Proceedings of the 5th WSEAS Int. Conf. on SIMULATION, MODELING AND OPTIMIZATION, Corfu, Greece, August 17-19, 2005 (pp650-665)

The use of Modeling & Simulation-based Analysis & Optimization of Software Testing

Ljubomir Lazić, SIEMENS d.o.o, Radoja Dakića 7, 11070 Beograd, Serbia&Montenegro,
http://www.siemens.co.yu
Nikos Mastorakis, Military Institutions of University Education, Hellenic Naval Academy, Terma Hatzikyriakou, 18539, Piraeus, Greece

Abstract:- The Software Testing Process (STP) raised many challenging issues in past decades of software development practice, several of which remain open. The System/Software under test (SUT) continually increases complexity of applied technology, software application domain model and corresponding process knowledge and experience. Today's SUT have billions of possible inputs and outputs. How does one obtain adequate test coverage with reasonable or even optimal number of test events i.e. test cases? How does one measure test effectiveness, efficacy, benefits, risks (confidence) of project success, availability of resources, budget, time allocated to STP? How does one plan, estimate, predict, control, evaluate and choose "the best" test scenario among hundreds of possible (considered, available, feasible) number of test events (test cases)? How does one judge, decide if satisfied/not satisfied program behavior, Pass/Fail result , Go/Ngo decision after test run i.e. does have Test Oracle? This paper describes the major issues that are encountered while developing framework of **I**ntegrated and **O**ptimized **S**oftware **T**esting **P**rocess (**IOSTP**). IOSTP framework combines few engineering and scientific areas such as: Design of Experiments, Modeling & Simulation, integrated practical software measurement, Six Sigma strategy, Earned (Economic) Value Management (EVM) and Risk Management (RM) methodology through simulation-based software testing scenarios at various abstraction levels of the SUT to manage stable (predictable and controllable) software testing process at lowest risk, at an affordable price and time. In order to significantly improve software testing efficiency and effectiveness for the detection and removal of requirements and design defects in our framework of IOSTP, during 3 years of our IOSTP framework deployment to STP we calculated overall value returned on each dollar invested i.e. ROI of 100:1.

Key-Words:- software testing, optimization, simulation, continuous risk management, earned value management, test evaluation, measurement.

1 Introduction

The increasing cost and complexity of software development is leading software organizations in the industry to search for new ways through process methodology and tools for improving the quality of the software they develop and deliver. However, the overall process is only as strong as its weakest link. This critical link is software quality engineering as an activity and as a process. Testing is the key instrument for making this process happen.

Software testing has traditionally been viewed by many as a necessary evil, dreaded by both software developers and management alike, and not as an integrated and parallel activity staged across the entire software development life cycle. One thing is clear - by definition, testing is still considered by many as only a negative step usually occurring at the end of the software development process while others now view testing as a "competitive edge" practice and strategy.

Solutions in software engineering are more complex-interconnect in more and more intricate technologies across multiple operation environments. With the increasing business demand for more software coupled with the advent of newer, more productive languages and tools, more code is being produced in very short periods of time.

In software development organizations, increased complexity of product, shortened development cycles, and higher customer expectations of quality proves that software testing has become extremely important software engineering activity. Software development activities, in every phase, are error prone so defects play a crucial role in software development. We usually think of testing in software development as something we do when we run out of time or after we have developed code. However, the fundamental approach as presented here focuses on testing as a fully integrated but independent activity with development that has a lifecycle all its own, and that the people, the process and the appropriate automated technology are crucial for the successful delivery of the software based system. Planning, managing, executing, and documenting testing as a key process activity during all stages of development is an incredibly difficult process.

Software vendors typically spend 30 to 70 percent of their total development budget i.e. of an organization's software development resources on testing. Software engineers generally agree that the cost to correct a defect

increase, as the time elapsed between error injection and detection increases several times depending on defect severity and software testing process maturity level [1,2].

Until coding phase of software development, testing activities are mainly test planning and test case design. Computer based Modeling and Simulation (M&S) is valuable technique in Test Process planning in testing complex Software/System under test (SUT) to evaluate the interactions of large, complex systems with many hardware, user, and other interfacing software components such are Spacecraft Software, Air Traffic Control Systems, in DoD Test and Evaluation (T&E) activities [4-6].

There is strong demand for software testing effectiveness and efficiency increases. Software/System testing effectiveness is mainly measured by percentage of defect detection and defect leakage (containment), i.e. late defect discovery. Software testing efficiency is mainly measured by dollars spent per defect found and hours spent per defect found. To reach ever more demanding goals for effectiveness and efficiency, software developers and testers should apply new techniques such as computer-based modeling and simulation - M&S [6-9].

The results of computer-based simulation experiments with a particular embedded software system, an automated target tracking radar system (ATTRS), are presented in our paper [6]. The aim is to raise awareness about the usefulness and importance of computer-based simulation in support of software testing.

At the beginning of the software testing task the following question arises: how should the results of test execution be inspected in order to reveal failures? Testing by nature is measurement, i.e. test results must be analyzed and compared with desired behavior.

This paper is contribution to software testing engineering by presenting challenges and corresponding methods implemented in **I**ntegrated and **O**ptimized **S**oftware **T**esting **P**rocess framework (**IOSTP**). IOSTP framework combines few engineering and scientific areas such as: Design of Experiments, Modeling & Simulation, integrated practical software measurement, Six Sigma strategy, Earned (Economic) Value Management (EVM) and Risk Management (RM) methodology through simulation-based software testing scenarios at various abstraction levels of the SUT to manage stable (predictable and controllable) software testing process at lowest risk, at an affordable price and time [6-22]. In order to significantly improve software testing efficiency and effectiveness for the detection and removal of requirements and design defects in our framework of IOSTP, during 3 years of our IOSTP framework deployment to STP of embedded-software critical system such as Automated Target Tracking Radar System [6,16,19], we calculated overall value returned on each dollar invested i.e. ROI of 100:1 .

The paper begins with an outline of fundamental challenges in software testing in section 2, then the problems with software testing and Integrated and Optimized Software Testing Process framework state-of-the-art methods implementation is described in section 3. The main contribution of M&S with illustrative details and experience of methods implemented in IOSTP are presented in section 4. In section 5, M&S as one effective and efficient test oracle solution is presented. Section 6 illustrate how M&S can be successfully exploited in Trade-Off Studies and Prioritization, Cost-Performance trade-offs i.e. as the IOSTP optimization model. Finally in section 7, some concluding remarks are given.

2 Software testing, as a part of software development process, is human intensive work with high uncertainty i.e. risks

There are at list four domains of software engineering where uncertainty is evident: uncertainty in requirements analysis, uncertainty in the transition from system requirements to design and code, uncertainty in software re-engineering and uncertainty in software reuse [24]. Software testing, like other development activities, is human intensive and thus introduces uncertainties and obeys Maxim of Uncertainty in Software Engineering-MUSE [24]. Afore mentioned uncertainties may affect the development effort and should therefore be accounted for in the test plan. We identify three aspects of test planning where uncertainty is present: the artifacts under test, the test activities planned, the plans and their fulfillments themselves. According to MUSE, uncertainty permeates these processes and products. Plans to test these artifacts, therefore, will carry their uncertainties forward. In particular, many testing activities, such as test result checking, are highly routine and repetitious and thus are likely to be error-prone if done manually, which introduces additional uncertainty. Humans carry out test planning activities at an early stage of development, thereby introducing uncertainties into the resulting test plan. Also, test plans are likely to reflect uncertainties that are, as described above, inherent in software artifacts and activities. Care must be taken during test planning to decide on the method of results comparison. Oracles, also, are required for validation and the nature of an oracle depends on several factors under the control of the test designer and automation architect [14,20]. Different oracles may be used for a single automated test and a single oracle may serve many test cases. If test results are to be analyzed, some type of oracle is required: a) oracle that gives exact outcome for every program case, b) oracle that provides range of program outcome and c) oracle that can not provide program outcome for some test cases.

2.1 Estimating quality

One of the reasons why projects are late and exceed their budgets is because they have so many defects that they cannot be released to users. Many of these defects escape detection until late in the testing cycle when it is difficult to repair them easily. Then, the testing process stretches out indefinitely. The cost and effort for finding and fixing defects may be the largest identifiable IT system cost estimate, so defects cannot be ignored in the estimating process.

Defect potentials are the sum of errors that occur because of:

- Requirements errors
- Design errors
- Coding errors
- User documentation errors
- Bad fixes or secondary errors.

Defect levels affect IT project costs and schedules. The number and efficiency of defect removal operations can have major impacts on schedules, cost, effort, quality, and maintenance. Quality estimates must consider defect potentials, defect removal efficiency, and maintenance.

All system defects are not equally easy to remove. Requirements errors, design problems, and bad fixes tend to be the most difficult. At the time of delivery, defects originating in requirements and design tend to far outnumber coding defects. While a number of approaches can be used to help remove defects, formal design and code inspections have been found to have the highest defect removal efficiency.

Typically, more than 50% of the global IT population is engaged in modifying existing applications rather than developing new applications. Estimates are required for maintenance for initial releases to repair defects in software applications to correct errors and for enhancements to add new features to software applications.

