Software Qual J (2018) 26:49–66
https://doi.org/10.1007/s11219-015-9285-3

Predicting failures in agile software development through data analytics

Feras A. Batarseh[1] · Avelino J. Gonzalez[2]

Published online: 9 August 2015
© Springer Science+Business Media New York 2015

Abstract Artificial intelligence-driven software development paradigms have been attracting much attention in academia, industry and the government. More specifically, within the last 5 years, a wave of data analytics is affecting businesses from all domains, influencing engineering management practices in many industries and making a difference in academic research. Several major software vendors have been adopting a form of "intelligent" development in one or more phases of their software development processes. Agile for example, is a well-known example of a lifecycle used to build intelligent and analytical systems. The agile process consists of multiple sprints; in each sprint a specific software feature is developed, tested, refined and documented. However, because agile development depends on the context of the project, testing is performed differently in every sprint. This paper introduces a method to predict software failures in the subsequent agile sprints. That is achieved by utilizing analytical and statistical methods (such as using Mean Time between Failures and modelling regression). The novel method is called: analytics-driven testing (ADT). ADT predicts errors and their locations (with a certain statistical confidence level). That is done by continuously measuring MTBF for software components, and using a forecasting regression model for estimating where and what types of software system failures are likely to occur. ADT is presented and evaluated.

Keywords Data Analytics · Agile · Context · Artificial intelligence

✉ Feras A. Batarseh
feras_batarseh@yahoo.com; fbatarse@gmu.edu

Avelino J. Gonzalez
avelino.gonzalez@ucf.edu

[1] College of Science, George Mason University, 4400 University Dr, Fairfax, VA 22030, USA

[2] Intelligent Systems Lab (ISL), Department of Electrical Engineering and Computer Science, University of Central Florida, 4000 Central Florida Blvd., Orlando, FL 32816, USA

1 Introduction

The agile software development lifecycle is based on the concept of incremental development, iterative deliveries, and context-driven testing (Sommerville 2007). Agile development does not promote best practices; rather, agile it is about adaptive planning and evolutionary development. After the agile manifesto[1] was introduced, multiple forms of lifecycles were influenced by it, such as extreme programming, crystal clear, feature-driven development, and scrum (Sommerville 2007) (the lifecycle of choice in this paper). The scrum process consists of the following entities: *roles, sprints, stakeholders* and a *scrum master*. Agile and scrum are discussed in more detail in the next section.

During the process of agile software development, much data are generated within specification documents, sprints definition, requirements definition, planning performed by the Scrum master, test cases suites, documentation, time-relevant metrics, number of failures, time spent in development, testing cost and other such processes and documents. All these data pieces could be used to measure, control, and improve the agile process. Most traditional lifecycles depend on process-driven decision making rather than data analysis, in this paper, ADT uses historical data analysis to help with decision making. Additionally, the availability of such data and the current advancements in software analytics enable software project management to gain insights and take decisions driven by the analytical results.

This paper is structured as follows: The next section provides definitions and a background discussion of agile and contextual software development. It also introduces the scrum development method and development lifecycles. Section 3 introduces the main novel contribution of this paper—analytics-driven testing (ADT). ADT uses statistical forecasting and failure predictions to create an estimate of where and when errors will occur in the future. Section 3 introduces these statistical methods (MTBF, Poisson, and Regression) and how they are used to create ADT. Afterward, Sect. 4 presents data used for testing ADT, provides an experimentation workflow, and introduces the experimental results for ADT. The last Sect. 5 consists of conclusions.

2 Background—agile and context

In this section, we discuss agile development (with a focus on Scrum) and its relation to context and software testing.

2.1 Agile software lifecycle

Agile software development is a recent revolutionary lifecycle development model. It is based on the *agile manifesto*. This agile manifesto was introduced in 2001 when 17 pioneers of the agile method met at the Snowbird Ski Resort in Utah in February 2001 to write it.[2] The manifesto is considered the foundation for agile principles. The principles that are relevant to this paper are: [3]

[1] The Agile Manifesto document: http://www.agilemanifesto.org/.

[2] See footnote 2.

[3] See footnote 2.

> 1. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage. 2. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale. 3. Working software is the primary measure of progress. 4. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

These principles affect all the steps of the lifecycle development; however, in this paper, the focus is on the testing phase. According to the traditional definition of software lifecycles (Sommerville 007), the main phases of software development include: (1) requirements, (2) specifications, (3) design, (4) development, (5) testing, (6) refinement, and (7) documentation. This is referred to as the waterfall model (Sommerville 2007). Other models include the spiral model,[4] reuse-oriented model (Sommerville 2007), incremental model (Sommerville 2007), and evolutionary development (Sommerville 2007). [5]In this paper, scrum was chosen for the following three reasons:

1. Scrum is agile and flexible
2. Scrum is context-driven—Could be easily modified to inject context-based testing into it.
3. Scrum is the most adopted lifecycle model these days by software corporations (Sommerville 2007; [6] (Qurashi and Qureshi 2014; Gunga et al. 2013), and it has recently been gaining much traction and success. The next Sect. (2.2) introduces scrum in detail.

2.2 Scrum development process

For many software engineers, the scrum development process is the direct opposite of the waterfall model. Scrum is much more dynamic, focuses on communication and collaboration, and emphasizes the idea of empirical processes. In scrum, projects are divided into *sprints*. These sprints are defined during the daily scrum meetings in which the software engineers, quality engineers and product owner participate and are supervised by the scrum master. Other team members include programmers, analysts, quality engineers, testers, and graphical designers.

The scrum master facilitates the relation between the team members and the product owner, drives the features and establishes plans. Sprints go on for a number of weeks (usually one to three). At the end of each Sprint, the progress and the outcomes are measured and evaluated to determine the next steps for the project. That allows the project's process and status to be dynamically modified and flexibly customized based on the current context and the completed, valid and verified work at the end of every sprint.

At each stage, the system's validity is evaluated by the scrum master, and then the development process is modified based on that evaluation. For example, if a certain module has more errors than the other, the Scrum master asks the testing team to focus more on the faultier module, and run more tests on it to improve its validity.

[4] Microsoft developer network: http://msdn.microsoft.com/en-US/enus/library/.

