processing (NLP) methods need to be applied for analyzing a test specification. Table 2 represents a part of a real, industrial test specification from a safety-critical train control management system, where the description of steps is outlined. Table 2 has been extracted from the IBM rational DOORS[1] database server at Bombardier Transportation and consists of several steps with a set of actions and expected reactions.

As we can see in Table 2, a typical manual test case specification at BT starts with an *Initial State* and ends with the *Clean up*. Recognizing this pattern has helped us to track the test steps among other information (e.g. testers ID, testing date, corresponding requirement) described textually in a test specification. Moreover, the overall time for performing some of the test steps are explicitly specified in some test specifications. For instance, in step 3, the *logout* process should be done in 10 s, which is the maximum allowed time for this step. In step 4, the testers need just *wait* for 20 s before starting the next step, thus the maximum execution time for this step is equal to 20 s. Sometimes, the timing information for executing a test case is specified in the corresponding requirement. For example in one of the software requirement specification (SRS) for the air supply system at BT, it mentions that: *Auxiliary compressor is permanently blocked or has not run for 30 s when requested*. As is obvious in this example, the maximum execution time for the corresponding test case (or test step) to this requirement can be extracted from the requirement specification. In our proposed approach, we capture and utilize all available timing information in the test specification, test scripts and sometimes the requirements specifications. As outlined earlier in this section, we parse and collect historical data for test cases in *Phase 1*. As a test specification is written by (various) testers, the initial problem does not involve Controlled Natural Language (CNL) text. In our recent work (Tahvili et al., 2017), we offered a solution for this problem by parsing the key characters in test cases in order to reduce or eliminate ambiguity and to identify the key elements of test cases. Generally, in order to understand the entire sentence, we need to understand some keywords per sentence, which are called the critical elements (Revlin, 2012). In other words, the critical elements are the main building blocks of a sentence, which can be divided into the Verb Phrases (VPs) and the Noun Phrases (NPs). In linguistics, the NP represents the objects and VP usually describes the actions of the object(s) in a sentence (Knott, 2012).

To make our proposed approach more efficient, for each test step (each sentence), *ESPRET* just captures *Verb* and *Objective Argument* as the critical elements per test step. Through processing of the critical elements, the main goal of a test step can be recognized and furthermore some redundant elements such as propositions, subjects and adjectives are removed automatically from a test case specification, which reduces the size of our database. For instance, the test step 3 in Table 2, is parsed by NLTK (Natural Language Toolkit in Python) as:

```
The query:
Logout from A1 cab through removing the key
card cab in 10 seconds.
Parse:
VP (VB Logout)
PP (IN from)
NP (NN A1) (NN cab)
PP (IN through)
```

[1] Dynamic object-oriented requirements system.

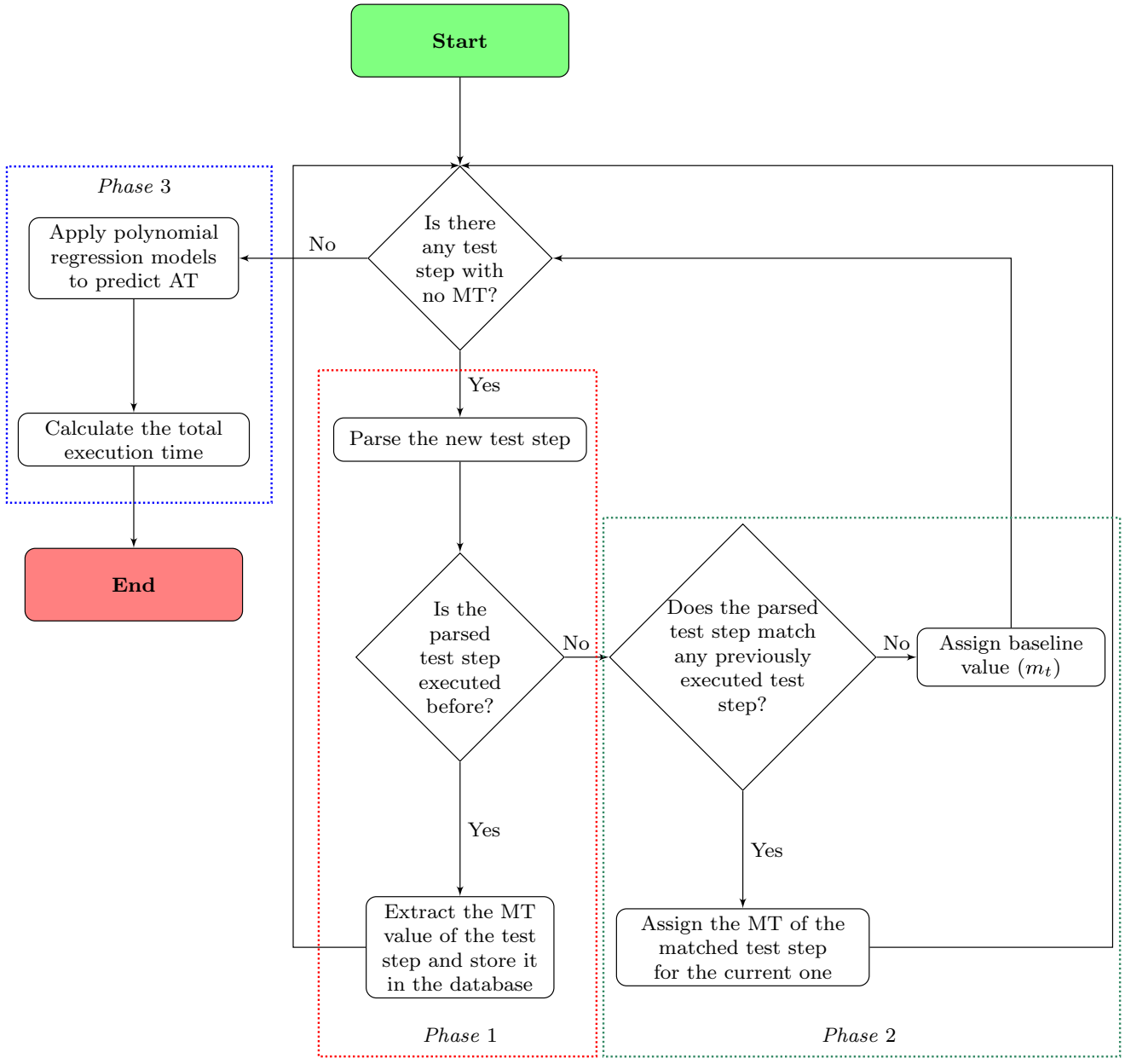

Fig. 1. The steps of the proposed approach.

```
VP (VBG removing)
NP (DT the) (JJ key) (NN card) (NN cab)
NP (CD 10) (NNS seconds)
```

Where VP, NP, PP, DT, JJ, CD, NNS denotes verb phrase, noun phrase, prepositional phrase, determiner, adjective, cardinal digit and noun plural respectively. However, different forms of a verb (bare form (VB), past tense (VBD), -ing form (VBG), etc.) are shown after parsing. *ESPRET* captures the *Verbs* and the unique nouns (NN) which comes after the VP as the *Objective Argument*. In the example above, *cab* is a duplicated word which is removed automatically and the following elements are saved in our database by *ESPRET* as:

`Logout A1 cab removing card`.

Moreover, the cardinal digits (if existing) are saved as the MT value for the corresponding *Verbs* and *Objective Argument*. To deal with some (possible) anomalous results, some of the key elements parsed by NLTK are adjusted in our database manually. Since the test specifications are written by testers with different experience levels and language skills, therefore the required effort for anomaly detection is dependent on the quality of the test specification.

3.2. The algorithm for estimating the maximum execution time

As the output of the previous phase, a set of verb and argument pairs that represent previously executed test steps are stored with their MT or both MT and AT values. In this phase, the test steps of other test cases are parsed and an MT value for them is estimated. Estimation of the MT values is done as follows.

The goal is to check whether a matching test step (i.e., pair of same *Verb* and *Argument*) has been executed before or not. To provide a clarification of the concept of MT and AT, we analyze the test step number 6 in Table 2, which is *Active cab 1*. This activation process may take different actual times, but the maximum allowed time for this process is, for example, 3 s. If this test step takes one more millisecond than 3 s for execution, the activation process will be stopped. However, the actual execution time is a

Table 2
A test specification example from the safety-critical train control management system - Bombardier Transportation.

Test case name:	Auxiliary compressor control	**Date:**	2018-01-20
Test case ID	**Test level (s)**	**Test Result**	**Comments**
3EST000233-3484 - RCM (v.1)	Sw/Hw Integration		
Test configuration			
TCMS baseline: TCMS 1.2.3.0			
Test rig: VCS Release 1.16.5			
VCS Platform 3.24.0			
Requirement(s)			
SRS-BHH-LineVolt1707			
SRS-BHH-Speed2051			
Tester ID			
BR490 – 1211			
Initial State			
No active cab			
Step	**Action**	**Reaction**	**Pass / Fail**
1	Ensure no cabs are active, enter train data entry view on IDU	Automatic dropping active (yes/no)	
2	Activate cab *A*2 lock and set signal braking mode from ATP to 109	Signal braking mode to IDU is set to 109	
3	Logout from *A*1 cab through removing the key card cab in 10 s	MIO-S “head light half-beam on right” = False	
4	Wait 20 s		
5	Reset dynamic brake in the train for 5 s	IDU in *B*1 car as On	
6	Active cab 1		
7	Clean up		

variable time, which might change per execution even on the same machine. Therefore, the AT values need to be predicted from the MT values. In the example above, executing *Active cab 1* on several machines might take different time. Therefore, both MT and AT values need to be considered. A time value will be assigned to these *Verb* and *Objective Argument* pairs as the maximum execution time. As part of this process, *ESPRET* first performs a log analysis on the previously executed test cases in our database. The goal is to check whether a matching test step (i.e., pair of same verb and argument) has been executed before or not. If a match is found, the MT value will be assigned for the newly parsed test step. In the case that *ESPRET* cannot find any matches from historical data for the parsed element, the baseline time value (m_t) as defined previously, is assigned for its MT. Knowing the maximum execution time (MT) for test cases can itself help testers and test managers for test case selection and prioritization. The test managers can schedule test cases for execution based on the maximum time that each test case takes. Therefore, *ESPRET* can be utilized as a supportive tool for making decisions on test scheduling.