3 IOSTP framework state-of-the-art methods implementation

Unlike conventional approaches to software testing (e.g. structural and functional testing) which are applied to the software under test without an explicit optimization goal, the **IOSTP** with embedded **R**isk **B**ased **O**ptimized **STP** (**RBOSTP**) approach designs an optimal testing strategy to achieve an explicit optimization goal, given a priori [6,22]. This leads to an adaptive software testing strategy. A non-adaptive software testing strategy specifies what test suite or what next test case should be generated, e.g. random testing methods, whereas an adaptive software testing strategy specifies what testing policy should be employed next and thus, in turn, what test suite or test case should be generated next in accordance with the new testing policy to maximize test activity efficacy and efficiency subject to time-schedule and budget constraints. The process is based on a foundation of operations research, experimental design, mathematical optimization, statistical analyses, as well as validation, verification, and accreditation techniques. The use of state-of-the-art methods and tools for planning, information, management, design, cost trade-off analysis, and modeling and simulation, Six Sigma strategy significantly improves STP effectiveness. Figure 1 graphically illustrates a generic **IOSTP** framework [12]**.**

The focus in this paper is description of IOSTP with embedded RBOSTP Approach to Testing Services that:

- Integrate testing into the entire development process
- Implement test planning early in the life cycle via Simulation based assessment of test scenarios
- Automate testing, where practical to increase testing efficiency
- Measure and manage testing process to maximize risk reduction
- Exploit Design of Experiments techniques (optimized design plans, Orthogonal Arrays etc.)
- Apply Modeling and Simulation combined with Prototyping
- Continually improve testing process by pro-active, preventive (failure mode analysis) Six Sigma DMAIC model
- Continually monitor Cost-Performance Trade-Offs (Risk-based Optimization model, Economic Value and ROI driven STP)

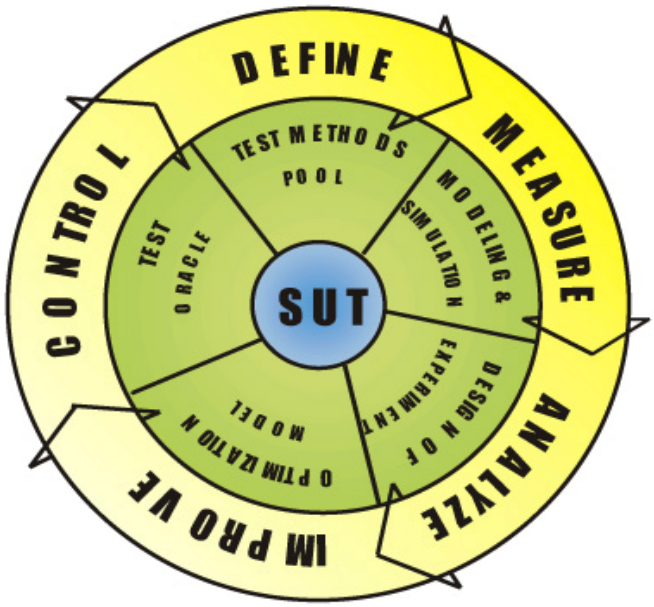


Fig. 1 Integrated and optimized software testing process (IOSTP) framework [12]

Framework models are similar to the structural view, but their primary emphasis is on the (usually singular) coherent structure of the whole system, as opposed to concentrating on its composition. IOSTP framework

model targeted specific software testing domains or problem classes described above. IOSTP is a systematic approach to product development (acquisition) which increases customer satisfaction through a timely collaboration of necessary disciplines throughout the life cycle. Successful definition and implementation of IOSTP can result in:

- Reduced Cycle Time to Deliver a Product
- Reduced System and Product Costs
- Reduced Risk
- Improved Quality

4 Modeling, simulation, and design of experiments for software testing

The IOSTP framework is a multi disciplinary integrated engineering solution to the testing process incorporating modeling and simulation, design of experiments, software measurement, and the Six Sigma approach to software test process quality assurance and control, as depicted in Figure 1 [12]. Unlike conventional approaches to software testing (e.g. structural and functional testing), which are applied to the software under test without an explicit optimization goal, the IOSTP approach designs an optimal testing strategy to achieve an explicit optimization goal, given a priori. This leads to an adaptive software testing strategy. A non-adaptive software testing strategy specifies what test suite or what next test case should be generated, e.g. random testing methods, whereas an adaptive software testing strategy specifies what testing policy should be employed next and thus, in turn, what test suite or test case should be generated next [6] in accordance with the new testing policy to maximize test activity efficacy and efficiency subject to time-schedule and budget constraints. The process is based on a foundation of operations research, experimental design, mathematical optimization, statistical analyses, as well as validation, verification, and accreditation techniques.

The focus in this section of the paper is on the application of M&S and DOE to minimize test suite size dramatically through black box scenario testing for the ATTRS real-time embedded software application, and also on using M&S as a test oracle in this same case study.

4.1 Model-based testing through simulation

As far as this paper is concerned, computer-based simulation is "the process of designing a computerized model of a system (or process) and conducting experiments with this model for the purpose either of understanding the behaviour of the system or of evaluating various strategies for the operation of this system"[23,38]. Put simply, a simulation allows one to develop a logical abstraction (an object), and then examine how that object behaves under differing stimuli. Simulation can provide insights into the designs of, for example, processes, architectures, or product lines before significant time and cost have been invested, and can be of great benefit in support of the testing process and training therein. There are several distinct purposes for computer-based simulation. One is to allow the creation of a physical object or system such as the ATTRS, as a logical entity in code. It is practical (and faster) to develop a simulator for testing physical system design changes. Changes to the physical system can then be implemented, tested, and evaluated in the simulation. This is easier, cheaper, and faster than creating many different physical engines, each with only slightly different attributes. Because of these features, network-based simulation tools allow one to develop large detailed models quite rapidly. The focus thus becomes less on the construction of a syntactically correct model and more on the model's semantic validity and the accuracy of its numerical driver. The simulation tools in today's market place, such as SLAM II, SIMSCRIPT, SIMAN, GPSS, PowerSim, MATLAB, etc, are robust and reasonably inexpensive. Unfortunately, before a simulation can be of benefit, a model of the system must be developed that allows the simulation developer to construct the computer-based simulation. Modeling is the first step—the very foundation of a good simulation. If the model is valid, it provides an opportunity to study system phenomena in a controlled manner, which may be very difficult otherwise due to an inability to control the variables in a real system.

However, parts of the simulation might not have well known interactions. In this case, one of the goals of simulation is to determine the real-world interactions. To make sure that only accurate interactions are captured, the best method is to start with a simple model, and ensure that it is correct and representative of the real world. Next, the interactions and complexity can be increased iteratively, validating the model after each increment. Additional interactions can be added until an adequate model is created that meets the desired needs. Unfortunately the previous description implies that clearly identified needs are available. This requires a valid statement of requirements.

It also requires planning for validation of the model. As in creating any software product, requirements and needs must be collected, verified, and validated. These steps are just as important in a simulation as they are in any system. A system that has not been validated by field-testing against the real world could produce invalid results. Abstraction and validation are equally necessary to create a reliable model that correctly reflects the real world, and also contains all attributes necessary to make the model a useful tool for prediction. The steps of abstraction and validation are in themselves, however, not totally sufficient to create a valid and usable model. Other steps are necessary to create a model that is of sufficient detail to be useful. These steps that describe

the process of producing and using a dynamic simulation are described by Cristie [3].

One of the most important problems facing the developer of a real-world simulation is that of trying to determine whether the simulation model is an accurate representation of the actual system being studied. In M&S based systems acquisition, computer simulation is used throughout the development and deployment process, not just as an analysis tool, but also as a development, integration, test, verification and sustainment resource. Because of this, verification and validation (V&V) in the simulation development process are most important tasks. If the model is to be credible, and a predictor of future behaviour, it is critical that it is validated [3,6].

Modeling and simulation (M&S) techniques are a cost-effective approach to reduce the time, resources and risks, and to improve the quality, of acquired systems.

Validated M&S can support all phases of IOSTP, and should be appropriately applied throughout the software life cycle for requirements definition; program management; design and engineering; test and evaluation; and field operation and maintenance presented in our works [6-9, 14-22]. The model of SUT is designed to consider the system as a whole and to allocate resources to maximize the benefits and credibility of applied M&S class associated with the overall IOSTP with embedded RBOSTP program.

Test engineers and operational evaluators play a key role in evaluating system/software performance throughout the life cycle. They identify:

- The information needed and when it must be available. This includes understanding the performance drivers and the critical issues to be resolved.
- The exact priority for what must be modeled first, then simulated, and then tested. This includes learning about the subcomponent level, the components, and the system level.
- The analysis method to be used for each issue to be resolved. Timing may have a significant effect on this. The design function can use models before any hardware is available. It will always be more expedient to use models and simulations at the early stage of system design. However, the design itself may be affected by operational considerations that require examination of real tests or exercise data. It will, given the training and logistic information required of systems today, be prudent in the long run to develop appropriate models and simulations.
- The data requirements and format for the analysis chosen. Included in this determination is the availability of instrumentation, not only for collecting performance data, but also for validating appropriate models and simulations.

Models and simulations can vary significantly in size and complexity and can be useful tools in several respects. They can be used to conduct predictive analyses for developing plans for test activities, for assisting test planners in anticipating problem areas, and for comparison of predictions to collected data. Validated models and simulations can also be used to examine test article and instrumentation configurations, scenario differences, conduct *what-if* tradeoffs and sensitivity analyses, and to extend test results.

