Reprinted from **Proceedings of the First International Conference on Advances in System Testing and Validation Lifecycle (VALID´09), Porto (Portugal), September 20-25, 2009, pp.80-85, IEEE Computer Society 2009**

Integration Test Order Strategies to Consider Test Focus and Simulation Effort

Lars Borner, Barbara Paech
Chair of Software Engineering,
Institute for Computer Science,
University of Heidelberg
Im Neuenheimer Feld 326,
Heidelberg, Germany
borner@informatik.uni-heidelberg.de
paech@informatik.uni-heidelberg.de

***Abstract* — The integration testing process aims at uncovering faults within dependencies between the components of a software system. Due to the lack of resources, it is usually not possible to test all dependencies. Fault prone dependencies have to be selected as test focus. This test focus has to be considered during the stepwise integration of the whole software system. An integration test order strategy has to devise an integration order that integrates dependencies selected as test focus in early integration steps. Furthermore the strategy has to minimize the effort to simulate not yet integrated components of the software system. Current approaches only focus on the reduction of the simulation effort, but do not take into account the test focus. This paper introduces an approach to determine an optimal integration testing order that considers both, the test focus and the simulation effort. The approach is applied to nine real software systems and the results are compared to six approaches.**

Keywords-integration testing; integration order; test focus, simulation effort, heuristic algorithms

I. INTRODUCTION

Today's software systems consist of thousands of software components. These components depend on each other. Testing the dependencies between components is the goal of the integration testing process, namely to uncover "… *component faults that cause intercomponent failure*" ([1], p 629). A dependency **A→B** is defined as unidirectional relationship between two components **A** and **B**. The dependent component **A** depends on the functionality of the independent component **B**.

It is not possible to test all dependencies within a software system [9]. Therefore, the dependencies to be tested have to be selected carefully. An approach to the selection of the test focus for the integration testing process is described in [2] and [3]. This approach uses information of former versions of a software system to uncover dependency properties that correlate with the number of faults in the participating components. This information is used to predict fault prone dependencies in the current version of the software system. These dependencies are selected as test focus.

Having decided which dependencies should be tested, several further decisions have to be made (see [4]). One of these decisions deals with the stepwise integration test order. A stepwise integration adds one component at a time to a set of already integrated components and tests the dependencies between the already integrated components and the new component. Thus the dependencies are systematically tested and the effort to uncover the cause of a fault is reduced [1]. The disadvantage is that components required in the current integration step, but not yet integrated, have to be simulated by using stubs [6].

An optimal integration test order has to satisfy two criteria: First, the dependencies selected as test focus have to be integrated as early as possible. A dependency is integrated, if and only if the dependent and independent component of the dependency is integrated. Second, the selected integration test order should utilize a minimal simulation effort. The simulation effort is the effort required to simulate non-integrated components. Current approaches like [5], [6], [8], and [12] only consider the last criterion and try to minimize the simulation effort. No existing approach can be found that considers the test focus and the simulation effort in the integration test order determination.

In Section 2 we introduce an approach to measure the test focus consideration of given integration test orders. Section 3 describes five algorithms that are used in existing approaches to derive an integration test order. Furthermore the Simulated Annealing algorithm [7] is described and used for the first time to derive an integration test order. Additionally an approach is proposed that derives an order that perfectly considers the test focus, but does not take into account the simulation effort. In a first experiment these seven approaches are applied to nine open source software systems to show how

well they consider the test focus and the simulation effort. The results of the first experiment are given in Section 3. Section 4 describes possible adaptations of existing algorithms with the goal to determine an integration test order that optimizes both, the test focus and the simulation effort. In a second experiment the adapted approaches are applied to the nine software systems and the results of the experiment are presented in Section 4. Section 5 summarizes the results of the paper.

II. MEASURING THE TEST FOCUS CONSIDERATION

For a given integration test order the test focus consideration has to be determined. The goal is to compare two orders to decide which one considers the test focus best. To satisfy the criterion of test focus consideration, all components involved in dependencies, which are selected as test focus, have to be integrated before components involved in not selected dependencies are integrated. Therefore, our approach divides the components to be integrated into two sets. The first set $\mathbf{C_{TF}}$ contains all components involved in dependencies selected as test focus. The second set $\mathbf{C_{NTF}}$ contains the remaining components. It is important to mention that both sets are disjoint. If a component A can be assigned to both sets, because A participates in dependencies selected as test focus and dependencies not selected as test focus, A is assigned to the set C_{TF}., because A is required to integrate the test focus dependency.