The estimated MT values for test steps in this phase are used as independent variables for predicting the AT values through performing a regression analysis in the next phase. There are multiple reasons of our selection of independent and dependent variables. First, the values of MT are readily available to us and are not changeable before the first execution. There is no additional information available for the test cases' execution time before the first execution except the inserted timing information in the requirement and test specifications. Second, in the context of the safety critical real-time system (the train control management system), timing is of fundamental importance since missed deadlines can cause catastrophic accidents. Previous research on worst case execution time (WCET) analysis of tasks show that determination of upper bounds on execution time is necessary in the development and validation of real-time systems (Wilhelm et al., 2008). Since both AT and MT represent the same construct (i.e. time), they both are interesting candidates for variables in our context.

3.3. Regression analysis for prediction of the actual execution time

In this phase, in order to predict the actual execution time of test cases (those whose steps have not been executed before) a regression analysis is performed. Generally, regression models consider the relation between a dependent and an independent (explanatory) variable simultaneously, where the dependent variable is predicted by an independent variable. In the present work, the value of the actual execution time has been considered as a dependent variable which can be predicted by the value of the maximum execution time (independent value).

Using different types of regression models can provide a prediction of the actual execution time for test cases, in such a way that, the AT value is modeled based on the MT value. In our previous work (Tahvili et al., 2017), we have applied a linear regression approach for predicting the AT values, where we assumed that the relationship between each dependent (AT) and independent (MT) variable pair is linear or straight line. However, the AT and MT values do not necessarily exhibit a linear relationship and therefore adding more features to the regression model can provide a better fitting (Refaeilzadeh et al., 2009). Furthermore, analyzing results from several experiments indicate that the linear assumption in practice is rarely true and a polynomial curve fitting might suit the data points better. Note that the polynomials are not able to model thresholds and also observations at one range of the predictor might have a strong influence on what the model does at a different range (Harrell, 2001). Thus, to provide a more accurate prediction of the AT values, we utilize a higher degree polynomial regression and a multivariate adaptive regression splines (MARS) in the present work. MARS is a nonparametric regression technique which can model nonlinearities and produce continuous models with continuous derivatives (Friedman and Roosen, 1995). The following assumptions must hold for building a regression model:

Suppose $(t_1, t_1'), (t_2, t_2'), \ldots, (t_n, t_n')$ represents MT (independent) and AT (dependent) respectively for n test steps ($n \geq 3$), then a polynomial regression can be formulated as:

$$t' = \beta_0 + \beta_1 t + \beta_2 t^2 + \cdots + \beta_{m-1} t^{m-1} + \epsilon, \tag{1}$$

where $\beta_i \ \ (i = 0, \cdots, m-1, m \geq 2)$ are unknown parameters and ϵ is a Gaussian error term which is tested to be sufficiently close to a normal distribution with mean 0 and standard deviation 1.4 (see Figs. 2 and 3).

The β_i can be determined by the least squares method as:

$$L_{m-1} = \sum_{i=1}^{n} (\beta_0 + \beta_1 t_i + \beta_2 t_i^2 + \cdots + \beta_{m-1} t_i^{m-1} - t_i')^2 \tag{2}$$

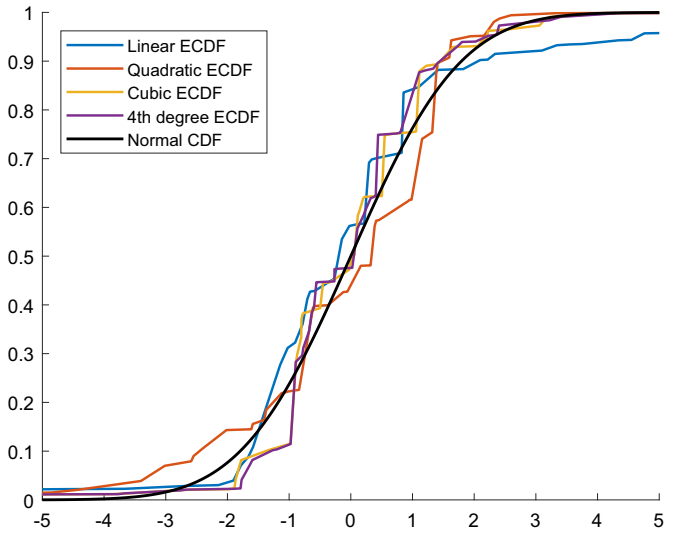

Fig. 2. Empirical CDF (ECDF) for the polynomial fit residuals compared to a normal CDF.

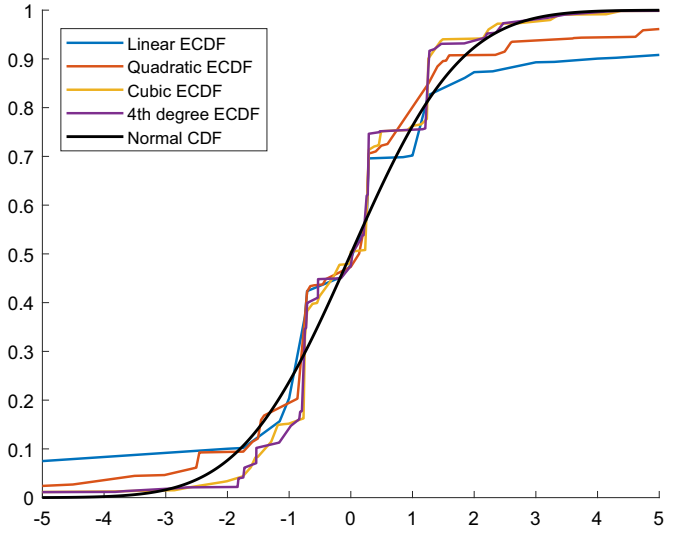

Fig. 3. Empirical CDF (ECDF) for the spline fit residuals compared to a normal CDF.

the condition for L_{m-1} to be a minimum is that:

$$\frac{\partial L_{m-1}}{\partial \beta_j} = 0 \tag{3}$$

for $(j = 0, \cdots, m-1)$, which means that, the amount of unknown parameters of β_i should be obtained to minimize L_{m-1}. Briefly AT (dependent variable) is modeled by MT (independent variable).

Lower degree polynomials have specific names, for example quadratic when $m = 2$, cubic when $m = 3$, and quartic (or 4th degree) when $m = 4$. To choose the optimal polynomial degree and to fit the optimal polynomial to the dataset, we measure the validation and the training errors for both polynomial and spline regression models by applying the cross-validation technique. Even though the fitted curve passes through almost all points, it is not necessary for it to perform well on unseen data (Chang et al., 2010). The reason behind this is that regression models are specialized to the structure in the training dataset. Applying the proposed regression models on unseen data may provide more accurate evaluation of the model's performance.

3.4. System architecture, implementation and database creation

The overall implementation architecture of *ESPRET* is given in Fig. 4. The historical data on previously executed test steps are collected from different sources such as: test specifications (e.g. in PDF, docx file), log files (e.g. XML), scripts (in case there exists a scripted version of a manual test case) and requirement specification (if the timing information exists in the corresponding requirement). In our implementation, a MySQL database has been created to store the information of test cases as well as the test steps constituting each test case. All test cases are extracted from the DOORS database as a separate Word file. For each test step of a test case, a record is inserted in the database, where the *Verb* and *Objective Argument* pair along with maximum and actual execution times of that step is recorded. As explained before, the verb and objective argument will be extracted through parsing the test specifications. *ESPRET* provides a user-friendly interface as shown in Fig. 5.

For this purpose, in *ESPRET*, we make use of NLTK (Natural Language Toolkit) in Python. The NLTK can also be used in deep learning through attempting to learn multiple levels of representation and abstraction that helps to make sense of different types of data such as natural texts (Dipayan, 2017). We however did not use the deep learning feature in our approach and is left as an interesting future work. For the parsing part, the NLTK Python platform is used in order to provide the entire Natural Language Processing (NLP) methodology such as tokenizing, stemming, chunking, chinking, lemmatizing and text classification (Bird et al., 2009). The NLTK helps us to parse various elements of each test step. Furthermore, a MySQL database has been created for capturing the parsed elements where a subset of raw test data from DOORS database are parsed, analyzed and later recorded in our separate database.

4. Empirical evaluation

In order to analyze the feasibility of *ESPRET*, we carried out an industrial case study at BT in Sweden, inspired by the guidelines of Runeson and Höst (Runeson and Höst, 2008) and specifically the way guidelines are followed in the paper by Engström et al. (2011). The software testing at BT is performed both manually and automatically, in such a way that the unit and regression testing is handled automatically, and only integration testing is performed manually. Moreover, the system level testing is handled by some approved suppliers to BT at the supplier's facility, separate from the normal testing process at BT.

4.1. Unit of analysis and procedure

The units of analysis in the case under study are manual test cases at the level of integration testing for a safety-critical train control subsystem at BT. The proposed approach is however not restricted to integration testing level and can be applied to any other manual testing procedure at any level of testing (e.g. unit, regression and system level) in other domains. The case study is performed in several steps:

(a) a total of 32 test suites (consists several test cases) for the underground subway train project in Stockholm (*C*30[2]) have been extracted from the DOORS database at BT,
(b) the extracted test cases are parsed by *ESPRET*. A total number of 3273 test steps have been recorded in our database,

[2] The *C*30 is the project name of the new subway carriages ordered by Stockholm public transport in 2008. In 2013, Bombardier was awarded contracts for wagons based on the MOVIA platform (BOMBARDIER, 2017).