Testing usually provides highly credible data, but safety, environmental, and other constraints can limit operational realism, and range cost and scheduling can be limiting factors. Modeling, especially credible model building may be very expensive although M&S can be available before hardware is ready to test. A prudent mix of simulation and testing is needed to ensure that some redesign is possible (based on M&S) before manufacturing begins.

While developing the Software Test strategy, the program office with test-team must also develop a plan to identify and fund resources that support the evaluation. In determining the best source of data to support analyses, IOSTP with embedded RBOSTP considers credibility and cost. Resources for simulations and software test events are weighed against desired confidence levels and the limitations of both the resources and the analysis methods. The program manager works with the test engineers to use IOSTP with embedded RBOSTP to develop a comprehensive evaluation strategy that uses data from the most cost-effective sources; this may be a combination of archived, simulation, and software test event data, each one contributing to addressing the issues for which it is best suited.

Success with IOSTP with embedded RBOSTP does not come easy, nor is it free. IOSTP with embedded RBOSTP, by integrating M&S with software testing techniques, provides additional sources of early data and alternative analysis methods, not generally available in software tests by themselves. It seeks the total integration of IOSTP with embedded RBOSTP resources to optimize the evaluation of system/software worth throughout the life cycle. The central elements of IOSTP with embedded RBOSTP are: the acquisition of information that is credible; avoiding duplication throughout the life cycle; and the reuse of data, tools, and information.

4.2 Design of experiments for software testing

A wide variety of approaches, methods, and analysis techniques, known collectively as design of experiments, has been around for many decades and is well documented in many books like those of Box et al. [44] or Montgomery [45]. One of the principal goals of experimental design is to estimate how changes in input

factors affect the results, or responses, of the experiment.

While these methods were developed with physical experiments in mind (like agricultural or industrial applications), they can fairly easily be used in computer-simulation experiments, and software testing as well [6,31]. Careful planning of how to experiment with a simulation model generally repays big dividends in terms of how effectively one can learn about the system and how one can exercise the model further. This section looks at such experimental design issues in the broad context of a simulation project. The term 'experimental design' has specific connotations in its traditional interpretation, and some of these will be mentioned below. However, the issues of planning simulations will also be covered in a broader context, which considers the special challenges and opportunities one has when conducting a computer-based simulation experiment rather than a physical experiment. This includes questions of the overall purpose of the project, what the output performance measures should be, how to use the underlying random numbers, measuring how changes in the inputs might affect the outputs, and searching for some kind of optimal system configuration. Specific questions of this type might include:

- What model configurations should be run?
- How long should the runs be?
- How many runs should be performed?
- How should the output be interpreted and analyzed?
- What is the most efficient way to perform the runs?

These questions, among others, are what one has to deal with when trying to design simulation experiments. In a stochastic simulation like the ATTRS case study, one would really like to know all about the distributions of the output variables. Usually one has to settle for various summary measures of the output distributions. Traditionally, people have focused on estimating the expected value (or mean) of the output variable distribution and this can be of great interest.

For people without statistical training, it can be difficult to organize information about the system under study in a way that aids the design of the experiment. To help clarify this process, the design task itself can be decomposed into five separate steps.

1. Define the goals of the experiment.
2. Identify and classify the independent and dependent variables.
3. Choose a probability model for the behaviour of the simulation model.
4. Choose an experiment design plan.
5. Validate the properties of the chosen design.

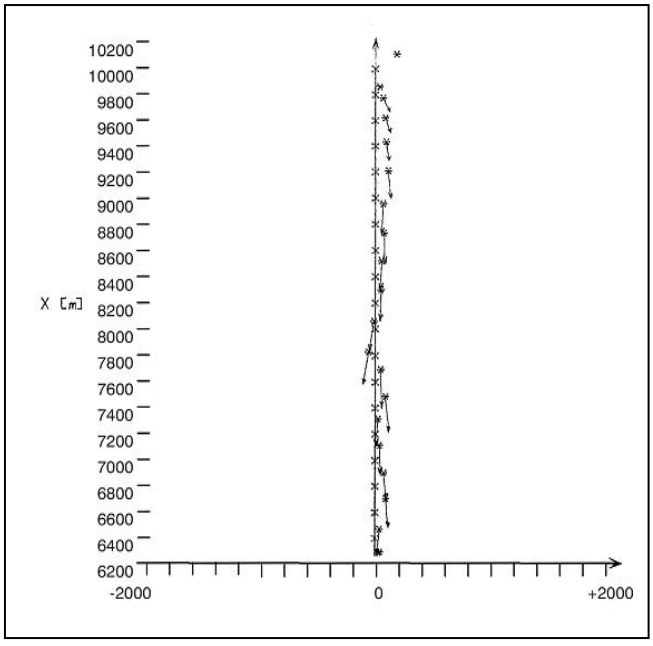
After defining the goals, an appropriate DOE plan should be chosen. One determines the number of distinct model settings to be run and the specific values of the factors for each of these runs. There are many strategies for selecting the number of runs and the factor settings for each run. These include random designs, combinatorial designs, mixture designs, sequential designs, factorial designs, and optimal designs (one of the latter is chosen in the ATTRS case study in this section). In this paper one particular method of experimental design applied to software testing is emphasized, i.e. orthogonal array-based robust testing (OART) [6,31], which is based on Taguchi robust design; it has a mathematical foundation in linear algebra (specifically, Galois field theory) and began with Euler as Latin squares. This method is exploited in ATTRS field testing [6]. Black-box testing of software components and systems is indispensable and requires test data for all input parameters. The number of test cases needed for exhaustive testing, i.e. for all possible combinations of input data is usually extremely large – almost always too large for the allocated testing resources that are always limited. The only solution is intelligent test case generation to cut down costs and improve the quality of testing. The OART method has been used to test software from many diverse industries, e.g., telecommunications, automotive, and electromechanical systems. The users have typically reported a factor of 2-to-1 or better productivity improvement compared to traditional testing methods [6]. The number of tests needed for this method is similar to the number of tests needed for the one-factor-at-a-time method, and with a proper software tool, the effort to generate the test plan can be small. Its ability to find faults is much better than the one-factor-at-a-time method and approaches 100 percent, especially when used in conjunction with code coverage analysis tools. The test cases generated by this method have the highest effectiveness, measured in terms of the number of faults detected per test. As an illustration, consider a simple scenario consisting of a system under test with 4 components (factors designated as A, B, C and D respectively) each of which has 3 possible elements (levels designated as 1, 2, 3):

There were three input variables in the field tests of ATTRS in our case study: A - the target speed with three levels (1 was 0 m/s, 2 was 150 m/s, 3 was 300m/s), B - the radar probability of target detection with three levels (1 was 70%, 2 was 80%, 3 was 100%, although the latter cannot be controlled accurately), and C - the radial acceleration of target manoeuvre with three levels (1 was 0g, 2 was 3g, 3 was 6g). The L9 orthogonal array algorithm was applied as shown in Table 1 (the last column from the original L9 has been omitted). Results of the field test and simulation-based experiments that were run afterwards are shown in Table 2. Mean and standard deviation error (μ, σ) of the differences between the simulation-based experiments and the field trials were compared and it was found that the

confidence interval of differences for $\beta = 1-\alpha = 0.95$ were in the range of field trial experimental error. If field testing is used as the comparative system, the calibration radar then has 10 times lower error in target position estimation. The results of both experiments verify that the required automatic target tracking quality of ATTRS was fulfilled. Some computer-base simulation experiments and field trial results are shown in Figure 2 and 3. An exhaustive testing suite would require the set of test cases to be executed for $3^4 = 81$ possible configurations using the one-factor-at-a-time method. The OART algorithm L9 [6] calculates only 9 combinations that are already sufficient to cover all pair-wise component interactions as shown in Table 1. The rows correspond to test cases; the columns correspond to the test parameters. Thus, the first test case comprises level 1 for each parameter, i.e. it represents the combination A1, B1, C1, D1. The second test case comprises combination A1, B2, C2, D2, etc. An orthogonal array (OA) has the balancing property that, for each pair of columns, all parameter-level combinations occur an equal number of times. In OA L9, there are nine parameter-level combinations for each pair of columns, and each combination occurs once. By applying the algorithm based on orthogonal arrays and covering arrays, the overall number of test cases can be dramatically reduced compared to exhaustive testing, yet with the certainty that test coverage of all pair-wise input parameter or software component (i.e. configuration) combinations is achieved as in the ATTRSt example. Suppose that another SUT has 13 input parameters each with 3 input parameter domains.