[5] See footnote 3.

[6] See footnote 3.

2.3 Context-driven testing

As the name applies, context-driven testing (CDT) is the process of validating and verifying software using context (Batarseh 2014). CDT is a rapidly growing testing paradigm. It is based on the idea that the value of a practice depends on its context of system validity and that projects unfold in unpredictable ways over time that need to be handled in real time. In this paper, we introduce a method that makes use of CDT. Ultimately, CDT is about not accepting a specific pre-defined set of best practices but that the best practice is what the recent context entails. The seven basic principles of CDT are (Batarseh 2014):

1. The value of any practice depends on its context.
2. There are good practices in context, but there are no best practices.
3. People, working together, are the most important part of any project's context.
4. Projects unfold over time in ways that are often not predictable.
5. The product is a solution. If the problem isn't solved, the product doesn't work.
6. Good software testing is a challenging intellectual process.
7. Only through judgment and skill, exercised cooperatively throughout the entire project, are we able to do the right things at the right times to effectively test our products.

In some cases, testing could not be done through context simply due to multiple constraints and factors such as: security limitations, privacy Laws, cultural changes at the organization, financial considerations or staff changes. Such events are not predictable and are very difficult to manage. This paper does not include such events, rather, ADT considers factors that have to do with the engineering of the software, human factors and regulations are outside the scope of this paper. Other methods have tackled context testing differently, related work in software testing is presented next.

2.4 Related work in software testing

Software testing has traditionally been broken down into validation and verification (V&V). Validation is the process of ensuring that the software matches the requirements, and verification is ensuring the system meets the functional specification. One of the most common definitions for validation is building the right system, and for verification is building the system right (Lee and O'Keefe 1994). During the last 30 years, software V&V has gone through many developments and changes. This section introduces these changes.

Computing researchers and engineers have been trying to create system development processes that are more intelligent. However, the pursuit of machine intelligence has been going on much longer than that; in the 1950s, when computing machines were introduced, these machines performed complex mathematical calculations, and it seemed that artificial intelligence (AI) of machines was just around the corner. However, after careful evaluation, it was realized that the introduced intelligence is merely that of a *one-task machine*. Massive research funding and countless research activities in AI did not produce significant improvement until what is referred to the AI winter in the 1980s. In the 1990s, AI research arose once again, and fields such as expert systems, knowledge-based systems, pattern recognition, and machine learning have blossomed.

These paradigms pushed AI research and eventually resulted in success stories in the nineties and beyond (e.g., IBM's Deep Blue and Watson machines, Google's driverless cars, face and voice recognition technologies, and many others). Some of these paradigms proved successful; however, researchers pointed that it will still take years of research

efforts before machines reach the intelligence of a rabbit, mouse or other living creatures, let alone matching human intelligence. Recent AI is transforming, it has been broken down to multiple fields, and each field has independent focus. In this paper, the focus is on adding intelligence to software testing, using one of the most recent AI paradigms, data analytics.

Testing should be performed at all stages of the system development process. There are different methods for testing that could be used for different purposes. Researchers have looked into applying conventional software validation models into AI systems, Lee et al. (Lee and O'Keefe 1994) and Mosqueira-Rey and Moret-Bonillo (2000) published two flagship papers in the field that reviewed most of existing tools and methods for validation, among other review papers [such as Vermesan et al.; Bach 2013; Rao 2014). Vermesan et al. introduced a system and a tool that evaluates testing methods and checks their applicability to AI. They introduced a scale of categories of classifications, composed of HA (highly applicable), A (applicable), LA (Low applicability), and NA (not applicable). The work of Vermesan et al. (helps to determine whether a method is applicable or not (applicability of applying conventional software validation models into AI systems) based on *mutation testing*, which is defined as the process of making minor modifications of a program's source code in a repetitive fashion. "These so-called *mutations* are based on well-defined *mutation operators* that either mimic typical user mistakes (such as using the wrong operator or variable name) or force the creation of valuable tests. The purpose is to help the tester develop effective tests or locate weaknesses in the test data used for the program or in sections of the code that are seldom or never accessed during execution (Vermesan et al.)."

Most methods are based on the conventional performance testing [such asCloudTest (Rao 2014), Cloud-TestGo (Rao 2014)]. Such testing tools focus on optimizing performance, other tools are driven by putting together multiple systems and ensuring their integration is error-less, such are called integration testing tools (such as PushToTest[7]). Recently, however, most focus has been on tools that reduce the manual aspect of testing and perform automated testing (such as Sauce Labs[8]) and also load testing (such as BlazeMeter[9]) (Rao 2014). Historically, testing was performed as field tests, simulation testing, face testing, subsystems testing, case testing, formal methods, or as sensitivity testing. However, in this paper, the interest is on a Turing test approach to testing. Turing tests were proposed by Alan Turing (Turing 1950) as a method to determine the intelligence of a computing machine when compared to a human being, where a machine and a human played a game anonymously. Turing tests apply the concepts of comparing human to machine and anonymity to the validation process for intelligent systems (Turing 1950; Onoyama et al. 2000; Richmond 2006). Another major scoop of this testing is predictive testing. Predictive testing is a type of testing that compares previous validation results with corresponding results of the system. When predictive validation is used, the results are saved, and for the next iteration, the same set of tests are performed and compared with the previous results. Predictive validation cannot be used in isolation; it is simply an approach to compare test results and evaluate the development process. More recent methods applied AI concepts to testing (Afzal et al. 2009; Afzal et al. 2010; Kalliosaari et al. 2012), for example in 2010, Afzal et al. (2009; Afzal et al. 2010) evaluated five different techniques for predicting the number of faults in testing using the following: (1). Particle swarm

[7] www.pushtotest.com/products.html.

[8] www.saucelabs.com.

[9] www.blazemeter.com.

optimization-based artificial neural networks (PSO-ANN), (2) Artificial immune recognition systems (AIRS), (3) Gene expression programming (GEP), (4) Genetic programming (GP), and (5) Multiple regressions (MR). Predictive testing using linear and exponential regression is used in this paper. It is important to note that none of the methods presented here deployed predictive data analytics in a direct manner; more specifically, no work was found in literature that uses MTBF or regression forecasting for software engineering or for testing, the details for the usage of MTBF are discussed in the following sections.