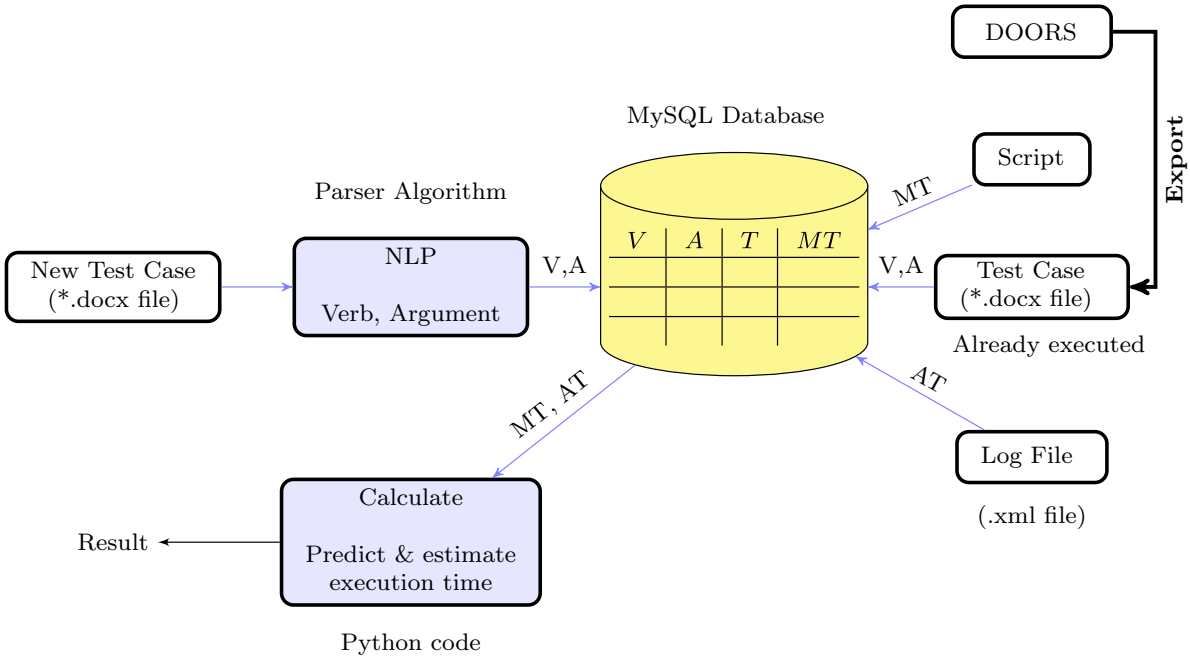

Fig. 4. The implementation architecture of *ESPRET*.

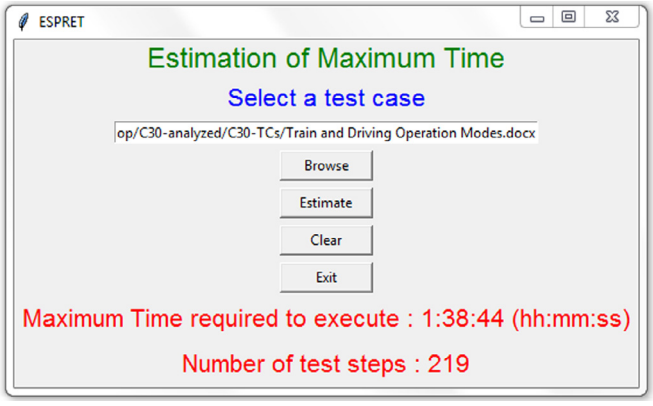

Fig. 5. The User Interface of *ESPRET*.

(c) more than 1100 test execution results (log files) have been analyzed,
(d) two types of regression models (polynomial and spline) are applied for predicting the actual execution time for manual test cases,
(e) the performance of the proposed regression models is evaluated through applying the cross-validation technique,
(f) the models' generalization ability is evaluated by applying the regression models on a set of unseen data.

4.2. Case study report

As is shown in Fig. 5, the user can easily insert a new test case specification file as an input, the expected output of *ESPRET* is the required maximum time for execution of the intended test case specification. Table 3 shows the test steps parsed by *ESPRET* for the test case illustrated in Fig. 5.

As we can see, *ESPRET* has assigned MT values for all test steps from the database. The maximum execution time in step 11 was not accessible from our database, which implies that this test step never got executed before. Therefore, the baseline time value

Table 3
The maximum execution time for the corresponding parsed test steps, exported from *ESPRET* database.

Step	Test Case ID	Verb	Objective Argument	MT(s)
1	3EST000236 – 8017	Login	system A	4
2	3EST000236 – 8017	Login	system B	23
3	3EST000236 – 8017	Login	system C	42
4	3EST000236 – 8017	Wait		20
5	3EST000236 – 8017	Turn on	MIO-S DX	4
6	3EST000236 – 8017	Set	MC, 100% traction	57
7	3EST000236 – 8017	Login	IDU maintainer, A1 cab	9
8	3EST000236 – 8017	Logout	removing key, A1 cab	9
9	3EST000236 – 8017	Open	door A1 cab	13
10	3EST000236 – 8017	Login	IDU, Driver, A2 cab	6
11	3EST000236 – 8017	Logout	removing key, A2 cab	3
12	3EST000236 – 8017	Login	full beam, button desk	23
⋮	⋮	⋮	⋮	⋮
217	3EST000236 – 8017	Active	emergency break	42
218	3EST000236 – 8017	Deactivate	A1 cab	23
219	3EST000236 – 8017	Press	emergency, stop button	25

($m_t = 3$) has been assigned for this test step. As stated in Table 3, there is no *Objective Argument* for test step 4, which is defined as 20 s waiting between steps 3 and 5. *ESPRET* captures the new test case specifications continuously and as soon as a test case is executed, the result of actual execution time will be entered in our database. Table 4 shows the maximum and actual execution times for some executed test cases (within the test suites) for the *C*30 project, consisting of the test suite id and names.

Since a test case will be executed usually more than one time, the AT column in Table 4 need to be updated continuously. On the other hand, in any successful execution, the value of MT should be bigger or equal to the AT value. Therefore, the MT values need to be compared to the AT values after each execution; in any case if *ESPRET* meets $MT \leq AT$, the value of MT will be updated with the AT value (making $MT = AT$). Note that the missing values (null) in the AT column in Table 4 represent those test cases which have not been executed yet.

In this paper, the results of 1187 test executions results for the *C*30 project test cases from a particular machine are captured to be

Table 4
Maximum and actual execution time for the *C*30 project test cases.

Nr	Test suites ID	Test suites name	MT	AT
1	3EST000236 – 6005	Drive and Brake functions	01: 03: 24	00: 57: 54
2	3EST000236 – 3456	Safe-Bogie	02: 10: 11	01: 54: 33
3	3EST000236 – 8003	Bogie	01: 08: 43	01: 00: 12
4	3EST000236 – 3455	Interior lighting	03: 06: 03	03: 01: 00
5	3EST000236 – 008	Safe-Exterior lights	01: 04: 10	01: 00: 24
6	3EST000236 – 2756	CCTV	02: 04: 39	01: 45: 12
7	3EST000236 – 8002	OSTS- Exterior lights functions	02: 00: 39	01: 54: 12
8	3EST000236 – 8010	Safe - Drive and Brake function	02: 04: 27	01: 54: 24
9	3EST000236 – 8013	Air Supply	02: 02: 33	01: 40: 58
10	3EST000236 – 8014	Safe-Reinitiate VCS	03: 03: 25	02: 48: 15
11	3EST000236 – 8019	Fire	03: 02: 02	03: 00: 24
12	3EST000236 – 0178	General Requirements	04: 05: 04	03: 44: 54
13	3EST000236 – 0179	ATP Function	01: 30: 00	01: 20: 11
14	3EST000236 – 8027	Emergency mode operation	02: 02: 16	02: 01: 45
15	3EST000236 – 8020	Vehicle coupler	02: 03: 26	01: 23: 00
16	3EST000236 – 8001	Outputs and IDU status	01: 03: 37	01: 00: 42
⋮	⋮	⋮	⋮	⋮
28	3EST000236 – 8029	Emergency mode operation	01: 04: 27	(null)
29	3EST000236 – 7011	DVS	01: 05: 07	(null)
30	3EST000236 – 2012	Exterior and interior access function	00: 02: 00	(null)
31	3EST000236 – 1415	HVITS	01: 01: 58	(null)
32	3EST000236 – 1703	TCMS functions	01: 07: 38	(null)

Table 5
The descriptive statistics of the *C*30 project dataset.