Table 1. L9 Orthogonal Array Design Experiment plan

Test Case No.	**Test Parameters**			
	A	**B**	**C**	**D**
1	**1**	**1**	**1**	**1**
2	**1**	**2**	**2**	**2**
3	**1**	**3**	**3**	**3**
4	**2**	**1**	**2**	**3**
5	**2**	**2**	**3**	**1**
6	**2**	**3**	**1**	**2**
7	**3**	**1**	**3**	**2**
8	**3**	**2**	**1**	**3**
9	**3**	**3**	**2**	**1**



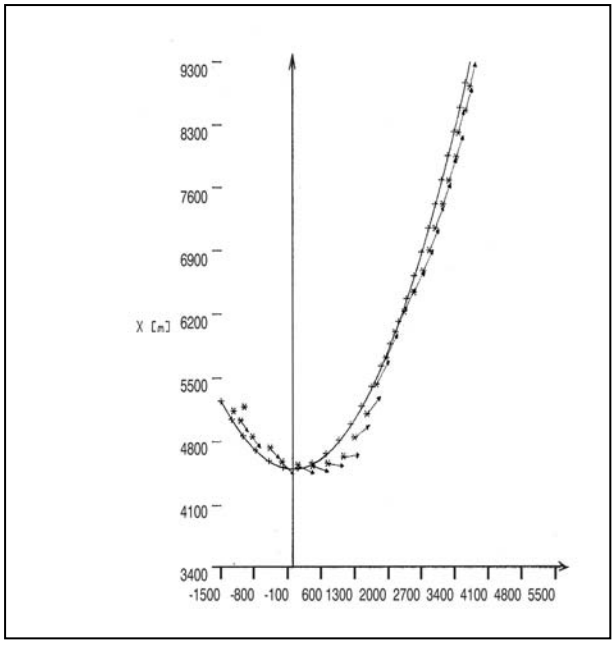


Fig. 2. Simulation result of straight-line target attack: target speed v=150 [m/s], Pd=0.94, and parabolic-curvilinear trajectory: v=300 [m/s], Pd=1.0, manoeuvre 6g.

Then exhaustive testing would require $3^{13} = 1{,}594{,}323$ possible input parameter combinations. The OART algorithm calculates just 15 configurations that are sufficient to cover all pair-wise input parameter interactions. The OART method is applicable to unit testing, integration and system scenario black-box testing, configuration testing, interoperability testing, and web testing.

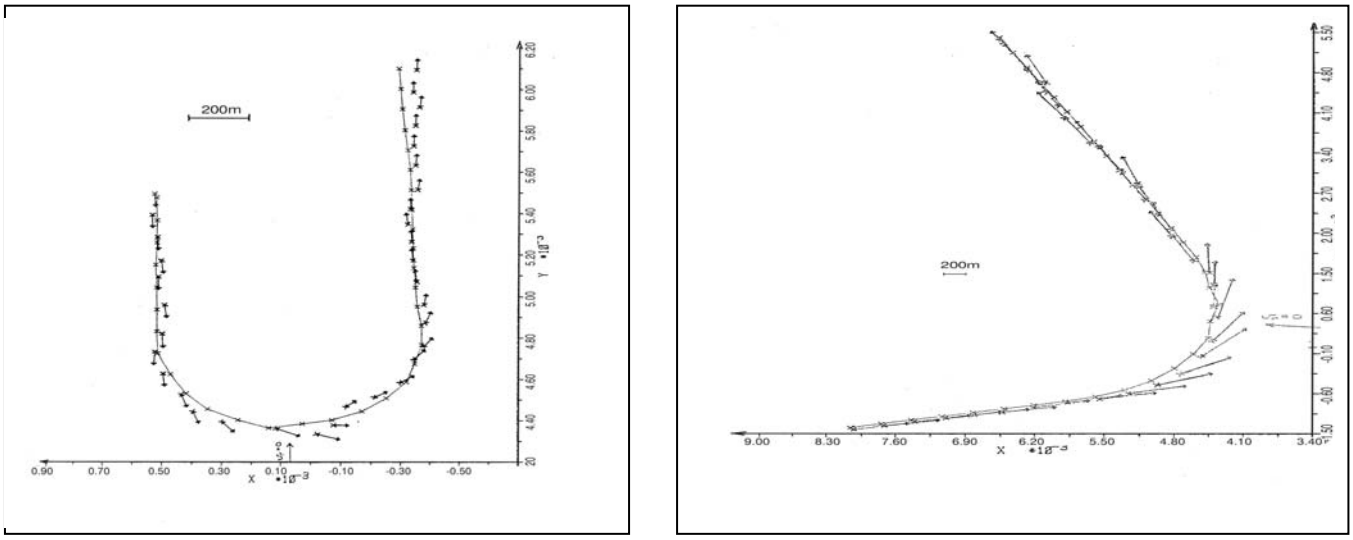


Fig. 3. Some field trial results with same conditions as in simulation cases shown on Fig. 2

Table 2. L9 OART compared results

No. T	Test Results							
	Field test				Simulation			
	μ_R [m]	σ_R [m]	μ_θ [mrad]	σ_θ [mrad]	μ_R [m]	σ_R [m]	μ_θ [mrad]	σ_θ [mrad]
1	105	70	20	10	95	65	16	8
2	78	23	10	4	73	21	9	4
3	77	21	9	3	72	20	7	3
4	104	68	18	9	92	62	15	7
5	82	41	13	8	80	40	12	7
6	76	19	8	3	69	17	8	2
7	102	72	17	9	91	61	15	7
8	53	36	6	12	54	35	6	10
9	43	34	3	11	42	31	4	11

5 Using simulation as a test oracle

There are many areas where simulation can be applied to support software development and acquisition. Such areas include requirements specification, process improvement, architecture trade-off analysis, and software testing. Of particular interest here is the use of simulation as a test oracle as described in our articles [6,12,14,20]. The term oracle may be used to mean several things in testing—the process of generating expected results, the expected results themselves, or the answer to whether or not the actual results are as expected.

The testing process is typically systematic in test data selection and test execution. For the most part, however, the effective use of test oracles has been neglected, even though they are a critical component of an effective method for designing in software quality for testability that should include the concurrent development of test oracles in the testing process [6,12,42]. Oracle development can represent a significant effort, which may increase design and implementation costs; however; overall testing and maintenance costs should be reduced. Oracle development must therefore be carefully integrated into the software development/testing life cycle. Oracles must be designed, verified and validated for unit testing, through subsystems testing (integration testing) up to system testing in a systematic, disciplined and controlled manner. Test oracles prescribe acceptable behaviour for test execution. In the absence of judging test results with oracles, i.e. use of a reference system, testing does not achieve its goal of revealing failures or assuring correct behaviour in a practical manner – manual result checking is neither reliable nor cost-effective.

In much of the research literature on software test case generation or test set adequacy, the availability of oracles is either explicitly or tacitly assumed, but applicable oracles are not described. In current industrial software testing practice, the oracle is often a human being. The most significant oracle characteristics are:

Relying on a human to assess program behaviour has two evident drawbacks: accuracy and cost. While the human "eyeball oracle" has an advantage over more technical means in interpreting incomplete, natural-language specifications, humans are prone to error when assessing complex behaviour or detailed, precise specifications, and the accuracy of the eyeball oracle drops precipitously with increases in the number of test runs to be evaluated. Automated test oracles are required for running large numbers of tests. An ideal test oracle would satisfy desirable properties of program specifications, such as being complete but avoiding over-specification, while also being efficiently checkable. These properties are in conflict, and many of the interesting issues and trade-offs in the design of test oracle systems come in various ways as those tensions

between desirable properties of specifications and necessary properties of implementations are resolved.
When simulation is used as an oracle, it tends to take on more of the capabilities of a "real" specification language, or provides more powerful facilities for deriving run-time checks from external specifications. Approaches to bridging the gap usually involve some combination of restricting the language to what can be effectively or efficiently checked (e.g. disallowing quantification over infinite sets), mapping implementation entities to specification-level entities, and/or taking advantage of the peculiarities of particular application domains.
The research literature on test oracles is a relatively small part of the entire research literature on software testing Weyuker has set forth some of the basic problems and argued that truly general test oracles are often unobtainable [43]. Some older proposals base their analysis either on the availability of pre-computed input/output pairs [42] or on a previous version of the same program, which is presumed to be correct. The latter hypothesis sometimes applies to regression testing, but is not sufficient in the general case. The former hypothesis is usually too simplistic: being able to derive a significant set of input/output pairs would imply the capability of analyzing the system outcome. Computer-based simulation at various levels of SUT abstraction can serve as a test oracle, which is able to derive a significant set of input/output pairs.
A simulation-centric development environment is a developer's dream, especially in the development of embedded real-time systems. It directly provides the many advantages inherent to software over hardware, reducing costs, risk, and schedule. These advantages are now briefly enumerated.
Control and early availability: The simulator can be executed a single-step at a time; it provides breakpoints triggered by expressions of arbitrary complexity; it can checkpoint everything while recording all intermediate variables that can serve as test cases for input variables or expected output variables depending on the simulated level of system abstraction; and it can be dramatically reconfigured in a matter of moments. The simulator generally has a shorter development cycle than hardware of comparable fidelity, and so can be available earlier in the project life cycle. Developers can begin work on a full-fidelity platform, rather than constructing ad hoc scaffolding for their individual areas of concern. Hardware/software design trade-offs can be explored before hardware is developed – no more tedious days with a logic analyzer trying to find an interrupt problem or race condition. This aspect of simulation satisfies several of the required characteristics of test oracles: predicting speed and short oracle execution time.
Visibility: The full state of the system, at any instant, is easily examined, check-pointed, logged, or modified. Formal analysis for deadlock potential or race conditions is feasible. Cache patterns can be analyzed in detail. Abuse of the hardware (e.g. use of an uninitialized variable, or failure to save the entire state during an interrupt, or setting both of the "never set this bit and that bit at the same time" bits) can be detected online and in situ.
Extensibility: The behaviour of the simulator is easily extended, especially with the use of modern scripting subsystems that give the sophisticated power-user access to the full feature set of the tool. This aspect of simulation satisfies the required characteristic of test oracle evolution through changes in the SUT.
Automated control: Hands-off and repeatable testing is encouraged because it is easy and natural to control the execution of the simulator via other software. Nightly integration testing is reasonable. Testing staff and schedule can be greatly reduced.
Modularity: A simulator can be constructed in a highly modular fashion, so that more urgently needed models are built first and available to the users before the entire simulator has been built and validated.