3 Software failures prediction

This section constitutes the major contribution of this paper, the ADT process. However, before that is introduced, data analytics and MTBF are presented in detail.

3.1 Brief introduction to data analytics

Data analytics is the process of collecting, cleaning, transforming, and analyzing the data using data mining and statistical modeling. The main steps in data analytics are defined as follows (Richmond 2006):

1. *Data Collection*: This is typically a time-intensive process, and the goal is to ensure that the data required for analysis are in place. That is followed by the definition of the statistical model outcome. This step aims to answer the question: what is the main goal of the analysis?
2. *Data Discovery and Transformation*: Dataset quality needs to be improved, variables that are to be used are transformed to a standard form, and unnecessary data are ruled out. Quick summary statistics are produced, such as standard deviations, means, and modes. Finally, bar charts and histograms are built to help understand the cardinality and the distribution of the data.
3. *Data Analysis*: Define the variables to be used for model development and investigate with different models. This stage also includes data visualizations that are done using heat maps, bubble charts, and other different forms of visualization. This leads to the most important stage, the actual analytic model development (Richmond 2006).
4. *Analytic Model Development*: This includes building multiple models, tuning the models, comparing model outcomes and model qualities. Main commonplace model categories used in machine learning and data mining for analytics include: classification, clustering, association, regression, and correlation.

This 4 step process is used in the experimental process described later in this paper. For software engineering, the process has evolved from the traditional waterfall form to include new philosophies that incorporate data analytics. The new development processes are agile and dynamic. Engineers become more collaborative when they use such lifecycles. Much data are generated, and many measurements are recorded during software development. Then data analytics are used on the generated data to help in decision making. In this paper, a forecasting regression model is used in combination with the analysis of MTBF. These two are described next.

3.2 Measuring elapsed time between software failures (MTBF)

Measuring mean time between failures (MTBF) is a common practice in industry. In many certification processes, MTBF is defined as the elapsed time between machine failure and the time that the machine started working again. That measurement is done by calculating: *run time* ÷ *number of failures*. However, this formula measures the value of system failures, but to have faith in this method and the numbers produced, *statistical confidence* needs to be measured as well. The confidence is very important, especially in cases of machines that fail in real time; faulty confidence could cause serious issues such as machines failures, financial losses, and in some cases, people's lives, among many other issues related to the overall health of the system.

In statistical forecasting, a model is built to forecast values in the future based on historical data. A regression model represents the line that best fits the existing data points, and then it extends into the future (using the regression formula). This line has a certain distance from the actual data points, the further that distance is, the worst the model is, and the closer the points are to the line, the better the model is. This model quality is what is referred to as *model goodness*. The better the model is, the more accurate and dependable the forecast is. To measure confidence, multiple statistical methods could be used, such as Chi-squared distribution, Weibull (1951), or Poisson distribution. In this paper, we use the Poisson distribution (Weibull 1951), as it is the most widely used distribution specifically for MTBFs (Speaks 2000). Weibull distributions are used in some cases for MTBF; however, Poisson/MTBF is the most commonplace combination for reliability engineering statisticians (Speaks 2000). *P* values and R-squared calculations, on the other hand, could be used at many stages to evaluate the outcomes of the model (more details to follow in this regard). Quality calculations are only valid when the *number of failures* is not equal to zero. The standard or initial MTBF confidence that many systems aim to accomplish is

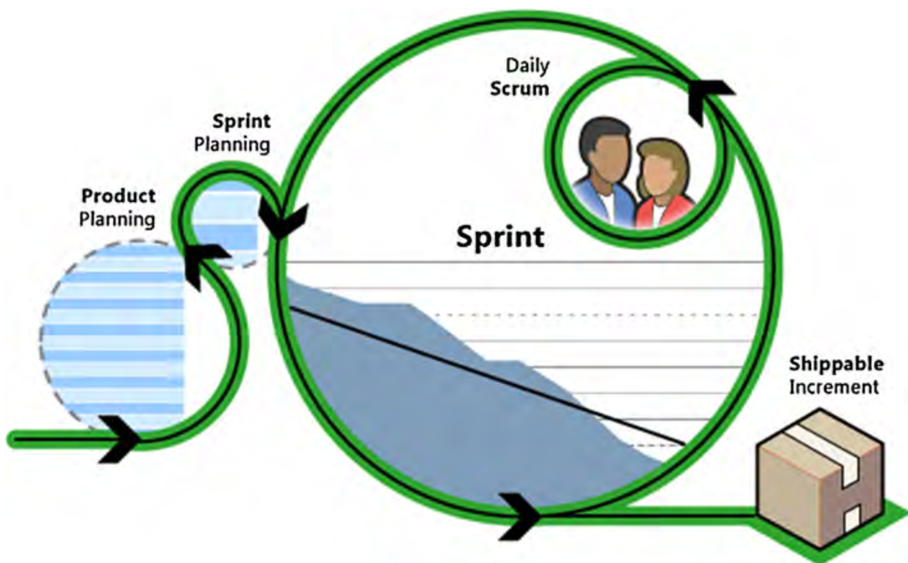

Fig. 1 The sprint-based process at Microsoft (Microsoft developer network: http://msdn.microsoft.com/en-US/enus/library/)

60 %. Depending on the system underhand, confidence could be raised up to 90 or 95 %. The higher the confidence is, the lower the MTBF that is introduced within that confidence. The reasoning is that the longer the time between failures is, the easier it is to guarantee that time before a failure, and therefore, a lower confidence in MTBF (Fig. 1).

In this paper, the MTBF analytic is applied to the agile development process of a software system. Confidence is a factor of desired MTBF, run time, and number of failures. Figure 2 below illustrates the relationship between run time, MTBF, data collection, system failures and confidence (Richmond 2006; Kalliosaari et al. 2012; Weibull 1951; Speaks 2000; Reed 2010). More importantly, the Figure shows where ADT falls within the bathtub distribution flow, more details in that regard to follow in next sections.