	MT	AT
Mean	7.20134793597304	3.77085088458298
Standard Error	0.413201966649295	0.242390555835765
Median	2	2
Mode	2	0
Standard Deviation	14.2359921341386	8.35104942565821
Pooled	2374	5.4861
1st quartile	2	0
2nd quartile	2	2
3rd quartile	6	3
Kurtosis	15.5245639682868	32.5290676464134
Skewness	3.80083649930494	5.19707054804772
Range	90	73
Minimum	0	0
Maximum	90	73
Count	1187	1187

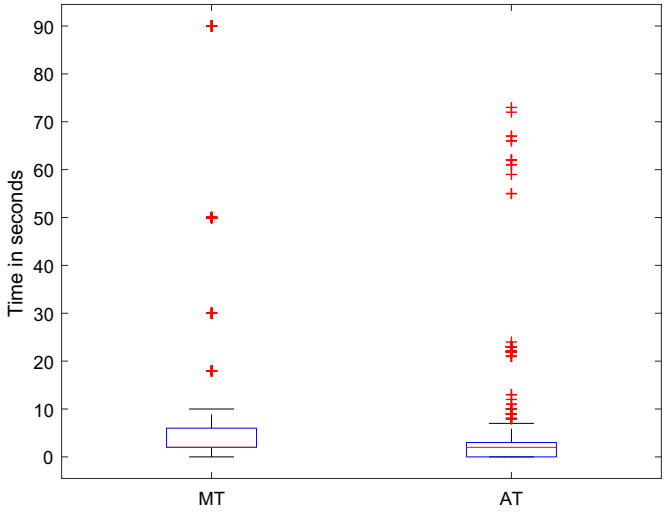

Fig. 6. The boxplot of the *C*30 dataset.

utilized for the regression analysis. Both polynomial and spline regression models are implemented in MATLAB (the source code implementation is available at Tahvili (2018)). The mentioned models are first fed with the labeled data, where the MT column is extracted from *ESPRET*'s database and the AT column is obtained through log analysis. To provide some basic information about the variables in the dataset (MT and AT) and also to highlight the potential relationships between the MT and AT, the descriptive statistics of the used dataset is summarized in Table 5.

Fig. 6 shows the box plot of the utilized dataset, which represents the distribution of the data based on the five-numbers summary: minimum, 1st quartile, median, 3rd quartile and maximum (see Table 5). In the both MT and AT boxes, the central mark indicates the median (is overlapped with the 1st quartile edge in the MT box) and the bottom and top edges of the box indicate the 25th and 75th percentiles.

The whiskers (the minimums and maximums outside of the 1st and 3rd quartiles) are plotted with lines and the outliers are individually depicted by using the red '+' symbol. Analyzing the whiskers in Fig. 6 shows that in most cases, the maximum execution time for the *C*30 project test cases is between 0 and 10 s whereas the actual execution time varies between 0 and 7 s. However, some test cases in the *C*30 project require more MT and thereby AT for execution. They are detected as outliers in Fig. 6. Hence the outliers in the *C*30 project dataset just represent the extreme cases; we have decided to include them as part of the data in our analysis so to represent those test cases that take a long execution time.

Figs. 7 and 8 represent the polynomial and spline regression fitting respectively. As illustrated in Fig. 7, we have applied 4 different regression techniques (linear, quadratic, cubic and 4th) on the test results dataset for recognizing the best fitting predictor. As can be seen in both Figs. 7 and 8, the higher degrees polynomial and spline are exhibiting a better fitting, compared with the liner regression model. Therefore, the quadratic, cubic and 4th degree regression models might provide a more accurate prediction for the AT values.

Moreover, the R-squared, which denotes the proportion of the variance in the dependent variable that is predictable from the independent variable (Devore, 2011), is calculated by Eq. (4) as fol-

Table 6
R-squared for polynomial and spline models and fit coefficients for polynomial models (Px and Sx are polynomial and spline fits of degree x).

	P1	P2	P3	P4	S1	S2	S3	S4
R^2	0.91289	0.95833	0.9703	0.97081	0.38723	0.91938	0.97095	0.9711
β_0	−0.26537	0.93525	−0.030871	−0.26497				
β_1	0.56048	0.21586	0.58813	0.7289				
β_2	0	0.0050369	−0.0082397	−0.022131				
β_3	0	0	0.00010596	0.00044052				
β_4	0	0	0	$-2.19E-06$				

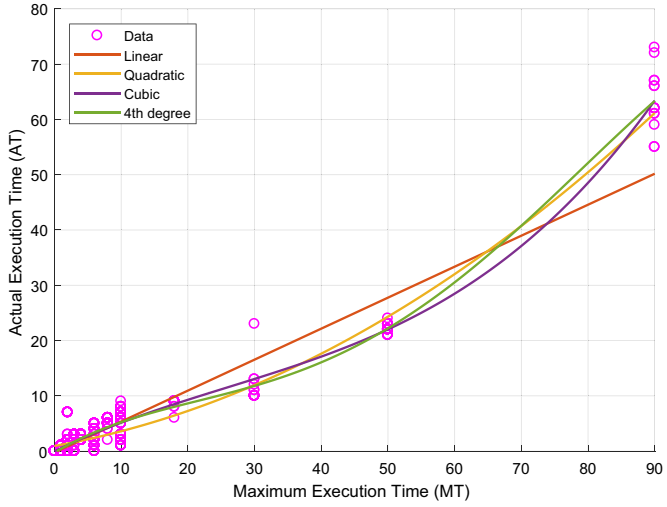

Fig. 7. Polynomial regression.

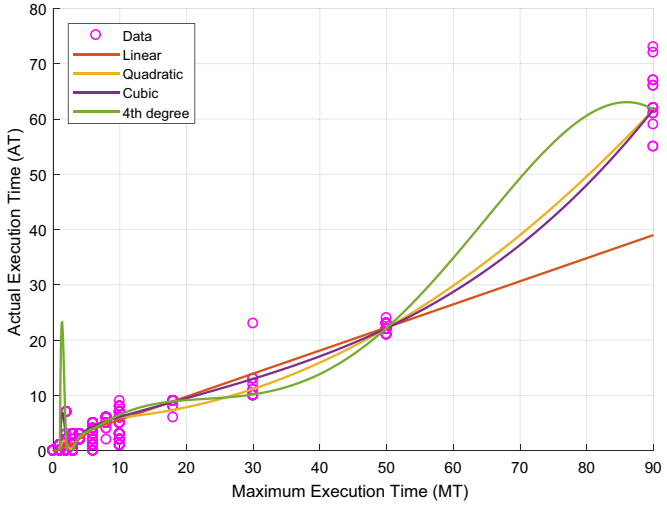

Fig. 8. Spline regression.

lows and given in Table 6.

$$R^2 \equiv 1 - \frac{\sum_i (AT_i - m(MT_i))^2}{\sum_i (AT_i - \text{MEAN}(AT))^2}, \tag{4}$$

The R-squared (R^2) is always between 0 and 1 and implies how close the data are to the fitted regression line. Table 6 represents the R-squared for polynomial and spline models and fit coefficients (β_i in Eq. (2)) for polynomial models.

According to Table 6, the R-squared for the both polynomial and spline models is higher than 0.9 in all cases except the linear spline. However, there is a large number of coefficients for the spline models which does not provide any additional information to the fitted regression spline model.

4.3. Model validation

To further analyze the performance of the regression models and to determine optimum parameter values, it is necessary to study the error rates of regression models and polynomial models (Muller et al., 2001). In a practical setting, error rates are estimated for a sample dataset, which poses validity threats to the study if the dataset is small (Jain et al., 2000). However, there is an essential risk of overfitting a regression model which reduces the model's generalizability outside the original dataset (Muller et al., 2001). It is therefore a standard practice to split a dataset into disjoint training and testing sets.

In this subsection, we measure the performance of the polynomial and spline regression models by estimating the prediction error, which provides the training accuracy over the training test set. In the most real cases, the expected prediction error cannot be calculated precisely and must therefore be estimated (Chen et al., 2012). There are several methods to accomplish this, such as hold-out validation, k-fold cross-validation and leave-one-out cross-validation which have been applied for developing appropriate estimators of the prediction error (Refaeilzadeh et al., 2009; Afzal et al., 2012). As mentioned earlier, test cases will be executed more than one time, continuously until the end of the project. Therefore, a large number of data points have been recorded by *ESPRET* for various executions. As it was mentioned earlier in this section, more than 1100 log files have been analyzed by us, which might increase the risk of capturing the noise of the data and also low bias but high variance and thereby can lead to overfitting. On the other hand, selecting a small dataset for training the model might cause under-fitting, and both cases can lead to poor model performance. In the present work, we apply k-fold cross-validation technique to assessing the performance of prediction models. Cross-validation is a technique to test a model on its predicative performance. In the k-fold cross-validation, the whole dataset should be divided randomly into k non-overlapped subsets of roughly equal size, let $D_1, D_2, \ldots, D_K$, then for $k = 1, \ldots, k$ a union of subsets $\bigcup_{i=1:i\neq k}^{K} D_i$ is utilized for fitting a model and the remaining subset D_k for validation (Chen et al., 2012). The average of the k estimates of the prediction errors from each loop presents the final prediction error. The partitions used in cross-validation help to simulate an independent dataset and get a better assessment of a model's predictive performance. In other words, after fitting the model on to the training data (partitions), the performance will be measured against each validation set and then averaged, gaining a better assessment of how the model will perform when asked to predict for new observations. The number of partitions to construct depends on the number of observations in the sample dataset as well as the decision made regarding the bias-variance trade-off, with more partitions leading to a smaller bias but a higher variance (Refaeilzadeh et al., 2009). While a

model may minimize the Mean Squared Error (MSE) on the training data, it can be optimistic in its predictive error. The square root of the MSE yields the Root Mean Square Error (RMSE), which provides a better measure of goodness of fit than a correlation coefficient (Martens and Martens, 2001).

For a dataset (*MT, AT*) with n data points and a model m, the RMSE of the model m, is defined as:

$$RMSE = \sqrt{\frac{\sum_i (m(MT_i) - AT_i)^2}{n}}, \tag{5}$$

where the vector norm is the Euclidean norm. For more than one iteration and partition, we define Mean Validation Error (MVE) as:

$$MVE = \frac{1}{N \cdot K} \sum_{i=1}^{N} \sum_{j=1}^{K} \sqrt{\frac{\left(m_{i,j}(MT_{i,j}) - AT_{i,j}\right)^2}{n_{i,j}}} \tag{6}$$

where N is the number of random partitionings (iterations) of (*MT, AT*), K is the number of partitions (folds) in one partitioning, ($MT_{i,j}$, $AT_{i,j}$) is the jth partition of the ith partitioning (the validation dataset), $n_{i,j}$ is the size of the of the validation dataset and $m_{i,j}$ is the fitted model of the ith partitioning when the jth partition is excluded.