6 Trade-Off Studies and Prioritization, Cost-Performance trade-offs i.e. IOSTP optimization model

- **Trade-Off Studies and Prioritization**

To fully leverage the IOSTP framework, design trade-offs that optimize system requirements vs. cost should be performed during the earliest phases of the software life cycle. Product/process performance parameters can be traded-off against design, development, testing, operations and support, training requirements, and overall life cycle costs. System attributes such as mission capability, operational safety, system readiness, survivability, reliability, testability, maintainability, supportability, interoperability and affordability also need to be considered. Quality Function Deployment (QFD) is one technique for evaluating trade-off scenarios [13,18]. It is predicated on gaining an understanding of what the end user really needs and expects. The QFD methodology allows for tracking/tracing trade-offs through various levels of the project hierarchy, from requirements analysis, through the software development process, to operational and maintenance support. Testing represents a significant portion of the software development efforts that must be integrated in parallel to QFD. Risk-Based Optimization of Software Testing Process i.e. RBOSTP is part of a proven and documented IOSTP [12,17] designed to improve the efficiency and effectiveness of the testing effort assuring the low project risk of developing and maintaining high quality complex software systems within schedule and budget constraints [21,22]. Basic considerations of RBOSTP are described in [21] article and some RBOSTP implementation issues, experience results are presented in [22]. In our article [22], we

describe how RBOSTP combines Earned (Economic) Value Management (EVM) and Risk Management (RM) methodology through simulation-based software testing scenarios at various abstraction levels of the system/software under test activities to manage stable (predictable and controllable) software testing process at lowest risk, at an affordable price and time.

Predicting future outcomes

Both EVM and RM attempt to predict the future outcome of the project, based on information currently known about the project. For EVM this is achieved using calculated performance indices, with a range of formulae in use for calculating Estimate at Completion (EAC). Most of these formulae start with the Actual Cost of Work Performed to date (ACWP, or Actual Cost AC), and add the remaining budget adjusted to take account of performance to date (usually using the Cost Performance Index CPI, or using a combined Performance Efficiency Factor based on both CPI and SPI). These calculations of the Estimate to Complete (ETC) are used to extrapolate the ACWP plot for the remainder of the project to estimate where the project might finally end (EAC), as shown in Fig. 4. However calculating EAC in this way does not take explicit account of the effect of future risks on project outcome. One simple way to do this is by adding an amount into the EAC calculation to account for risk-weighted contingency or management reserve. RM predicts a range of possible futures by analyzing the combined effect of known risks and unknown uncertainty on the remainder of the project. When an integrated covering the uncompleted portion of the project, as in Fig. 5. In the same way that the initial spend baseline should be determined using both risk and earned value data, the remaining element of the project should also be estimated using both sets of information.

It is also possible to use risk analysis results to show the effect of specific risks (threats or opportunities) on project performance as measured by earned value. Since the risk analysis includes both estimating uncertainty and discrete risks, the model can be used to perform "what-if" scenario analysis showing the effect of addressing particular risks. For example, if a key threat is modelled using a probabilistic branch, a "what-if" analysis can set the probability of the threat occurring to zero, simulating the result if that risk is removed. Similarly the effect of capturing key opportunities can also be shown.

- **Cost-Performance trade-offs i.e. IOSTP optimization model**

Activity-Based Costing (ABC), which focuses on those activities that bring a product to fruition, is considered a valuable tool for IOSTP framework cost analysis. Costs are traced from IOSTP activities to products, based on the consumption of each product of those activities.

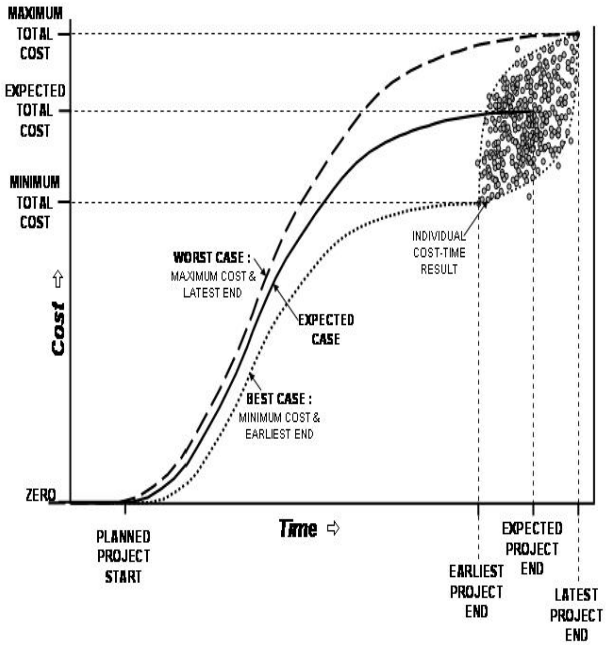


Fig. 4 Risk-Based cumulative spend model (BCWS or PV)

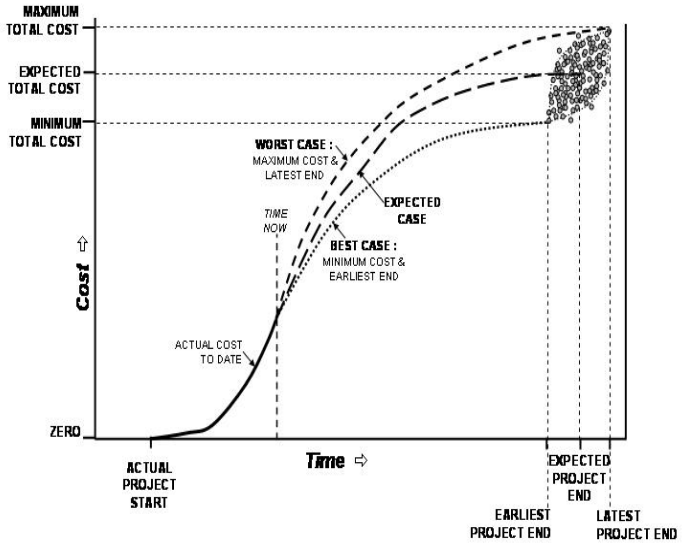


Fig. 5 Risk-Based calculation of remaining project performance

The cost of the product is measured as the sum of all activities performed, including overheads, capital costs, etc. Stakeholders are most interested in the benefits that are available and the objectives that can be achieved. The benefit-based test reports present this clearly. Project management is most interested in the risks that block the benefits and objectives. The benefits-based test reports focus attention on the blocking risks so that the project manager can push harder to get the tests that matter through [22].If testers present risk and benefits based test reports, the pressure on testers is simply to execute the outstanding tests that provide information on risk. The pressure on developers is to fix the faults that block the tests and the risks of most concern. Testers need not worry so much about justifying doing more

testing, completing the test plan or downgrading "high severity" incidents to get through the acceptance criteria. The case for completing testing is always self-evident: has enough test evidence been produced to satisfy the stakeholders' need to deem the risks of most concern closed? The information required by stakeholders to make a release decision with confidence might only be completely available when testing is completed. Otherwise, they have to take the known risks of release. How good is our testing? Our testing is good if we present good test evidence. Rather than getting so excited about the number of faults we find, our performance as testers is judged on how clear is the test evidence that we produce. If we can provide evidence to stakeholders for them to make a decision at an acceptable cost and we can squeeze this effort into the time we are given, we are doing a good testing job. This is a different way of thinking about testing. The definition of good testing changes from one based on faults found to one based on the quality of information provided. Consider what might happen if, during a test stage, a regression test detects a fault. Because the test fails, the risk that this test partially addresses becomes open again. The risk-based test report may show risks being closed and then re-opened because regression faults are occurring. The report provides a clear indication that things are going wrong – bug fixes or enhancements are causing problems. The report brings these anomalies directly to the attention of management. Main task is development of a versatile Optimization Model (OM) for assessing the cost and effectiveness of alternative test, simulation, and evaluation strategies. The System/Software under test and corresponding testing strategy-scenario make up a closed-loop feedback control system. At the beginning of software testing our knowledge of the software under test is limited. As the system/software testing proceeds, more testing data are collected and our understanding of the software under test is improved. Software development process parameters (e.g. software quality, defect detection rates, cost etc.) of concern may be estimated and updated, and the software testing strategy is accordingly adjusted on-line. The important ingredients in successful implementation of IOSTP with embedded RBOSTP are (1) a thoughtful and thorough evaluation plan that covers the entire life cycle process, (2) early identification of all the tools and resources needed to execute that software test and evaluation process plan and timely investment in those resources, (3) assuring the credibility of the tools to be employed, and (4) once testing is accomplished, using the resulting data to improve the efficacy of the test event, models and simulations. In order to provide stable (controlled and predictable) IOSTP we integrated two of the leading approaches: Earned (Economic) Value Management (EVM) and Risk Management (RM). These stand out from other decision support techniques because both EVM and RM can and should be applied in an integrated way across the organization that some authors [40,41], recently recognized as Value-Based Testing. Starting at the project level, both EVM and RM offer powerful insights into factors affecting project performance. While this information is invaluable in assisting the project management task, it can also be rolled up to portfolio, program, departmental or corporate levels, through the use of consistent assessment and reporting frameworks. This integration methodology operates at two levels with exchange of information. The higher, decision making level takes into account the efficacy and costs of models, simulations, and other testing techniques in devising effective programs for acquiring necessary knowledge about the system under test. The lower, execution level considers the detailed dimensions of the system knowledge sought and the attributes of the models, simulations, and other testing techniques that make them more or less suitable to gather that knowledge. The OM is designed to allow planners to select combinations of M&S and/or tests that meet the knowledge acquisition objectives of the program. The model is designed to consider the system as a whole and to allocate resources to maximize the benefits and credibility of applied M&S class associated with the overall IOSTP with embedded RBOSTP program [22].