If one looks at the bigger picture of the life time of any machine, equipment, or any software system, it is evident to see that it could be presented with what is referred to the bathtub distribution (refer to Fig. 2). That distribution represents the idea that during the initial/final steps of the lifecycle of a system, the system is prone to failures. That is referred to infant mortality stage at the beginning and wear-out stage at the end (the bathtub distribution). MTBF (Speaks 2000) is only measured during the useful life period, and not during either initial or final stages. This paper applies these concepts to agile software testing.

Based on a recent study by the National Institute of Standards and Technology (NIST), locating these errors is the most resource/time-consuming activity (Tassey 2002). The premise of this paper is that predicting the locations with high confidence shall help the engineering team in minimizing the negative impacts of software errors on the development process.

The next section introduces the ADT process; it aims to predict software failures and their locations in a contextual agile development lifecycle through data analytics. Failures in the context of ADT consist of the errors in syntax, runtime, compiling, launching, or the user interface.

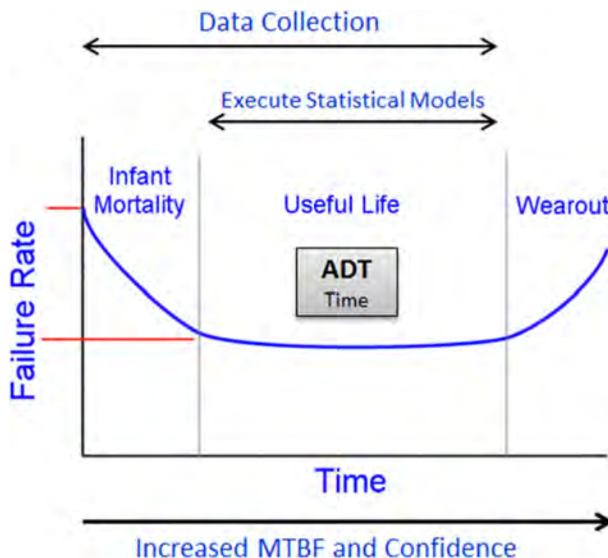

Fig. 2 Bathtub distribution applied for software and ADT (Reed 2010)

The main problem statement that this paper aims to address is as follows:

3.2.1 Problem statement

Error allocation and prediction are considered to be the *most time-consuming* activities of software testing (based on a comprehensive NIST report (Tassey 2002)). As established in previous sections of this paper, literature has a void in *effective* error allocation methods using *AI-driven* principles. ADT is introduced to help analysts and engineers predict errors and their locations (with high confidence) through a formal and AI-driven statistical method, and influence the overall health of the system positively.

To address the aforementioned problem statement, ADT, the novel method introduced in this paper is presented in detail in the next section.

3.3 Analytics-driven testing (ADT)

The testing procedure presented in this paper is based on data mining and analytics; therefore, the name ADT. The main pillars of ADT are: (1) Scrum, (2) CDT, (3) Regression Forecasting, and 4. MTBF Analytics. Each of these four pillars includes three major drivers of ADT. That is presented in Fig. 3 below, which is key to understanding the different parts of ADT. All the items presented in the graph below are introduced in previous sections.

Based on these four pillars, and in a nutshell, ADT is described as follows: *MTBF and regression forecasting are used to predict failures in a scrum development lifecycle*. To expand on this definition, the process of ADT has the following major 6 phases:

1. *Infant Mortality*: The software project starts. The scrum master defines the goals of the first group of sprints. The master also ensures the readiness to collect data
2. *Data Collection*: For each sprint, data are collected using the ADT tool (refer to Fig. 4). All software failures are recorded, and the time between these failures (MTBF) is measured. The data collected includes: software module, sprint number, run time, and number of failures. Poisson is used as basis for MTBF.
3. *Useful Life Phase*: For each sprint after the *infant mortality* and data collection phases (refer to Fig. 2), the model is retrained and software failures are measured. More importantly, in this phase, the regression model is deployed\prepared for the upcoming step 4.

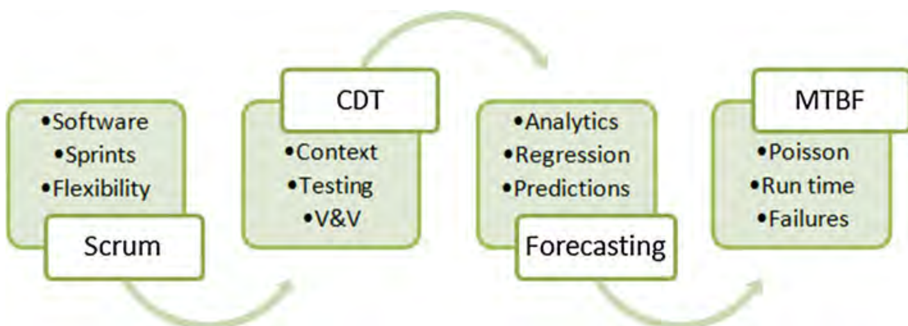

Fig. 3 The 4 pillars of *analytics-driven testing* (ADT)