The test focus consideration TFC for a given integration test order ITO is calculated as follows:

TFC(ITO) = Number of correctly integrated dependencies

Concerning the test focus, a dependency is integrated correctly, if the dependent and independent components of the dependency are integrated in the first *X-1* steps of the given ITO, where X is the number of components in C_{TF}: i.e. $\mathbf{X = |C_{TF}|}$. To consider the test focus perfectly, all components in C_{TF} have to be integrated before the components in C_{NTF}. In such a case the test focus consideration is the number of dependencies selected as test focus.

This approach is illustrated by the example system described in [6] and shown in Figure 1. The system is modeled by using an Object Relation Diagram [6] and consists of eight components and 17 dependencies. The dependencies are labeled according to the kind of dependency, where *As* stands for Association, *Ag* for Aggregation and *I* for Inheritance.

Within the example system the dependencies **E→A**, **E→F**, **C→H** und **H→C** where randomly selected as test focus. Therefore, the components in C_{TF} are **A**, **C**, **E**, **F** and **H** and should be integrated in the first 4 (=$|C_{TF}|$-1) integration steps.

The order **ACGHEDBF** is one possible ITO. In a first step **A** and **C** are integrated, in the second step **G,** in the third step **H**, in the fourth **E** and so on. For this order the three dependencies **E→A**, **C→H**, and **H→C** are correctly integrated, because the components **A**, **C** and **H** are integrated in the first four integration steps. The component **E** is integrated in the fourth step but **F** is integrated in the seventh step, therefore the dependency **E→F** is integrated wrongly. This means 75% of the test focus dependencies are integrated correctly. An order not considering the test focus very well is **GBDFEACH**. Only the dependency **E→F** is integrated correctly because the components **E** and **F** are integrated in the first 4 steps. These are 25% of all dependencies.

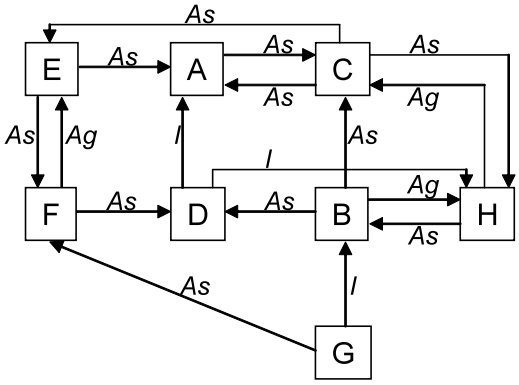

Figure 1. Example System [6]

Our approach can be used to compare existing algorithms. For every existing algorithm the derived ITO is analyzed to clarify how many dependencies are integrated correctly concerning the test focus.

III. EXISTING APPROACHES

Several approaches can be found in the literature, which derive an optimal integration test order. All approaches only focus on minimizing the simulation effort that means they try to minimize the number of components, dependencies, service calls and/or attribute accesses to be stubbed. In an acyclic software system this could be done by simply using the topological sort algorithm [6]. If the software system contains dependency cycles, "*the proposed strategy consists of identifying strongly connected components (SCCs) and removing associations until there is no cycle in the SCCs*" ([6], page 594). Removing an association, or in general a dependency, from a given software system, leads to a new stub that has to be realized.

The algorithms applied in this experiment can be divided into two categories: graph-based algorithms and heuristic algorithms. Algorithms of the first category systematically try to remove dependencies. They use different approaches to select the dependencies to be removed. In our experiments we applied the algorithm of Tai and Daniels, Le Traon et al. and Briand et al. summarized in [6]. The heuristic approaches systematically create random integration test orders. They use a cost function to compare two orders to identify the best. Through several iterations new orders are created and after a finite number of iterations the best order is selected. In the literature the only heuristic algorithm that is applied to derive an integration test order is the Genetic algorithm

[5]. In our work we also analyzed a random based algorithm and an approach called Simulated Annealing [7]. As far as we know this is the first research work that uses the Simulated Annealing to derive an integration test order.

In the following sections both categories and algorithms shortly are introduced. Afterwards an algorithm that derives an integration order that perfectly considers the test focus is proposed.

A. Graph Based Algorithms

Graph based algorithms use as input to derive an integration test order a model such as an Object Relation Diagram shown in Figure 1. They identify cycles in the model and dependencies to be removed in order to break the cycles. One of these approaches is the algorithm of Tai and Daniels [12]. In a first step they only consider inheritance and aggregation dependencies to assign Major level numbers to each component. In a second step the associations are analyzed to compute additional Minor level numbers. The Major and Minor level numbers are used to determine the integration order.