- **Risk Management implemented in IOSTP**

In order to implement RBOST we use one of favorite schedule risk software includes **RISK+** from **C/S Solutions, Inc** an add-in to Microsoft Project at www.cs-solutions,com**.** We suggest, also, **@RISK** for **Project Professional** from **Palisade Corporation**, also an add-in to Project at www.palisade.com, **Pertmaster** from **Pertmaster LTD (UK)** at www.pertmaster.com reads **MS Project** and **Primavera** files and performs simulations. **Pertmaster** is substituting for an older product, **Monte Carlo** from **Primavera Systems** which links to **Primavera Project Planner (P3)** www.primavera.com . Risk+ User's Guide provides a basic introduction to the risk analysis process. The risk analysis process is divided into following five steps, as depicted in Fig. 6.

1. The **first step** is to plan our IOSTP with embedded **RBOSTP** project. It is important to note that the project must be a complete critical path network to achieve meaningful risk analysis results. Characteristics of a good critical path network model are:

- ✓ There are no constraint dates.
- ✓ Lowest level tasks have both predecessors and successors.
- ✓ Over 80% of the relationships are finish to start.

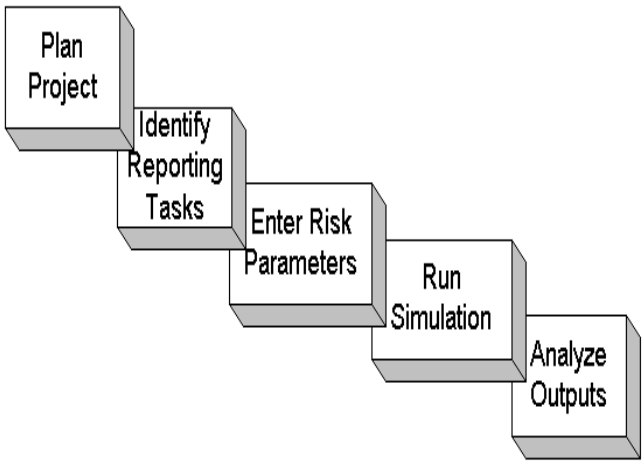


Fig. 6 The risk analysis process implemented in RBOSTP

In the Risk + tutorial, we use the DEMO.MPP project file, which has the characteristics of a good critical path network model. Since the scheduling process itself is well covered in the Project manual we won't repeat it here.

2. The second step is to identify the key or high risk tasks for which statistical data will be collected. Risk + calls these Reporting Tasks. Collecting data on every task is possible; however, it adds little value and consumes valuable system resources. In this step you should also identify the Preview Task to be displayed during simulation processing.

3. The third step requires the entry of risk parameters for each non-summary task. For each non-summary task enter a low, high, and a most likely estimate for duration and/or cost. Next, assign a probability distribution curve to the cost and duration ranges. The probability distribution curve guides Risk + in the selection of sample costs and durations within the specified range. See the section titled "Selecting a Probability Distribution Curve" in the Risk+ manual for more information on selecting a curve type. Update options such as "Quick Setup" and "Global Edit" can dramatically reduce the effort required to update the risk parameters.

4. The fourth step is to run the risk analysis. Enter the number of iterations to run for the simulation, and select the options related to the collection of schedule and cost data. For each iteration of the simulation, the Monte Carlo engine will select a random duration and cost for each task (based upon its range of inputs and its probability distribution curve), and recalculate the entire schedule network. Results from each iteration are stored for later analysis.

5. The final and fifth step is to analyze the simulation results. Depending on the options selected, Risk + will generate one or more of the following outputs:

- Earliest, expected, and latest completion date for each reporting task
- Graphical and tabular displays of the completion date distribution for each reporting task
- The standard deviation and confidence interval for the completion date distribution for each reporting task
- The criticality index (percentage of time on the critical path) for each task
- The duration mean and standard deviation for each task
- Minimum, expected, and maximum cost for the total project
- Graphical and tabular displays of cost distribution for the total project
- The standard deviation and confidence interval for cost at the total project level

Risk + provides a number of predefined reports and views to assist in analyzing these outputs. In addition, you can use Project's reporting facilities to generate custom reports to suit your particular needs.

Project cost and schedule estimates often seem to be disconnected. When the optimistic estimate of schedule is retained, in the face of the facts to the contrary, while producing an estimate of cost, cost is underestimated. Further, when the risk of schedule is disregarded in estimating cost risk, that cost risk is underestimated. In reality cost and schedule are related and both estimates must include risk factors of this estimating process because of uncertainty of test tasks' cost and time estimation that RBOSTP optimization model testing includes and described by equation (1) with constraints in our article [22]. The strategy for integration of schedule and risk begins with an analysis of the risk of the schedule [21,22].

- **Critical path method scheduling - some important reservations**

The critical path method (CPM) is a key tool for managing project schedules. A schedule "network" represents the project strategy or plan. CPM computes the shortest project completion duration and earliest completion date. The longest path through the network is called the "critical path." According to CPM, any delay on the critical path will delay the project. On the one hand, CPM is traditional and well-accepted. It is essential for developing the logic of the project work and for managing the day-to-day project activities. On the other hand, the accuracy of CPM completion date forecast depends on every task taking just as long as its duration estimate indicates – in short, **CPM is accurate only if everything goes according to plan**. Experienced project managers realize that real projects do not often go according to plan:

- The estimates of activity durations are at best careful estimates of future work and at worst just guesses or unrealistically short, calculated by how much time you have rather than how long the work takes.
- Even if the activity durations are most likely estimates, the CPM completion date is not the most likely project completion date.
- The path identified as the "critical path" may not be the one that will be most likely to delay the project and which may need management attention.

Is there some method of analysis and planning that can improve the accuracy of our scheduling? Yes, there is, and it is **schedule risk analysis in three steps**.

Case 1: Three Steps to a Successful Schedule Risk Analysis
The three steps to a successful risk analysis are described. They are: (1) create the CPM schedule for the project, (2) estimate the uncertainty in the activity durations with low and high ranges, and (3) perform a risk analysis of the schedule, using a Monte Carlo simulation method.

Step 1: A CPM Schedule
Assume a simple project with two activities and a finish milestone. Suppose the durations are set at 40 working days for A101 and 70 working days for A102. If this project is scheduled to start on January 2, 2000, CPM shows that this simple project will take 110 working days (40 + 70 = 110) and complete on June 2, 2000 (see Figure 7, below)

Step 2: The Activity Duration Ranges
To do a risk analysis we need to estimate duration ranges for each activity which are based on the low (optimistic) and high (pessimistic) scenarios for the work on the activity. High ranges, for instance, can be determined by examining the various things that could go wrong such as technical problems, site conditions, supplier delays, and permitting issues -- factors which are often called "risk drivers."
These duration ranges are determined by searching interviews of the project manager and the staff who made the estimates, will manage the project and are familiar with the possible problems. The ranges of pessimistic (Max Rdur) and optimistic (Min Rdur) durations for the two-activity schedule are shown in Fig. 7.

Step 3: Simulate the Project Schedule
Once the activities' duration ranges and distributions have been determined, the schedule risk analysis can determine how risky the entire project schedule is.

- How likely we to overrun the completion are date of June 2, 2000? Is June 2 even the "most likely" date for this simple project? If not, what completion date is most likely?
- How many days are needed for a contingency to reduce the overrun risk exposure to an acceptable level?
- Which activities are the most likely to delay the project?