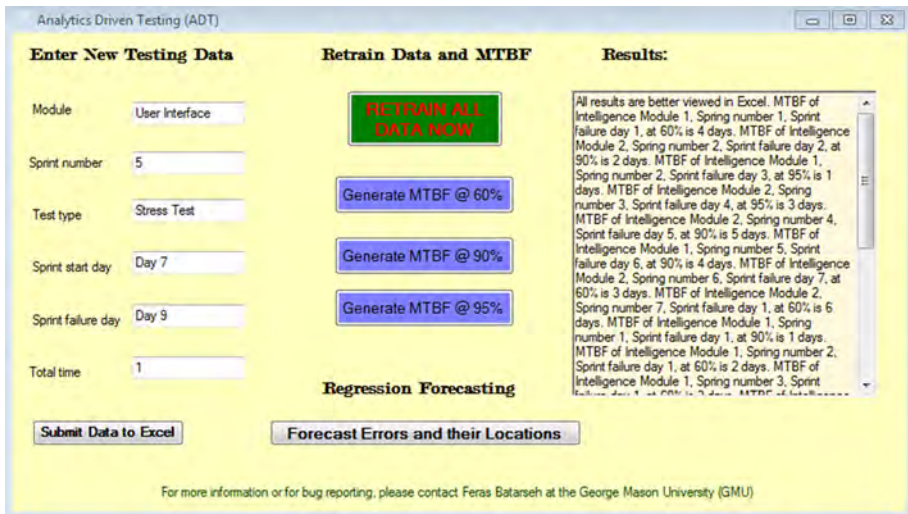

Fig. 4 The ADT tool

4. *Regression Forecast*: MTBF for the next sprint is predicted for each module. That is done using the linear regression formula, (embedded in the R script and the Java tool), and the data collected (from steps 1-3).
5. *CDT Tuning*: The context of testing changes based on the results, testers focus on modules with lower MTBFs and execute test cases that are relevant to errors that ADT forecasted. The testers are performing proactive testing using the results from steps 3 and 4. The goal of this phase is to increase MTBF in the next round and eventually reduce all system failures
6. *Validity Measure*: If the sprint is declared as the last development sprint, then testing is over and system validity is measured by the overall system MTBF. If management does not deem the MTBF values as acceptable, engineers need to loop back to step 3 and retrain the regression model with the new data (the ADT tool facilitates that process) and so on. If MTBF numbers are acceptable, testing is over

Refer to the experiment in the next section for a detailed example of all the ADT steps. The ADT steps presented in this process are used in the experimental evaluation. For the experiment, an AI system is developed, sprints are defined, data are collected, and the MTBF and regression models are deployed.

4 Experimental evaluation

The development data were recorded and entered into an excel sheet (using the tool shown in Fig. 4). R scripts were developed and executed within a Java tool to measure MTBF, and the goodness of the model. This section introduces the experimental process and the results.

5 Experimental background

There is an abundance of data analytical tools available in both commercial and open source venues. Some examples include: SAS, SPSS, Tableau, MicroStrategy, Pentaho, Knime, Stata, and R. Such tools provide a data-analytics-driven workflow and makes it easier for analysts to pivot, analyze, structure, and perform mining on their data. These tools aid in the overall decision-making process of many industries such as health institutions, financial banks, government, and many others. However, these tools have rarely been used for software development decision making. Data analytical tools use statistical models for decision making. Instead of using one of the off-the-shelf tools (due to their expensive prices and their enterprise-deployment nature), we developed our own prediction and MTBF models and applied them on data in Excel and the statistical calculations are done in R within a Java Tool. R is a statistical language that consists of thousands of statistical, analytical and machine learning packages. The language is widely used among statisticians, analysts and data miners. R is an open source language developed at the University of Auckland, New Zealand. For more information on R, go to: www.r-project.org.

In our code, we used the *lm* function for linear regression and used the Poisson distribution function *ppois* for MTBF. The scripts are available from the authors upon request. Furthermore, a Java interface was developed to query the data in Excel and execute the R scripts then return results. A screenshot of the tool is shown below in Fig. 4.

The developed tool provides an interface for the engineer/tester to enter testing data at every sprint, retrain the data whenever they add more rows and then generate new MTBF scores and new regression forecasts based on historical data. The results are then shown in the text area; however, the excel sheet is also updated, and it could be used for further visualization of results. See screenshots of Excel data below (Figs. 6 and 7).

The data and the Excel sheets used for this experiment have been uploaded to a public web server and are available for all non-commercial research purposes, it can be found under the following paths: https://www.dropbox.com/s/xxkk4a7q6bgwapw/Regression.xlsx?dl=0 https://www.dropbox.com/s/0j5olwhwnl9ybcr/Experimental%20Data.xlsx?dl=0.

For more details on the tool, the code, and the scripts, please contact the authors. The next section introduces the process followed to perform the experiment for ADT.

5.1 Experimental process

Through the process described earlier, an experiment was conducted on an example AI system; the system has six main modules:

1. Human–Computer Interaction (HCI) Module: This module has the intelligence and the logic that interacts with users actions (mouse clicks in this case). The module monitors the human involvement with the system and uses built in logic to reply to that.
2. Intelligent Module 1: This is the first of three intelligent modules, and it contains logic about deduction, reasoning, and problem solving. It prepares data for further processing that occurs in the second and third intelligent modules.
3. Intelligent Module 2: This is the second module (out of three) that has the algorithms responsible for machine learning. This module has built-in algorithms such as clustering, inductive logic, reinforcement learning, and classification.

4. Intelligent Module 3: This module has code that represents advanced AI methods. Algorithms in this module include neural networks, natural language processing (NLP), and genetic programming.
5. Knowledge base: The knowledge base has a set of rules that represent the knowledge that the system contains. This module is very important as it represents the core logic, knowledge, and intelligence of the system.
6. User Interface: This module has all the graphical user interface components, such as designs, images, and objects such as button, dropdown lists…etc. User interfaces require vital validation efforts as well.

This six-module system is considered to be a mid-size system; it can consume any type of data and perform AI analysis on the data through the models in the Intelligent Modules. For example, the user inputs data through the User Interface, and then the rules in the Knowledge base are used in conjunction with the Intelligent Modules to train the data, and provide results that the user can view, evaluate and modify through the HCI module. Development required 30 sprints, throughout 98 business days (4–5 months of development). During development, the sprints start and end days are recorded, as well as failures, Sprint number, and run time in hours. Refer to Fig. 6 for a sample from the data set. Most importantly, MTBF was measured and used for future data points' prediction. These data are used for failure prediction (of future systems development). To evaluate such predictions, two regression models are built, one linear and other exponential regression models. The quality of these models was measured and proved to have high quality. Model "goodness" was measured using a developed a Java tool, the R-squared formula (the official statistical name is: coefficient of determination; Sykes) was coded in R, and it measure the goodness of a model per row as it is evident in the experimental results. R-squared is one way of measuring the quality of a model; it divides the sum of squares of all the data points of the regression model, over the total sum of squares. The R-squared value is measured using the distance of the forecasted prediction line and the actual data