Moreover, the Mean Training Error (MTE) can be calculated as:

$$MTE = \frac{1}{N \cdot K} \sum_{i=1}^{N} \sum_{j=1}^{K} \sqrt{\frac{\left(m_{i,j}(\overline{MT}_{i,j}) - \overline{AT}_{i,j}\right)^2}{\overline{n}_{i,j}}} \tag{7}$$

where $\left(\overline{MT}_{i,j}, \overline{AT}_{i,j}\right)$ is the complementary dataset to ($MT_{i,j}$, $AT_{i,j}$) (the training dataset) and $\overline{n}_{i,j}$ is the size of the training dataset.

We need to consider that both MVE and MTE represents average of RMSE for all randomly selected partitionings. In other words, by using MVE and MTE, we are computing the RMSEs in every single partition for every single K. The unit of measurement is seconds for both MVE and RMSE, where in MVE we measure the average of RMSE on the lot of randomly selected cross-validation datasets. The mean squared error (MSE), root mean squared error (RMSE) or mean training error (MTE) could be used to summarize the errors.

In addition, the k-fold cross-validation still suffers from issues of high variance (Alves et al., 2013). Since, it is not certain which data points will end up in the validation set, the result might be entirely different for different sets. Therefore, in the present work, we perform MVE and MTE and we assume $k = 5$ for partitioning 1187 data points. However, at each iteration a different result set might be obtained, thus we run the model with 10, 100 and 1000 iterations, in order to find the most accurate values for both MVE and MTE.

Figs. 9 and 10 represent the result of the k-fold cross-validation for computing the prediction error for polynomial and spline regression models. The RMSE is measured for the degrees on polynomials and spline. The unit of measurement for RMSE is seconds (Y-axis) and X-axis represents the polynomial degree (Fig. 9). As we can see in Fig. 9, the cross-validation error decreased sharply from 2.5 s of time between degree 1 and degree 3 and it goes asymptotically to 1.4 s. We also observe a significant reduction for the spline's prediction error in Fig. 10 but it does not tend to zero and it increases again from degree 4 to 5. According to Fig. 9, a degree 3 or 4 polynomial seems to fit the model the closest while also holding the most predictive power. Every blue line in the Figs. 9 and 10 is a mean of the five-different root mean square errors of the cross-validation test for each partition.

The resulting evaluation metric is known as model accuracy, which is a better estimate of *out of sample* performance than training accuracy because we fitted and tested the model on different

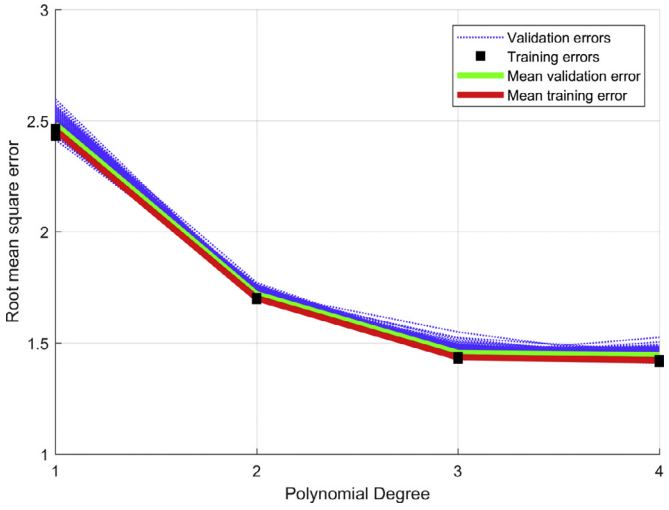

Fig. 9. Polynomial cross-validation. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

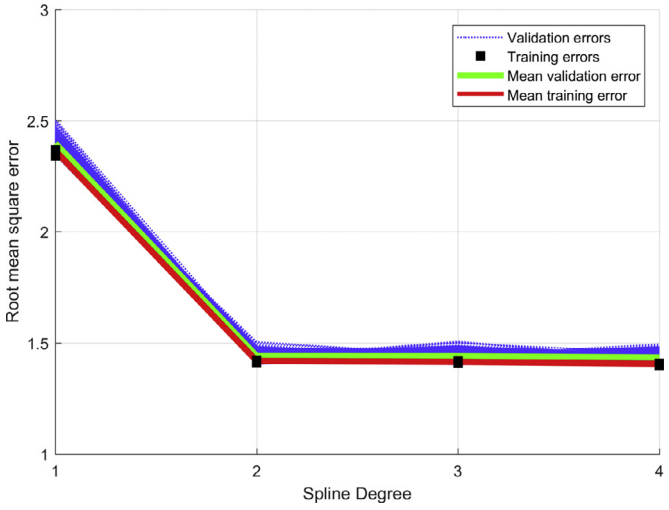

Fig. 10. Spline cross-validation. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

sets of data. The model accuracy does not reward overly complex models and thus it helps us to avoid overfitting. However, there is a drawback to the train-test split procedure. It turns out that the model accuracy is a high variance estimate of *out of sample* accuracy, which means the model accuracy can change a lot, depending on which observations happen to be in the testing set.

Table 7 shows the mean cross-validation and training error for 10, 100 and 1000 iterations which are performed to approaching a more accurate value for MVE and MTE.

According to Table 7, the spline regression models are provided less values for both mean validation and mean training errors. Furthermore, both MVE and MTE are decreased when a higher degree is applied on the dataset. In the experiments, a small increase in the mean validation error was further observed for fifth degree polynomials and splines. The differences between 10, 100 and 1000 iterations are small, and no significant change is observed as the number of iterations increased.

As Table 7 represents, 100 and 1000 iterations have same values in some degrees, but both are better than 10.

Table 7
Mean validation error (MVE) and mean training error (MTE) for various iterations.

Number of iterations (n)		Polynomial		Spline	
		MVE	MTE	MVE	MTE
10	Linear	2.4713	2.4588	2.4029	2.3643
	Quadratic	1.7190	1.7014	1.4422	1.4197
	Cubic	1.4540	1.4357	1.4407	1.4152
	4th degree	1.4511	1.4219	1.4288	1.4071
100	Linear	2.4811	2.4575	2.3955	2.3644
	Quadratic	1.7183	1.7014	1.4402	1.4194
	Cubic	1.4554	1.4356	1.4398	1.4155
	4th degree	1.4476	1.4224	1.4347	1.4066
1000	Linear	2.4833	2.4576	2.3982	2.3637
	Quadratic	1.7193	1.7013	1.4412	1.4194
	Cubic	1.4552	1.4356	1.4409	1.4153
	4th degree	1.4456	1.4227	1.4353	1.4065

4.4. Model evaluation using unseen data

The ability to perform well on unseen data is called generalization and is the desirable characteristic of any demanded model (Chen et al., 2012). To achieve the previously specified goal, we trained the mentioned regression models on the labeled data from the *C*30 project at Bombardier to allow the models to develop quantitative relationships between MT and AT. In this subsection, we evaluate the trained regression models, for their ability to predict AT using unseen data not part of the training set. The unseen data is gathered from another ongoing testing project at Bombardier Transportation, called *BR*490[3] project.

Evaluating the predictive performance of models using unseen data is a critical step in a model development, which can be performed by measuring different factors, e.g. the *percentage prediction error*, the *absolute and relative predictive performance* (Guang et al., 1995). In this subsection, we measure the predictive performance of the polynomial and spline regression models on the unseen data, though applying the *percentage of predictions error* (ε) using the following equation (Guang et al., 1995):

$$\varepsilon = \frac{|Predicted\ time - Recorded\ time|}{Recorded\ time} \times 100 \quad (8)$$

where the predicted time is forecasted by the polynomial and spline regression models and the recorded time is captured after execution from the system log files. As stated before, the actual execution time, is a machine dependent value and it varies from one machine to another. To avoid any confusion, we just analyzed the results of passed test cases in a particular machine (machine *A*). Table 8 represents the results of the predicted AT values for 50 unseen test steps from the *BR*490 project, where the AT values are predicted using both polynomial and spline regression models in various degrees. The maximum execution time for the test cases is an input to the regression models, which are already trained by the labeled dataset from *C*30 project (Subsection 4.3). Moreover, the recorded actual execution time for the new dataset (unseen data) has been obtained from the log files (inserted in the *Recorded AT* column in Table 8) from machine *A*. However, the inserted data in *Recorded AT* column can be different on any other machine. It was mentioned before that, the maximum execution time (inserted in the *MT* column) should be greater or equal to the actual execution time (inserted in *Recorded AT* column), comparing these two columns in Table 8 indicates that the condition is true. As we can see in Table 8, the predicted values for AT are different by using different regression models.

[3] The *BR*490 series is an electric rail car specifically for the S-Bahn Hamburg GmbH network in production at Bombardier Hennigsdorf facility.

As we observed in Table 7, the 4th degree and the cubic spline regression models respectively have a less MVE value compared with the linear and quadratic in all three iterations. In fact, the spline regression model in degree 4 fits the training dataset better for the *C*30 project. In order to identify the best fitting regression model for the *BR*490 project, the percentage of prediction error (ε) is measured on the prediction results in Table 8 by using Eq. (8). Table 9 shows the percentage of prediction errors (ε), which can be utilized to compare the accuracy of the predicted time.