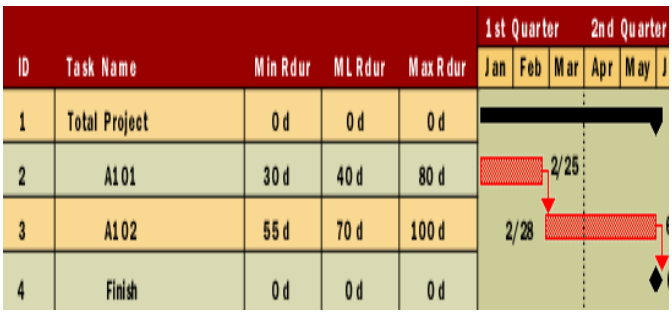


Fig. 7 Test events duration distribution

The most common method of determining schedule overrun risk is to simulate the project by solving (or iterating) it hundreds or thousands of times on the computer. This is called Monte Carlo simulation, and it combines the distributions of uncertain duration accurately.
Suppose that the risk analyst determines that 2,500 iterations will be sufficient for the accuracy needed. The result of that simulation is a cumulative likelihood distribution that represents the likelihood of the project completing on or before each possible date. This distribution is shown in Fig. 8 below:

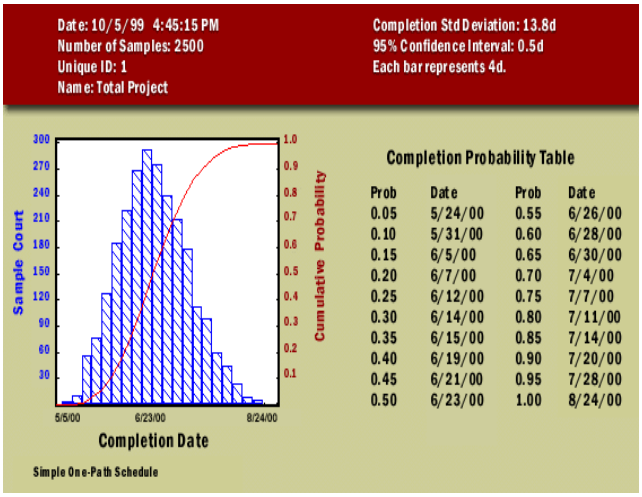


Fig. 8 The result of simulation of a cumulative likelihood distribution of completion date

From the risk analysis we can see:

- The CPM completion date of June 2, 2000 is between 10% and 15% likely to be adequate for this simple project. Placing confidence in completion by June 2 is very likely to get the contractor and customer in trouble.
- The most likely completion date is close to June 19, not June 2 as predicted by CPM. **The common sense notion, that adding "most likely" durations along a critical path will result in the most likely project completion date, is simply wrong, in all cases.**
- The average completion date is June 23, 2000. If this simple project were done 100 times, its average

completion would be about a 3-week overrun of the CPM duration, providing for the holidays.

- The results show that July 11, 2000 has an 80% likelihood of success. This is a level of protection from overruns that might be required for a conservative contractor or owner/customer. Hence, a 6-week contingency is needed to reduce the risk of overrun to an acceptable level for this conservative company.

The CPM project end-date of June 2 is highly optimistic. Any owner, customer or contractor who agrees to that date is in trouble now on this project. Without a risk analysis, the existence or degree of trouble is unknown.

Risk Analysis Topics - the Merge Bias

This is only a 2-activity project. Real life projects are subject to more risk than this.

Most projects have activities planned simultaneously along parallel paths. At the end of the project, and often at important internal milestones, these paths converge. Examples include; (a) piping, duct, framing and electrical work must be completed before an inspection can be conducted, or (b) several components that must be finished before systems integration and testing can be done. Most project overrun risk occurs at path convergence (or merge) points because projects can be delayed because a delay on any one of the paths will delay the work. This is the **"merge bias"** at work. To see the merge bias, consider expanding the simple schedule above to a 2-path project where the second path is exactly the same as the one in Figure 1 above. Clearly, CPM analysis shows that this project, too, will finish at the same time as the one-path schedule does, June 2, 2000. When we analyze the risk of this two-path schedule, however, we see that it is riskier because either path can cause an overrun. Compare Fig. 9 results to those in Fig. 8 above.

- The average completion date is now July 6, 2000, not June 23.
- The CPM date of June 2, 2000 is now less than 5% likely, not 10 – 15%.
- The 80th percentile is now July 20, 2000 rather than July 11.

These results reflect the working of the merge bias when parallel paths converge.

- **The role of the Six Sigma strategy in software development/testing process**

In order to assure controlled and stable (predictive) testing process in time, budget and software quality space we need to model, measure and analyze software-testing process by applying Six Sigma methodology across the IOSTP solution as presented in our works [15-19].

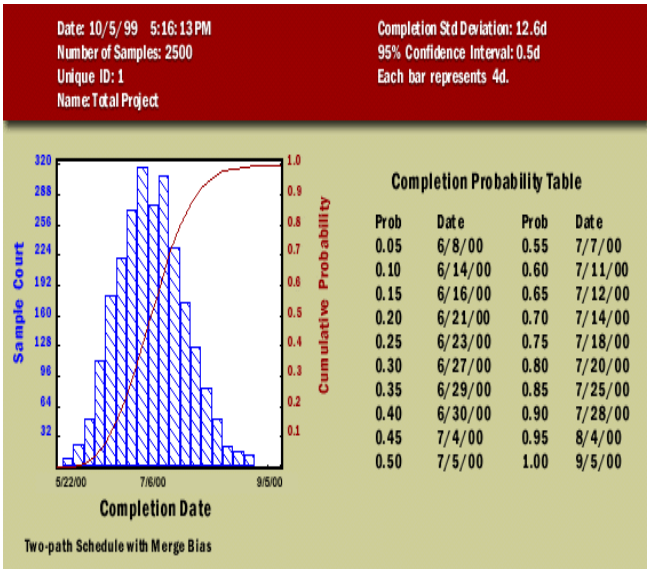


Fig. 9 Two-path schedule cumulative likelihood distribution of completion date simulation result

The name, Six Sigma, derives from a statistical measure of a process's capability relative to customer specifications. Six Sigma is a mantra that many of the most successful organizations in the world swear by and the trend is getting hotter by the day. Six Sigma insists on active management engagement and involvement, it insists on financial business case for every improvement, it insists on focusing on only the most important business problems, and it provides clearly defined methodology, tools, role definitions, and metrics to ensure success. So, what has this to do with software? The key idea to be examined in this article is the notion that estimated costs, schedule delays, software functionality or quality of software projects are often different than expected based on industry experience. Six Sigma tools and methods can reduce these risks dramatically i.e. Six Sigma (***6σ***) deployment in SDP/STP called DMAIC for "Define, Measure, Analyze, Improve, and Control", because it organizes the intelligent control and improvement of existing software test process [15-16]. Experience with 6σ has demonstrated, in many different businesses and industry segments that the payoff can be quite substantial, but that it is also critically dependent on how it is deployed. The main contribution of our [15-19] works is mapping best practices in Software Engineering, Design of Experiments, Statistical Process Control, Risk Management, Modeling & Simulation, Robust Test and V&V etc. to deploy Six Sigma to the STP. In order to significantly improve software testing efficiency and effectiveness for the detection and removal of requirements and design defects in our framework of IOSTP with deployed Six Sigma strategy, during 3 years of our 6σ deployment to STP we calculated overall value returned on each dollar invested i.e. ROI of 100:1.

7 Conclusions

In software development organizations, increased complexity of product, shortened development cycles, and higher customer expectations of quality proves that software testing has become extremely important software engineering activity. Software development activities, in every phase, are error prone so defects play a crucial role in software development. At the beginning of software testing task we encounter the question: How to inspect the results of executing test and reveal failures? What is risk to finish project within budget, time and reach required software performance i.e. quality? How does one measure test effectiveness, efficacy, benefits, risks (confidence) of project success, availability of resources, budget, time allocated to STP? How does one plan, estimate, predict, control, evaluate and choose "the best" test scenario among hundreds of possible (considered, available, feasible) number of test events (test cases)? IOSTP framework solved these issues combining few engineering and scientific areas such as: Design of Experiments, Modeling & Simulation, integrated practical software measurement, Six Sigma strategy, Earned (Economic) Value Management (EVM) and Risk Management (RM) methodology through simulation-based software testing scenarios at various abstraction levels of the SUT to manage stable (predictable and controllable) software testing process at lowest risk, at an affordable price and time. In order to significantly improve software testing efficiency and effectiveness for the detection and removal of requirements and design defects in our framework of IOSTP, during 3 years of our IOSTP framework deployment to STP we calculated overall value returned on each dollar invested i.e. ROI of 100:1.

Lessons Learned

It is our dream that software engineering will become as much of an engineering discipline as the others; users will have just as much confidence that their software is as defect free as their cars, highway bridges, and aircraft. Test should be used to certify that the software components implement their designs, and that these designs satisfy their requirements. Analyzing testing requirements should be done in parallel with analyzing the software components' requirements. Tests should be designed in parallel with designing the components. Test implementation should occur in parallel with implementing the components, and developing integration tests should be done in parallel with integration. The source of software defects is a lack of discipline in proper requirements analysis, design, and implementation processes. Testing must physically occur after implementation, so reliance on it to detect defects delays their correction. Until software defects are attacked at their source, software will continue to be developed as if it were an art form rather than a craft, engineering discipline, or a science.

Treating software testing as a discipline is a more useful analog than treating it as an art or a craft. We are not artists whose brains are wired at birth to excel in quality assurance. We are not craftsmen who perfect their skill with on-the-job practice. If we are, then it is likely that full mastery of the discipline of software testing will elude us. We may become good, indeed quite good, but still fall short of achieving black belt - status. Mastery of software testing requires discipline and training.

A software testing training regime should promote understanding of fundamentals. I suggest three specific areas of pursuit to guide anyone's training:

First and foremost, master software testers should understand software. What can software do? What external resources does it use to do it? What are its major behaviors? How does it interact with its environment? The answers to these questions have nothing to do with practice and everything to do with training. One could practice for years and not gain such understanding.