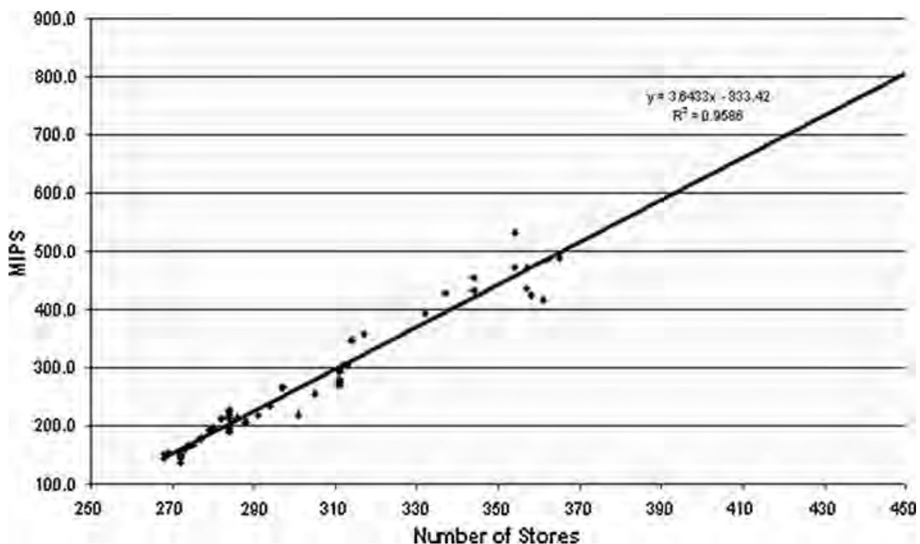

Fig. 5 Finding the best fit line, forecasting and measuring R^2 (Sykes)

line. *The less that distance is, the better the model is*. The regression model finds the line that best fits the data points, for example, see Fig. 5 below, it shows the formula of the line that explains the best fit of the data points (y), the line is then extended into the next (future) data points to create the forecast. R-squared is measured as well to evaluate the quality.

Sykes describes regression (linear or exponential) analysis as follows: "Regression analysis is a statistical tool for the investigation of relationships between variables. Usually, the investigator seeks to ascertain the causal effect of one variable upon another—the effect of a price increase upon demand." A regression analysis example measures the effect of changes in the cash supply upon the inflation rates and so on.

For regression, the data miner assembles data of the variables of interest and deploys regression to estimate the quantitative effect of all the variables on the variable that they influence. In this paper, the influenced variable is MTBF. The data shown in Fig. 6 below were collected during the development process. For the six modules presented previously, the data collected during the development process of each module consisted of: Sprint number, Test type, Sprint length (sprint start day, and sprint failure day), Total time for the sprint in days, and then the run time for the sprint before failing in hours. Based on these variables, failure forecasts are calculated using MTBF (*run time* ÷ *number of failures*), and confidence is measured, as discussed earlier in this paper.

5.2 Experimental results

Using the data collected as described in the previous section, the experiment was conducted and the results were recorded. With this evaluation process, ADT can predict when and where will the next error happen, and accordingly the scrum master can take preventive measure to avoid future errors. This method provides a preventive measure during development and guides the team to expect an error before it happens, and even make early changes to avoid it totally.

When historical failure data is collected, it has the following:

1. ID for each row, a common database practice (assign one row record an ID)
2. Software module (all modules describe in Sect. 4.1)

ID	Software Module	Sprint Number	TestType	Sprint Start Day	Sprint Failure Day	Total Time	Run Time in Hours	MTBF@ 60% Confidence	Failure Forecast
1	Intelligence Module 1	1	FunctionalityTest	Day 1	Day2	1	8	4	?
2	Intelligence Module 1	1	PerformanceTest	Day 1	Day2	1	8	4	?
3	Intelligence Module 1	1	StressTest	Day 1	Day2	1	8	4	?
4	Intelligence Module 1	1	GuiTest	Day 1	Day2	1	8	4	?
5	Knowledge Base	1	FunctionalityTest	Day 1	Day2	1	8	4	?
6	Knowledge Base	1	PerformanceTest	Day 1	Day2	1	8	4	?
7	Knowledge Base	2	StressTest	Day 3	Day 4	1	8	4	?
8	Knowledge Base	2	GuiTest	Day 3	Day 4	1	8	4	?
9	User Interface	2	FunctionalityTest	Day 3	Day 4	1	8	4	?
10	User Interface	2	PerformanceTest	Day 3	Day 4	1	8	4	?
11	User Interface	2	StressTest	Day 3	Day 4	1	8	4	?
12	User Interface	2	GuiTest	Day 3	Day 4	1	8	4	?
13	Human-Computer Inte	2	FunctionalityTest	Day 3	Day 4	1	8	4	?
14	Human-Computer Inte	2	PerformanceTest	Day 3	Day 4	1	8	4	?
15	Human-Computer Inte	2	StressTest	Day 3	Day 4	1	8	4	?

Fig. 6 Initial dataset (data collected from the scrum development lifecycle)

3. Sprint number: as mentioned previously scrum testing is based on sprints, recording the sprint number is essential. It is used as an input to the forecasting model.
4. Time dimension, the sprint start and failure day. These 2 values are used to measure time for the sprint that experienced a failure.
5. Total time in days, hours, and MTBF (inputs to the regression model as well).

The regression model built for ADT, MTBF is what is referred to in statistics: the *dependent* variable (because that is the value that we want to forecast). All other variables (except ID) are used as *independent* variables (inputs).

By looking at the results from the experiment, the new data columns have the following (see Fig. 7):

1. Historical MTBF values (recorded values from previous software engineering efforts) and measured to 60 % using the method presented in Sect. 3.2.
2. MTBF prediction values: using the regression model, new values are generated and are used for the predictions. This is calculated using the math presented in Sect. 4.1. To deploy this model, the user enters a new "row" of data (representing a new data point) into this regression model (represented in the excel sheet), and the model will provide a result (in the MTBF Prediction data column) based on the formula of the regression line (presented in the sections before), and therefore, that number is the *prediction* for the next agile sprint.
3. The difference between actual and predicted values; equal to: Historical MTBF values—MTBF prediction values.

The data consist of 181 rows, the total of difference for all rows is 360, with an average of MTBF = 114 min. Average testing runtime in hours for the system that was used for experimentation is 1770 min (29.5 h).