According to Table 9, the mean percentage of prediction error lies between 30% – 40%, which has been confirmed as good enough by our industrial partner for estimation purposes. Note that the best predicting model is dependent on the nature of the data and since the actual execution time is a machine-dependent value and might change for each execution, there is some inherent uncertainty in the prediction. The expected percentage of the prediction error can be more precisely estimated with the addition of more execution results. Moreover, the upper bound on the values of MT and AT adds to the uncertainty in the prediction results. The *C*30 project has maximum execution times between 0 and 90 s (see Table 5), whereas the maximum execution time for the *BR*490 project can be as large as 600 s. The differences between MT and AT values is another important factor in the prediction accuracy. The performance of the regression models for a dataset which has a close relationship in the dependent and independent variables might be different. Therefore, for a dataset such as *C*30 project the 4 degree spline regression model shows the best performance but not on the *BR*490.

5. Threats to validity

In this section, we discuss the validity threats, the research limitations and challenges in conducting this study.

- **Construct validity** addresses if the study measures what we intend it to measure (Robson, 2011). The largest threat to construct validity in this study is using the time-frame of test cases for estimating the maximum execution time. The overall execution time of a manual test case is a sum of several factors such as skill of testers, times for the installation process, the setup time for preparing the testing environment and some other human factors. All mentioned factors are time consuming and an accurate time measurement of such factors may yield better estimation of the maximum execution time for manual test cases.
- **Internal validity** addresses the conclusions of the study (Runeson and Höst, 2008). In order to reduce the threats to internal validity, multiple test process artifacts such as test specification, requirement specification, test scripts and test logs are used. One of the threats is related to the structure of test cases in the study. In this study, the natural language processing was performed on a set of well-defined test cases, which are parsed and analyzed quickly by the NLTK. For a more complicated structure of test cases, NLTK may or may not perform accordingly, which can impact the captured data in the database.
- **External validity** refers to the generalization of the findings (Wohlin et al., 2000). *ESPRET* has been applied on a limited number of test cases for two industrial projects at Bombardier Transportation. However, the findings should be applicable to similar contexts. We have provided as much context information as possible to enable comparison to other proposed approaches and studies. Moreover, the regression analysis (*phase 3*) is performed on a set of unseen data, in such way that the regression models are trained by the dataset from *C*30 project and tested on the dataset from *BR*490 project.

Table 8
Maximum execution time and predicted actual execution time for the *BR*490 project by using different degrees of polynomial and spline. The *recorded AT* column represents the actual time that every test step took for execution on a particular machine (*A*) at Bombardier Transportation.

Nr.	MT	Predicted AT (Polynomial)				Predicted AT (Spline)				Recorded AT
		Linear	Quadratic	Cubic	4th degree	Linear	Quadratic	Cubic	4th degree	Machine *A*
1	320	179	586	282	106	225	669	268	114	317
2	18	9.82	6.45	8.5	8.02	9.94	7.48	9.07	9.38	16
3	45	25	20.8	19.4	18.2	29.2	21.5	20.3	19.5	31
4	48	26.6	22.9	20.9	20.8	31.3	23.6	21.9	21.3	45
5	173	96.7	189	404	220	120	208	394	224	164
6	18	9.82	6.45	8.5	8.02	9.94	7.48	9.07	9.38	10
7	69	38.4	39.8	36.1	39.7	46.2	41.3	37.5	40	55
8	19	10.4	6.85	8.9	8.33	10.6	7.84	9.46	9.7	18
9	210	117	268	741	110	147	300	716	106	189
10	19	10.4	6.85	8.9	8.33	10.6	7.84	9.46	9.7	16
11	400	224	893	570	312	282	103	543	339	366
12	250	140	370	129	288	175	417	123	307	234
13	180	101	203	457	319	125	224	445	329	164
14	59	32.8	31.2	27.7	29.6	39.1	32.2	29	29.8	55
15	65	36.2	36.2	32.5	35.4	43.4	37.5	33.8	35.7	62
16	600	336	194	203	197	424	227	191	216	588
17	120	67	99.4	135	75	82.5	107	135	79.6	109
18	45	25	20.8	19.4	18.9	29.2	21.5	20.3	19.5	39
19	77	42.9	47.7	44.8	48.7	51.9	49.5	46.3	49.3	74
20	100	55.8	72.9	82.3	72.5	68.3	77.4	83.5	75	81
21	90	50.2	61.2	63.4	63.3	61.2	64.5	64.8	64.8	80
22	99	55.2	71.7	80.3	71.8	67.6	76	81.5	74.2	90
23	29	16	11.4	12.7	11.5	17.8	12.1	13.3	12.7	27
24	14	7.58	4.94	6.88	6.73	7.09	6.16	7.45	7.99	10
25	65	36.2	36.2	32.5	35.4	43.4	37.5	33.8	35.7	40
26	52	28.2	25.8	23.2	23.7	34.1	26.5	24.3	24.1	44
27	590	330	188	192	183	417	223	185	250	587
28	9	4.78	3.29	4.67	4.81	3.53	4.79	5.29	5.8	6
29	11	5.9	3.92	8.58	5.63	4.95	5.3	6.18	6.75	8
30	124	69.2	105	148	71.3	85.4	113	148	76.3	108
31	90	50.2	61.2	63.4	63.3	61.2	64.5	64.8	64.8	74
32	66	36.7	37.1	33.4	36.5	44.1	38.5	34.7	36.7	60
33	109	60.8	84.3	103	77.2	74.7	90	104	80.7	90
34	36	19.9	15.2	15.4	14.2	22.7	15.8	16.2	15.1	28
35	29	16	11.4	12.7	11.5	17.8	12.1	13.3	12.7	17
36	26	14.3	9.95	11.6	10.5	15.6	10.7	12.2	11.8	14
37	14	7.58	4.94	6.88	6.73	7.09	6.16	7.45	7.99	13
38	74	41.2	44.5	41.3	45.2	49.8	46.4	42.8	45.7	44
39	8	4.22	2.98	4.2	4.37	2.82	4.55	4.83	5.28	6
40	9	4.78	3.29	4.67	4.81	3.53	4.79	5.29	5.8	4
41	11	5.9	3.92	5.58	5.63	4.95	5.3	6.18	6.75	9
42	10	5.34	3.6	5.13	5.23	4.24	5.04	5.74	6.29	4
43	15	8.14	5.31	7.29	7.06	7.8	6.48	7.86	8.36	10
44	73	40.6	43.5	40.2	44.1	49.1	45.3	41.7	44.5	70
45	60	33.4	32	28.5	30.5	39.8	33.1	29.7	30.8	50
46	56	31.1	28.8	25.7	26.9	37	29.7	26.8	27.2	47
47	61	33.9	32	29.2	31.5	40.5	33.9	30.5	31.7	56
48	88	49.1	58.9	60.1	61.2	59.8	62	61.6	62.5	81
49	35	19.4	14.7	15	13.7	22	15.3	15.7	14.7	31
50	30	16.5	11.9	13.1	11.8	18.5	12.6	13.7	13	25

Table 9
The mean percentage of prediction error (ε) for polynomial and spline regression models.

The mean percentage of prediction error		
Degree	Polynomial	Spline
Linear	32.33	24.83
Quadratic	40.32	36.23
Cubic	45.23	43.32
4th degree	37.18	34.09

- **Conclusion validity** addresses the factors which can affect the observations and may lead to an inaccurate conclusion (Cozby and Rawn, 2012). In other words, an inadequate data analysis can yield conclusions that a proper analysis of the data would not have supported (Drost, 2011). In order to address this issue, we have measured both training and cross-validation errors on the training dataset (the *C*30 project) and have further measured the generalization error on an unseen dataset (the *B*490 project). Furthermore, noise removal has been performed on the training data. The percentage prediction error discussed in Table 9 indicates that the proposed regression analysis on unseen data is acceptable and that the approach is likely to generalize well to other projects within the same domain.
- **Reliability** addresses the repeatability of the study (Runeson and Höst, 2008). Occasionally, the NLTK is not able to parse all the characters in a sentence correctly, which is related to the positions of the components of a sentence. Sometimes if a test step starts with a verb, NLTK identifies the verb as a noun. For instance, in the test step: *'Wait for 23 s'*, the word *Wait* can sometimes be identified as a

Table 10
The most frequently used verbs in the test cases.

Nr	Verb	Count	Nr	Verb	Count	Nr	Verb	Count
1	Set	591	6	Force	155	11	Lock	125
2	Active	287	7	Wait	149	12	Try	120
3	Turn	183	8	Turn on	142	13	Open	118
4	Remove	172	9	Deactivate	139	14	Push	118
5	Press	169	10	Drive	131	15	Restore	111

noun (and sometimes as a verb) by the NLTK. Our parsing algorithm in such a case, recognizes the first word of a sentence as a verb and the remaining sentence is parsed to identify nouns. The obtained nouns are then used as arguments which support the main verb.

In addition, most of the language parsing techniques have performance issues when a large set of data is processed. There are demerits in the available tools for natural language processing. Among them, the NLTK is comparatively better in processing and parsing natural language. One of the advantages of using NLTK is that it supports parsing multiple languages, but still it exhibits performance issue for parsing a large set of test specifications.

6. Discussion and future extensions

Our goal is to design, implement and evaluate a tool that estimates the execution time of test cases based on test specifications. To this end, we make the following contributions:

- We have proposed an NLP-based approach to parse critical elements of each test step. The maximum execution time for various parsed test steps have been extracted by analyzing multiple test process artifacts. The proposed approach has been implemented as a tool called *ESPRET*.
- The evaluation of the proposed approach was performed through applying *ESPRET* on an industrial testing project (*C*30) in a safety-critical train control management subsystem (Bombardier Transportation). Furthermore, the actual execution time for *C*30 project's test cases, has been predicted by performing a regression analysis, using polynomial and spline regression models.
- The prediction error of the proposed regression models has been measured by *k*-fold cross-validation method.
- The proposed regression models are applied on a set of unseen data, using another industrial testing project *BR*490 at Bombardier Transportation. Moreover, the percentage of the prediction error has been measured to show the performance of the methods on the unseen data.