Second, master software testers should understand software faults. How do developers create faults? Are some coding practices or programming languages especially prone to certain types of faults? Are certain faults more likely for certain types of software behavior? How do specific faults manifest themselves as failures?

Third, master software testers should understand software failure. How and why does software fail? Are there symptoms of software failure that give us clues to the health of an application? Are some features systemically problematic? How does one drive certain features to failure?

Understanding software, faults and failures is the first step to treating software testing as a discipline. Treating software as a discipline is the first step toward mastering software quality. And there is more, always more to learn. Discipline is a lifelong pursuit. If you trick yourself into thinking you have all the answers, then mastery will elude you. But training builds knowledge so the pursuit itself is worthwhile whether or not you ever reach the summit.

Perhaps we need to embrace Tester Pride and let the world know about the contributions we make.

References:

[1] Boehm B. *Software Risk Management.* IEEE Computer Society Press, Los Alamitos California, 1989.

[2] Burstein I. at all. *Developing a testing maturity model, Part II*. Illinois Institute of Technology, Chicago, 1996.

[3] Cristie A. Simulation: An enabling technology in software engineering. *CrossTalk the Journal of Defense Software Engineering*, April 1999.

[4] http://www.acq.osp.mil. URLs cited were accurate as of April 2002.

[5] http://www.msosa.dmso.mil. URLs cited were accurate as of May 2001.

[6] Lazić, Lj., Velašević, D., Applying simulation to the embedded software testing process, SOFTWARE TESTING, VERIFICATION AND RELIABILITY, Volume 14, Issue 4, 257-282, John Willey & Sons, Ltd., 2004.

[7] Lazić, Lj., Automatic Target Tracking Quality Assesment using Simulation, 8th Symposium on Measurement - JUREMA, 29-31 October, Kupari-Yugoslavia, 1986.

[8] Lazić, Lj., Computer Program Testing in Radar System,. Masters thesis, University of Belgrade, Faculty of Electrical Engineering, Belgrade, Yugoslavia, 1987.

[9] Lazić, Lj., Method for Clutter Mup Alghoritm Assesment in Surveilance Radar, 11th Symposium on Measurement - JUREMA, April, Zagreb-Yugoslavia, 1989.

[10] Lazić, Lj., Software Testing Methodology, YUINFO'96, Brezovica, Serbia&Montenegro, 1996.

[11] Lazić, Lj., Velašević, D., Integrated and optimized software testing process based on modeling, simulation and design of experiment, 8th JISA Conference, Herceg Novi, Serbia&Montenegro, June 9-13, 2003

[12] Lazić, Lj., Velašević, D. i Mastorakis, N., A framework of integrated and optimized software testing process, WSEAS TRANSACTIONS on COMPUTERS, Issue 1, Volume 2, 15-23, January 2003.
[13] Lazić, Lj., Medan, M., SOFTWARE QUALITY ENGINEERING versus SOFTWARE TESTING PROCESS, TELFOR 2003(Communication Forum), 23-26 November, Beograd, 2003.
[14] Lazić, Lj., Velašević, D. i Mastorakis, N., The Oracles-Based Software Testing: problems and solutions", WSEAS MULTICONFERENCE PROGRAM, 3rd WSEAS Int.Conf. on SOFTWARE ENGINEERING, PARALLEL & DISTRIBUTED SYSTEMS (SEPADS 2004), February 13-15, Salzburg, Austria, 2004, also in WSEAS Transactions on Computers, Issue 4, Volume 3, p 1026-1036, October 2004.
[15] Lazić, Lj., Velašević, D. i Mastorakis, N., Software Testing Process Management by Applying Six Sigma, WSEAS Joint Conference program, MATH 2004, IMCCAS 2004, ISA 2004 and SOSM 2004, Miami, Florida, USA, April 21-23, 2004.
[16] Lazić, Lj., Velašević, D., Software Testing Process Improvement by Applying Six Sigma, 9th JISA Conference, Herceg Novi, Serbia & Montenegro, June 9-13, 2004.
[17] Lazić, Lj., Integrated and Opitimized Software Testing Process, TELFOR 2004 (Communication Forum), 23-26 November, Beograd, 2004.
[18] Lazić, Lj. SOFTWARE TESTING versus SOFTWARE MAINTENANCE PROCESS, Simpozijum Infoteh-Jahorina, 23-25 March, 2005.
[19] Lazić, Lj., Mastorakis, N., Software Testing Process Improvement to achieve a high ROI of 100:1, 6th WSEAS Int. Conf. On MATHEMATICS AND COMPUTERS IN BUSINESS AND ECONOMICS (MCBE'05), March 1-3, Buenos Aires, Argentina 2005.
[20] Lazić, Lj., Mastorakis, N., Applying Modeling&Simulation to the Software Testing Process – One Test Oracle solution, 4th WSEAS Conference on Automatic Control, Modeling and Simulation (ACMOS 2005), TELEINFO 2005, and AEE 2005 WSEAS MultiConference, Prague, Czech Republic, March 13-15, 2005
[21] Lazić, Lj., Mastorakis, N., RBOSTP: Risk-based optimization of software testing process Part 1, accepted in WSEAS Proceedings + JOURNAL in the 9th WSEAS International on COMPUTERS Vouliagmeni, Athens, Greece, July 2005
[22] Lazić, Lj., Mastorakis, N., RBOSTP: Risk-based optimization of software testing process Part 2, accepted in WSEAS Proceedings + JOURNAL in the 9th WSEAS International on COMPUTERS Vouliagmeni, Athens, Greece, July 2005
[23] DoD Integrated Product and Process Development Handbook, August 1998
[24] H. Ziv and D.J. Richardson, Constructing Bayesian-network Models of Software Testing and Maintenance Uncertainties, International Conference on Software Maintenance, Bari, Italy, September 1997.
[25] Humphrey, W. S., Making Software Manageable, CrossTalk, December 1996, pp. 3-6.
[26] Baber, R. L., The Spine of Software: Designing Provably Correct Software: Theory and Practice, John Wiley & Sons Ltd., Chichester, United Kingdom, 1987.
[27] Daich, G. T., Emphasizing Software Test Process Improvement, Crosstalk, June 1996, pp. 20-26, and Daich, Gregory T., Letters to the Editor, CrossTalk, September 1996, pp. 2-3, 30.
[28] Burnstein, I.; Suwannasart, T.; and Carlson, C.R., Developing a Testing Maturity Model: Part I, CrossTalk, August 1996, pp. 21-24; Part II, CrossTalk, September 1996, pp. 19-26.
[29] Paulk, M. C.; Curtis, B.; Chrissis, M. B.; and Weber, C. V., Capability Maturity Model^{SM} for Software, Version 1.1, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, February 1993.
[30] Hausler, P. A.; Linger, R. C.; and Trammel, Adopting Cleanroom Software Engineering with a Phased Approach, IBM Systems Journal, volume 33, number 1, 1994, p. 95.
[31] Bernstein, L.; Burke Jr., E. H.; and Bauer, W. F., Simulation- and Modeling-Driven Software Development, CrossTalk, July 1996, pp. 25-27.
[32] Page-Jones, M., What Every Programmer Should Know About Object-Oriented Design, Dorset House Publishing, New York, New York, 1995.
[33] Linger, R.C., Cleanroom Software Engineering: Management Overview, Cleanroom Pamphlet, Software Technology Support Center, Hill Air Force Base, Utah, April 1995.
[34] Capability Maturity Model Integration (CMMI), Version 1.1, Software Engineering Institute, CMU/SEI-2002-TR-011, TR-012, March 2002
[35] Schulmeyer, G. G., Zero Defect Software, McGraw-Hill, Inc., New York, New York, 1990.
[36] Martin, J., System Design from Provably Correct Constructs, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1985.
[37] *C/S Solutions, Inc.*, Risk+ User's Guide, 2002
[38] DoD Defense Acquisition Deskbook -- Compendium of Acquisition-Related mandatory and discretionary guidance, including risk management (http://www.deskbook.osd.mil)
[39] Carleton, Anita D., Park, Robert E., Goethert, Wolfhart B., Florac, Willliam A., Bailey, Elizabeth K., & Pfleeger, Shari Lawarence. Software Measurement for DoD Systems: Recommendations for Initial Core Measures (CMU/SEI-92-TR-19). Software Engineering Institute, Carnegie Mellon University, September 1992.
Available online:
www.sei.cmu.edu/publications/documents/92.reports/92.tr.019.html
[40] Le K., Phongpaibul M., and Boehm B., Value-Based Verification and Validation Guidelines, CSE University Southern California, TR UC-CSE-05-502, February 2005
[41] Hillson D., Combining Earned Value Managment and Risk Management to create synergy, found at www.risk-doctor.com, reached April 2005.
[42] Richardson DJ. TAOS: Testing with analysis and oracle support, Proceedings of the 1994 International Symposium on Software Testing and Analysis (ISSTA '94), Seattle, WA, August 1994; ACM Press, New York; 138-153.
[43] Weyuker EJ. On testing non-testable programs. The Computer Journal 1982; 25(4): 465-470.
[44] Box GEP, Hunter WG, Hunter JS. Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building. Wiley, New York, 1978.
[45] Montgomery DC. Design and Analysis of Experiments. 4th ed., Wiley, New York, 1997.