Software module	Sprint failure day	Sprint number	Test type	Sprint start day	Run Time In Hours	Total time	Mtbf @60% Confidence	MTBF Prediction	Difference	Mtbf @60% Confidence Predictor
Human-Computer Interaction Module	Day 19	5	FunctionalityTest	Day 14	40	5	20	20	0	99.942%
			GuiTest	Day 14	40	5	20	20	0	99.942%
			PerformanceTest	Day 14	40	5	20	20	0	99.942%
			StressTest	Day 14	40	5	20	20	0	99.942%
	Day 4	2	FunctionalityTest	Day 3	8	1	4	4	0	99.942%
			PerformanceTest	Day 3	8	1	4	4	0	99.942%
			StressTest	Day 3	8	1	4	4	0	99.942%
	Day 49	13	FunctionalityTest	Day 45	32	4	16	15	(1)	99.942%
	Day 50	14	GuiTest	Day 50	0	0	0	0	0	99.942%
			PerformanceTest	Day 50	0	0	0	0	0	99.942%
			StressTest	Day 50	0	0	0	0	0	99.942%
	Day 60	19	FunctionalityTest	Day 60	16	2	8	8	0	99.942%
	Day 67	20	FunctionalityTest	Day 61	48	6	24	24	0	99.942%
			GuiTest	Day 61	48	6	24	48	24	99.942%
				Day 61	48	6	24	48	24	99.942%
			PerformanceTest	Day 61	48	6	24	48	24	99.942%
				Day 61	48	6	24	48	24	99.942%
			StressTest	Day 61	48	6	24	48	24	99.942%
				Day 61	48	6	24	48	24	99.942%
	Day 7	3	GuiTest	Day 5	16	2	8	8	0	99.942%
	Day 77	25	FunctionalityTest	Day 70	56	7	28	28	0	99.942%

Fig. 7 MTBF forecasting results

Percentage of failures of prediction for all time used or experimentation in testing is 6.5 %. For all the 181 rows, for all the testing times that is recorded the regression model for MTBF, using ADT had an overall statistical success rate of: 93.5 %.

6 Conclusions

ADT is based on four pillars that aim to redesign the traditional testing process: context of the testing process, forecasting software failures, mean time between failures and regression analysis. This section introduces summaries, future work, and our main contributions.

6.1 Summary

Software engineers and testers have always sought to accomplish *complete* software quality. In software testing, validation and verification are the main two pillars of software quality and evaluation. Being able to find errors and failures that already exist in the software is valuable, but what is much more helpful is the ability to predict software failures (and their locations) before they occur, especially if that is done with high statistical confidence. Predictive analytics are used in ADT to accomplish future failures/errors reduction; many other methods in the literature are introduced to reduce errors after they are actually detected (Meziane and Vadera; Batarseh and Gonzalez 2015; Batarseh et al. 2013). However, in this paper, a proactive software failure prediction method is introduced and evaluated. ADT uses MTBF and regression analysis to predict future errors and their locations in a sprints-based lifecycle (scrum).

6.2 Future work and contributions

ADT users are required to pay close attention to ongoing behavioral changes. As mentioned before, privacy, security, and cultural changes can have a direct effect on ADT. Nonetheless, behavioral changes occur in patterns. Research has been done specifically for scrum patterns extractions (Kennett 2013). Such patterns are developed with instructions

Fig. 8 A pattern's template (Kennett)

Name:	it should be short and as descriptive as possible
Examples:	one or several diagrams/drawings illustrating the use of the pattern,
Context:	it focuses on the situation where the pattern is applicable,
Problem:	it is a description of the major forces/benefits of the pattern as well as its applicability constraints
Solution:	it details the way to solve the problem, it is composed of static relationships as well as dynamic ones describing how to construct an artefact according to the pattern. Variants are often proposed along with specific guidelines for adjusting the solution with regards to special circumstances. Sometimes, the solution requires the use of other patterns.

presented in (Kennett 2013), however, patterns have generic templates; these are shown in Fig. 8. The purpose of creating a pattern is to define a reusable scrum component. Building scrum patterns with ADT injected in them is one of the future steps of this research.

ADT does not consider the design phase of test cases; therefore, as part of future plans for ADT, it is important to extend ADT into the design phase. Testing design has multiple statistical aspects, some of which are discussed in (Kenett 2010). Testing design improves the overall process of testing, and also helps the organization go up the CMMI pyramid. One of the main challenges of ADT, however, is that it is highly dependent on previous errors occurring. ADT requires data to be successfully deployed, if no data are recorded, MTBF or regression models cannot be built. However, this is part of an overall challenge of statistics and data analytics deployments. Although this model improves as more data are available (dependent on the abundance of data), the predictions will have less confidence with less data, but the quality (R-squared) of the statistical model will not be affected.

MTBF is often used as part of industrial reliability engineering practices; it has never been used for software testing as shown here. There has been minimal effort in building test execution decision-making processes during development (Herzig 2014) (besides the traditional software reliability work (Musa et al. 1988)). It is important to create predictions during the development process (pre-release) for post-release errors (Herzig 2014). The paradigm in (Herzig 2014) is consistent with the incremental advantages of MTBF usage within ADT. Other research has used error correction after prediction (Rafi 2012), although it is a very important part of the process, error correction is outside the scope of this paper. However, as part of future steps, ADT will be expanded in that direction. It is also essential to measure the effort associated with predicting errors (Mende 2010), although ADT's process is mostly automated (using R and the presented tool), another important part of future work is measuring time and effort associated with ADT. Furthermore, as it is the nature of statistical modeling, most models require extensive and continuous maintenance; otherwise, models will be victims to multiple negative usability and long-term effects. As part of the future experimental work, our model will go through a set of tests to gauge such effects and provide specific suggestions on how to eliminate them. Data need to be continuously updated and validated to ensure that the models stay relevant (and usable). Some suggested model and *data validation* methods that would possibly eliminate a good percentage of negative long-term effects include: (1) Data values' summary statistics for each round of training or scoring of the model, (2) check for outliers such as negative numbers, decimal points, and other similar mathematical outcomes and (3) monthly model consistency and data redundancy checks. More experimentation with long-term effects is part of future work.

Data analytics (used in ADT) could be either exploratory (such as clustering or any type of descriptive analytics) or confirmatory (such as regression, forecasting, and most of predictive analytics). In this paper, confirmatory data analytics are used (regression forecasting specifically). The ADT failure prediction is based on evaluating historical data, looking for patterns, and forecasting the future iterations failure times (MTBF) using linear regression within an agile lifecycle. ADT was experimented with an AI system; the results of the experiment are recorded and deemed successful, given that the regression model used for forecasting errors had a success rate (quality) of 93.5 %.