Knowing the execution time of test cases provides an overview of the overall time for testing a system, which can lead to on-time delivery of the final product (Tahvili et al., 2015). Running *ESPRET* in the early stage of a testing process can give an opportunity for the test managers and the testers to prioritize and schedule test cases for execution. One of the improvements we are seeking is the more accurate, easy and faster discovery of the relationship between *Verb* and *Objective Argument* in our database. Perhaps association rule mining is applicable here. Association rules are based on the criteria support and confidence to identify the most important relationships. Support is an indicator to show how frequently an item appears in the database and confidence indicates the number of times it has been found (Hipp et al., 2000). A total number of 149 unique verbs have been recorded by *ESPRET* until today. Table 10 shows the number of 15 most frequent verbs used in the test specifications at BT.

In the future, *ESPRET* will be able to predict the related *Objective Argument* by analyzing the presented *Verbs* in Table 10 in the database. In other words, when *ESPRET* faces a verb in a new test step (such as verb *'Set'* in the Table 10), some suggestions (such as *'temperature'* or *'pressure'*) which might be a match for *'Set'* will appear automatically.

Creating a library for test cases and synonyms, is one of the potential aspect that can make *ESPRET* more efficient. The recorded parsed elements in the database can be classified into different groups in order to be replaced by the synonyms. If a new element is detected and encountered, *ESPRET* will be able to compare this element with a similar (synonym) element in the database.

As stated before, the overall time for executing a test case manually by testers depends on both system properties and testers skills. For instance, the required time that an inexperienced tester needs to execute a test case (e.g., finding a signal in the simulation software) is higher than an experienced tester. In this work, we just focused on the required time that a system takes to execute test cases by analyzing the test specifications. In the future, some other effective parameters such as the system properties, testing environment and human factors (education, experience, skills, working environment, etc.) can be considered for estimating the overall execution time for manual test cases. Considering that the cost for executing test cases is a function of execution time, we may assess the required cost for testing by using this study. We already have started working on the cost estimation for performing the testing projects at Bombardier Transportation.

In another usage scenario, *ESPRET* may even play a supporting role for test automation. One of the classical mistakes of a test automation team is not choosing right test cases for automation. At Bombardier Transportation, the test specifications are used as a starting point for the generation of test automation scripts. The automated tests serve as support for manual testing in a smaller trial set of tests. In the scripted version of a test specification, the value of maximum time is assigned as a timeout value. The scripted timeout value specifies the maximum amount of time that a script can run before it is terminated. If the processing time of the script exceeds the maximum value that has been designated for the scripted timeout value, an error will be generated (Yamamoto, 2013). Furthermore, by analyzing the recorded test step in the *ESPRET*'s database, we can recognize which test steps are frequently used into several test cases.

The test automation scripts are only a support device to manual testing and at Bombardier Transportation, the test specifications are used as an initial information for generation of test scripts. The test specifications used in this study are well-structured. As explained in Section 5, sometimes the NLTK has a performance issue on parsing. The SpaCy toolkit (Industrial-Strength Natural Language Processing), is an open-source software library for advanced natural language processing and it would be a potential candidate to replace the NLTK.

By computing the percentage of prediction error for the proposed regression models, we have a general overview of how well the regression techniques will perform on a new dataset. However, other prediction techniques can be examined. For instance, the neural networks (NNS) can be utilized for the prediction and forecasting of AT variables. Moreover, the Wilcoxon signed-rank (Dietterich, 1998) test or *t*-test can be utilized as way to com-

pare the performance of different prediction methods in the future. More experimentation with testing the accuracy of ESPRET for estimating MT is part of future work, e.g., experimenting ESPRET when a certain number of test steps with baseline MT values exist.

7. Conclusion

Estimation and prediction of the execution time for test cases can potentially play a vital role in test case selection, prioritization, scheduling and automation. Since the required time for running test cases is not available before execution, we need to estimate a time value for test cases to enable their scheduling. In this paper, we introduced, applied and evaluated our proposed approach and tool, *ESPRET*, for estimating and predicting the execution time for manual test cases. Our proposed approach takes multiple test process artifacts such as test specification, requirement specification, test scripts and test logs as input. *ESPRET* is designed based on parsing of textual test specifications. *ESPRET* reads the test specification (e.g., in the format of word document files) and stores the *Verbs* and *Objective Argument* via parsing of all test steps. Our tool has been implemented using NLTK (the Natural Language Toolkit), written in Python. Moreover, a MySQL database has been created for recording the parsed elements. By continuous monitoring of the historical execution data, *ESPRET* assigns a time value for each test step. *ESPRET* sums up the execution time of each test step and produces the maximum execution time for the corresponding test case. In order to predict the actual execution time for manual test cases, different degrees of the polynomial and spline regression models have been applied in this work. In short, a relationship model between the AT and MT values, as dependent and explanatory variables respectively, is established using regression models, which is utilized for predicting the AT values of test cases with no execution records. Moreover, the validation and testing errors of the proposed regression models are measured by using *k*-fold cross-validation. Both polynomial and spline regression models have been trained with the dataset form the *C*30 project and then the AT values for the *BR*490 project are predicted. Our empirical evaluations at Bombardier Transportation and analysis of the results of two industrial projects show that *ESPRET* is an applicable tool for estimating and predicting the execution time for manual test cases. Finally, *ESPRET* can be utilized as a supportive tool for making decisions on prioritization, selection, and general scheduling of test cases based on their estimated execution time values.

Acknowledgments

ECSEL & VINNOVA (through projects MegaM@RT2 & TESTOMAT) and the Swedish Knowledge Foundation (through the projects TOCSYC (20130085) and TestMine (20160139)) have supported this work. The authors are also thankful to Jonas Österberg and Christian Hurtig at Mälardalen University, Pasqualina Potena at RISE SICS Västerås, Markus Einarson, Ola Sellin and Mahdi Sarabi at Bombardier Transportation in Västerås Sweden and also Mohammad Mehrabi for their support during the project.

References

Afzal, W., Alone, S., Glocksien, K., Torkar, R., 2016. Software test process improvement approaches. J. Syst. Soft. 111 (C), 1–33.
Afzal, W., Ghazi, A.N., Itkonen, J., Torkar, R., Andrews, A., Bhatti, K., 2015. An experiment on the effectiveness and efficiency of exploratory testing. Empirical Softw. Eng. 20 (3), 844–878.
Afzal, W., Torkar, R., 2008. Incorporating metrics in an organizational test strategy. In: 2008 IEEE International Conference on Software Testing Verification and Validation Workshop.
Afzal, W., Torkar, R., Feldt, R., 2012. Resampling methods in software quality classification. Int. J. Software Eng. Knowl. Eng. 22 (02), 203–223.
de Almeida, E.R.C., de Abreu, B.T., Moraes, R., 2009. An alternative approach to test effort estimation based on use cases. In: 2009 International Conference on Software Testing Verification and Validation, pp. 279–288. doi:10.1109/ICST.2009.31.
Alves, E.L.G., Machado, P.D.L., Massoni, T., Santos, S.T.C., 2013. A refactoring-based approach for test case selection and prioritization. In: 8th International Workshop on Automation of Software Test (AST), pp. 93–99. doi:10.1109/IWAST.2013.6595798.
Angelis, L., Stamelos, I., Morisio, M., 2001. Building a software cost estimation model based on categorical data. In: Proceedings Seventh International Software Metrics Symposium, pp. 4–15. doi:10.1109/METRIC.2001.915511.
Aranha, E., Borba, P., 2007. An estimation model for test execution effort. First International Symposium on Empirical Software Engineering and Measurement doi:10.1109/ESEM.2007.73.
Bird, S., Klein, E., Loper, E., 2009. Natural Language Processing with Python. O'Reilly.
BOMBARDIER, 2017. Bombardier Wins Order to Supply New Generation MOVIA Metro Fleet for Stockholm. Technical Report.
Bush, A., Baladi, G., on Road, A.C.D.-., Materials, P., on Soil, A.C.D.-., Rock, 1989. Nondestructive Testing of Pavements and Backcalculation of Moduli. ASTM STP 1026. ASTM.
Casey, V., 2008. Software Testing and Global Industry: Future Paradigms. Cambridge Scholars Publisher.
Chang, Y., Hsieh, C., Chang, K., Ringgaard, M., Lin, C., 2010. Training and testing low-degree polynomial data mappings via linear svm. J. Mach. Learn. Res. 11, 1471–1490.
Chen, C., Wang, Y., Chang, Y., Ricanek, K., 2012. Sensitivity Analysis with Cross-Validation for Feature Selection and Manifold Learning. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 458–467 doi:10.1007/978-3-642-31346-252.
Cozby, P., Rawn, C., 2012. Methods in Behavioural Research. McGraw-Hill Ryerson.
Devore, J., 2011. Probability and Statistics for Engineering and the Sciences. Cengage Learning.
Dietterich, T.G., 1998. Approximate statistical tests for comparing supervised classification learning algorithms. Neural Comput. 10 (7), 1895–1923. doi:10.1162/089976698300017197.
Dipayan, D., 2017. Deep Learning with Hadoop: Build, Implement and Scale distributed deep learning models for large-scale datasets.
Drost, E.A., 2011. Validity and reliability in social science research. Educ. Res. Perspect. 38 (1).
Dustin, E., Garrett, T., Gauf, B., 2009. Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality. Pearson Education.
Engström, E., Runeson, P., 2011. Decision support for test management and scope selection in a software product line context. In: 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, pp. 262–265. doi:10.1109/ICSTW.2011.80.
Engström, E., Runeson, P., Ljung, A., 2011. Improving regression testing transparency and efficiency with history-based prioritization – an industrial case study. In: 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation, pp. 367–376. doi:10.1109/ICST.2011.27.
Felderer, M., Ramler, R., 2014. Integrating Risk-based Testing in Industrial Test Processes, 22. Kluwer Academic Publishers, pp. 543–575.
Flemström, D., Potena, P., Sundmark, D., Afzal, W., Bohlin, M., 2018. Similarity-based prioritization of test case automation. Softw. Qual. J. doi:10.1007/s11219-017-9401-7.
Friedman, J.H., Roosen, C.B., 1995. An introduction to multivariate adaptive regression splines. Stat. Methods Med. Res. 4 (3), 197–217. doi:10.1177/096228029500400303.
Garousi, V., Felderer, M., Fernandes, J.a.M., Pfahl, D., Mäntylä, M.V., 2017a. Industry-academia collaborations in software engineering: An empirical analysis of challenges, patterns and anti-patterns in research projects. In: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, pp. 224–229. doi:10.1145/3084226.3084279.
Garousi, V., Felderer, M., Kuhrmann, M., Herkiloğlu, K., 2017b. What industry wants from academia in software testing? hearing practitioners' opinions. In: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering. ACM, pp. 65–69. doi:10.1145/3084226.3084264.
Guang, W., Baraldo, M., Furlanut, M., 1995. Calculating percentage prediction error: a user's note. Pharmacol. Res. 32 (4), 241–248. doi:10.1016/S1043-6618(05)80029-5.
Hao, D., Zhang, L., Mei, H., 2016. Test-case prioritization: achievements and challenges. Front. Comput. Sci. 10 (5), 769–777. doi:10.1007/s11704-016-6112-3.
Harrell, F., 2001. Regression Modeling Strategies: With Applications to Linear Models, Logistic Regression, and Survival Analysis. Graduate Texts in Mathematics, Springer.
Hipp, J., Güntzer, U., Nakhaeizadeh, G., 2000. Algorithms for association rule mining - a general survey and comparison. SIGKDD Explor. Newsl. 2 (1), 58–64. doi:10.1145/360402.360421.
ISO/IEC/IEEE 29119-1:2013(E), 2013. ISO/IEC/ STANDARD IEEE 29119-1, Software and Systems engineering Software Testing Part 1: Concepts and Definitions. Standard. ISO/IEC/IEEE.
Itkonen, J., Mantyla, M.V., Lassenius, C., 2007. Defect detection efficiency: test case based vs. exploratory testing. In: First International Symposium on Empirical Software Engineering and Measurement, pp. 61–70. doi:10.1109/ESEM.2007.56.
Jain, A.K., Duin, R.P.W., Mao, J., 2000. Statistical pattern recognition: a review. IEEE Trans. Pattern Anal. Mach. Intell. 22 (1), 4–37. doi:10.1109/34.824819.
Kasurinen, J., Taipale, O., Smolander, K., 2010. Test case selection and prioritization: Risk-based or design-based? In: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 10:1–10:10. doi:10.1145/1852786.1852800.
Knott, A., 2012. Sensorimotor Cognition and Natural Language Syntax. MIT Press.
Li, Z., Harman, M., Hierons, R.M., 2007. Search algorithms for regression test case