References

Afzal, W., Torkar, R., & Feldt, R. (2009). A systematic review of search based testing for non-functional system properties, *Journal of Information and Software Technology*, vol. 51.

Afzal, W., Torkar, R., & Feldt, R. (2010). Search—based prediction of fault-slip-through in large software projects, In *2nd IEEE International Symposium on Search Based Software Engineering*, pp 79–88.

Bach, J. (2013). *Heuristic test planning: Context model*, by Satisfice, Inc.

Batarseh, F. A. (2014). Context-driven testing on the cloud. In P. Brézillon & A. J. Gonzalez (Eds.), *Context in computing: A cross disciplinary approach for modeling the real world* (pp. 25–44). New York: Springer.

Batarseh, F. A., & Gonzalez, A. J. (2012). *Incremental lifecycle validation of knowledge-based systems through commonKADS*, Published at the IEEE Transactions on Systems, Man and Cybernetics (TSMC-A).

Batarseh, F. A., & Gonzalez, A. J. (2015). Validation of knowledge-based systems: a reassessment of the field. *Artificial Intelligence Review, 43*, 485–500. doi:10.1007/s10462-013-9396-9.

Batarseh, F. A., Gonzalez, A. J., & Knauf, R. (2013). Context-assisted test cases reduction for cloud validation, In *Proceedings of the Eighth International and Interdisciplinary Conference on Modeling and Context 2013*, Annecy, France.

Gunga., V, Kishnah., S, & Pudaruth., S. (2013). Design of a multi-agent system architecture for the scrum methodology, In *Proceedings of the international journal of software engineering and applications (IJSEA)*, Vol. 4, No. 4.

Herzig, K. (2014). Using pre-release test failures to build early post release defect prediction models, *The 25th IEEE international symposium on software reliability engineering*.

Kalliosaari, L., Taipale O., & Smolander, K. (2012). Testing in the cloud: Exploring the practice, a paper published at the IEEE software magazine.

Kenett, R., & Baker, E. (2010). *Process improvement and CMMI for systems and software*, a book published by CRC Press, ISBN 9781420060508.

Kennett, R. (2013). *Implementing scrum using business process management and patterns analysis methodologie*s, Published at the Dynamic Relationships Management Journal.

Lee, S., & O'Keefe, R. (1994). Developing a strategy for expert system validation and verification. *IEEE Proceedings of the IEEE Transaction on systems Man and Cybernetics, 24*, 643–655.

Mende, T. (2010). *Effort-aware defect prediction models*, Published at the 14th European Conference on Software Maintenance and Reengineering (CSMR).

Meziane, F., & Vadera, S. (2010). Artificial intelligence in software engineering, Chapter 14, information science reference. Computer Software Development. ISBN 978-1-60566-758-4.

Mosqueira-Rey, E. & Moret-Bonillo, V. (2000). Validation of intelligent systems: A critical study and a tool. In *Proceedings of expert systems with applications*, pp. 1–16.

Musa, J. D., Laninno, A., & Okumoto K. (1988). *Software reliability: Measurement, prediction, and application*, Published at the International Quality and Reliability Engineering Journal by Wiley. Vol 4.

Onoyama, T., Oyanagi, K., Kubota, S., & Tsuruta, S. (2000). Concept of validation and its tools for intelligent systems, In *IEEE 2000, the proceedings of the digital object identifier, TENCON*, pp. 394–399.

Qurashi, S., & Qureshi, M. (2014). Scrum of scrums solution for large size teams using scrum methodology, *Proceedings of the Life Science Journal*, *11*(8), 443–449.

Rafi, S.M. (2012). Incorporating *fault dependent correction delay in SRGM with testing effort and release policy analysis*, Published at the CSI Sixth International Conference on Software Engineering (CONSEG).

Rao, R. (2014). Ten cloud based testing tools, a report published under: Tools Journal, cloud based testing tools.

Reed, W. (2010). A flexible parametric survival model which allows a bathtub shaped hazard rate function. *Journal of Applied Statistics, 8*, 1–17.

Richmond, B. (2006). *Introduction to data analytics handbook*. Migrant and Seasonal Head Start Technical Assistance Center, Academy for Educational Development. http://files.eric.ed.gov/fulltext/ED536788.pdf.

Sommerville, I. (2007). *Software Engineering*. 8th edition, 2007, Chapter 4, published by Addison Wesley.

Speaks, S. (2000). Reliability and MTBF overview, White Paper by Vicor Reliability Engineering.

Sykes A. (2014). *Introduction to Regression Analysis*. A paper published at the University of Chicago College of Law. http://www.law.uchicago.edu/files/files/20.Sykes_.Regression.pdf.

Tassey, G. (2002). The economic impacts of inadequate infrastructure for software testing, A report prepared by the National Institute of Standards and Technology, US Department of Commerce.

Turing, A. (1950). Computing machinery and intelligence, In *Proceedings of Mind 1950*, LIX, Number 236, pp. 433–460.

Vermesan, A., & Hogberg, F. (1999). Applicability of conventional software verification and validation to knowledge base components: A qualitative assessment, In *Proceedings of the 5th European symposium on validation and verification of knowledge based systems—theory, tools and practice, (EUROVAV '99)*, pp. 343–364.

Weibull, W. (1951). A statistical distribution function of wide applicability. *Journal of Applied Mechanics*, 293–300.

Feras A. Batarseh is a Research Assistant Professor with the College of Science at George Mason University (Fairfax, VA). He received his B.Sc. in Computer Science in 2006 from Princess Sumaya University for Technology (Amman, Jordan). He received his M.Sc. and Ph.D. degrees from the University of Central Florida (Orlando, FL) in Computer Engineering in 2007 and 2011 respectively. His research interests span the areas of Artificial Intelligence, Data Analytics and Software Testing.

Avelino J. Gonzalez is a Professor in the Department of Electrical Engineering and Computer Science at the University of Central Florida (Orlando, FL), and a director of the Intelligent Systems Lab (ISL). He received his Ph.D. in Electrical Engineering in 1979 from the University of Pittsburgh.