prioritization. IEEE Trans. Software Eng. 33 (4), 225–237. doi:10.1109/TSE.2007.38.
Martens, H., Martens, M., 2001. Multivariate analysis of quality. an introduction. Meas. Sci. Technol. 12 (10), 1746.
Muller, K.R., Mika, S., Ratsch, G., Tsuda, K., Scholkopf, B., 2001. An introduction to kernel-based learning algorithms. IEEE Trans. Neural Netw. 12 (2), 181–201. doi:10.1109/72.914517.
Nageswaran, S., 2001. Test effort estimation using use case points. Qual. Week 1–6.
Nguyen, V., Pham, V., Lam, V., 2013. qestimation: A process for estimating size and effort of software testing. In: Proceedings of the 2013 International Conference on Software and System Process (ICSSP'13).
Refaeilzadeh, P., Tang, L., Liu, H., 2009. Cross-Validation. Springer US, Boston, MA, pp. 532–538.
Revlin, R., 2012. Cognition: Theory and Practice. Worth Publishers.
Robson, C., 2011. Real world research : a resource for users of social research methods in applied settings, third edition Chichester, West Sussex John Wiley & Sons.
Runeson, P., Höst, M., 2008. Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering 14 (2), 131. doi:10.1007/s10664-008-9102-8.
Sharma, A., Kushwaha, D.S., 2010. Complexity measure based on requirement engineering document and its validation. In: 2010 International Conference on Computer and Communication Technology (ICCCT), pp. 608–615. doi:10.1109/ICCCT.2010.5640472.
Sharma, A., Kushwaha, D.S., 2013. An empirical approach for early estimation of software testing effort using srs document. CSI Trans. ICT 1 (1), 51–66. doi:10.1007/s40012-012-0003-z.
e. Silva, D.G., de Abreu, B.T., Jino, M., 2009. A simple approach for estimation of execution effort of functional test cases. In: Second International Conference on Software Testing Verification and Validation doi:10.1109/ICST.2009.47.
Singh, S., Sahib, F., 2014. Optimized test case prioritization with multi criteria for regression testing. International Journal of Advanced Research in Computer Engineering & Technology.
Srivastava, P.R., Varshney, A., Nama, P., Yang, X.-S., 2012. Software test effort estimation: a model based on cuckoo search. Int. J. Bio-Inspired Comput. 4 (5), 278–285. doi:10.1504/IJBIC.2012.049888.
Strandberg, P.E., Afzal, W., Sundmark, D., 2018. Decision making and visualizations based on test results. 12th International Symposium on Empirical Software Engineering (ESEM).
Strandberg, P.E., Sundmark, D., Afzal, W., Ostrand, T.J., Weyuker, E.J., 2016. Experience report: Automated system level regression test prioritization using multiple factors. 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE).
Tahvili, S., 2016. A Decision Support System for Integration Test Selection.
Tahvili, S., 2018. Polynomial and spline regression analysis. https://github.com/sahar82/JSS.
Tahvili, S., Afzal, W., Saadatmand, M., Bohlin, M., Sundmark, D., Larsson, S., 2016a. Towards earlier fault detection by value-driven prioritization of test cases using fuzzy topsis. In: 13th International Conference on Information Technology : New Generations.
Tahvili, S., Saadatmand, M., Bohlin, M., 2015. Multi-criteria test case prioritization using fuzzy analytic hierarchy process. In: The Tenth International Conference on Software Engineering Advances.
Tahvili, S., Saadatmand, M., Bohlin, M., Afzal, W., Ameerjan, S.H., 2017. Towards execution time prediction for test cases from test specification. In: 43rd Euromicro Conference on Software Engineering and Advanced Applications.
Tahvili, S., Saadatmand, M., Larsson, S., Afzal, W., Bohlin, M., Sundmark, D., 2016. Dynamic integration test selection based on test case dependencies. The 11th Workshop on Testing: Academia-Industry Collaboration, Practice and Research Techniques.
Torkar, R., Awan, N., Alvi, A., Afzal, W., 2010. Predicting software test effort in iterative development using a dynamic bayesian network. In: Proceedings of the 21st International Symposium on Software Reliability Engineering - Industry Track.
Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., Stenström, P., 2008. The worst-case execution time problem - overview of methods and survey of tools. ACM Trans. Embed. Comput. Syst. 7 (3), 36:1–36:53. doi:10.1145/1347375.1347389.
Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., 2000. Experimentation in Software Engineering: An Introduction. Kluwer Academic Publishers.
Yamamoto, H., 2013. Network system, server, client terminal, timeout information providing method, timeout information display method, and programs. SA Patent 8495222.
Zhu, X., Zhou, B., Wang, F., Qu, Y., Chen, L., 2008. Estimate test execution effort at an early stage: an empirical study. In: International Conference on Cyberworlds doi:10.1109/CW.2008.34.

Sahar Tahvili is a researcher at RISE (Technical Research Institute of Sweden) and also an industrial Ph.D. student at Mälardalen University, embedded systems division. In 2014, she graduated as M.Phil. in Applied Mathematics with emphasis on optimization from Mälardalen University. Sahar's research focuses on advanced methods for testing complex software-intensive systems and designing the decision support systems (DSS). She also has a background in Aeronautical Engineering. Sahar holds a licentiate degree in software testing, "A Decision Support System for Integration Test Selection", since 2016.

Wasif Afzal is an Associate Professor at Mälardalen University in Sweden. He likes to do empirical research within software engineering in general and within software verification and validation in particular. Some keywords that describe his interests are (not exhaustive): software testing and quality assurance, prediction and estimation in software engineering, application of artificial intelligence techniques in software engineering (including search-based software engineering), decision-making based on software analytics, software metrics and evidential assessment of software engineering literature.

Mehrdad Saadatmand is a researcher in Software Engineering at the SICS Swedish ICT, Västerås. He holds a PhD degree in Software Engineering from Mälardalen University. He has worked with the SAVE project team at Mälardalen Real-Time Centre (MRTC) as part of his master thesis. He got involved in the MBAT European project which aimed to exploit the synergy between Model-Based Analysis and Testing in verification of embedded systems.

Markus Bohlin is the leader of the Industrial Efficiency (IND-E) focus area at RISE SICS. He received his PhD at Mälardalen University in 2009 and in 2013 he was appointed as Associate Professor (Docent) in Computer Science at Mälardalen University. His research interests are in the real-world application of optimization methods and operations research to industrial processes.

Sharvathul Hasan Ameerjan got his Master of Science degree in Software Engineering at Mälardalens University in 2017. He works as a test developer at Cinnober Financial Technology in Sweden.