The algorithms of Le Traon et al. [8] and Briand et al. [6] use the algorithm of Robert Tarjan [13] to identify SCCs. For every SCC the selected dependencies are removed to break the SCC. The approaches differ according to the way the dependencies, which are to be removed, are selected. Le Traon et al. compute a weight for every component and remove all incoming dependencies of a component with the highest weight. Briand et al. compute a weight for the association in an SCC. The association with the highest weight is removed. After removing one or more dependencies, Le Traon et al. and Briand et al. apply Tarjan's algorithm again to identify further SCCs and select new dependencies to be removed. This is repeated until Tarjan's algorithm does no longer find any cycles in the remaining graph. Finally the integration test order can be determined by simply using the topological sort algorithm.

B. Heuristic Algorithms

Heuristic algorithms do not use the model of the software system to identify dependencies to be removed in order to break cycles. They create several random integration test orders. They use a cost function to compare the orders to identify the best order.

The only heuristic approach used to derive integration test orders is the Genetic algorithm as described by Briand et al. in [5]. Genetic algorithms associate optimization problems with biological concepts. An integration test order can be seen as a chromosome consisting of genes. The genes are the components of the software system. Every gene in a chromosome has a specific position. One chromosome is part of a population. A population is a set of chromosomes. By modifications like crossover and mutation the chromosomes of a given population can be transformed. The new chromosomes are part of a new population. However, only the fittest chromosomes are modified to get better chromosomes. The goal of the Genetic algorithm is to let the fittest chromosomes "survive" and bad chromosomes "die". After a finite number of iterations, where new populations were created, the best chromosome is selected as the test focus.

When using heuristic algorithms the challenge is to define a cost function that determines a numeric value that indicates the fitness of a chromosome. Briand et al. propose in [5] a cost function for an integration test order. They use the number of attribute accesses and the number of service calls that have to be stubbed by a given integration order. Both values are normalized that means for every dependency the number of attribute accesses in this dependency is divided by the maximum number of attribute accesses in all dependencies. The service calls are normalized in the same way. To make sure that no inheritance dependencies are broken, Briand et al. use a precedence table to avoid chromosomes that break inheritances. Therefore, they do not need to consider the inheritances in their cost function. In our experiments we do not use a precedence table. We adapted the cost function of Briand et al. and added a new parameter to consider the inheritances. The new parameter is set to "1" if the dependency is an inheritance and is set to "0" if not. As shown in the following equations, we use the weight means to compute the simulation effort (SE) of a broken dependency A→B. A_{Norm} is the normalized number of attribute accesses, S_{Norm} the normalized number of service calls and I is the inheritance parameter. The weights W_A, W_S and W_I are used to adapt the equations to the project context. They can be used to specify that simulating an inheritance is harder than breaking service calls and attribute accesses. The sum of all three weights has to be "1". To make sure that as few inheritances as possible are broken by the derived integration test order, we use the values W_I=0.9 and W_A=W_S=0.05. To compute the fitness of a given integration test order, the simulation effort SE for all broken dependencies has to be sum up.

$$\mathbf{SE(AB)=(W_A*A_{Norm}(AB)+W_S*S_{Norm}(AB)+W_I*I(AB))*1/3}$$

A simpler heuristic approach to derive an integration test order is the random based approach. This approach tries *X*-times to create a random integration order without taking into account the old orders. Every time a new order is randomly generated, the order is compared to the current best order by using the cost function above. If the fitness of the new one is smaller than the fitness of the current best, the new one becomes the current best.

Another heuristic approach that can be used to derive integration test orders is the Simulated Annealing algorithm. Burkard and Rendl describe in [7] how this algorithm can be applied to optimization problems. In our work we adapt this algorithm to be used in integration test order derivation. The Simulated Annealing algorithm is inspired by physical processes. The algorithm "… *was proposed […] to simulate a collection of atoms in equilibrium at given temperature t*" ([7], p. 170). The equilibrium is described by the energy E that the collection shows. Position changes of the atoms in the collection lead to changes in the energy. Adapted to the integration test

order derivation the collection of atoms represents an order. The energy can be expressed by the fitness of the order. By changing the position of two components the fitness of the order will change. If the new order fits better, it will be accepted. If this is not the case the new order is accepted with a given probability. The probability enables the algorithm to pass a local optimum to find the global optimum. The probability P depends on the current temperature t and the difference between the fitness of the old and the new order as shown in the following equation:

$$\mathbf{P = exp\ (\ (\ F(IR_{new}) - F(IR_{old})\ )\ /\ t)}$$

At the beginning the temperature is high and thus the probability to accept worse orders will also be high. This means that at the beginning worse orders are accepted. During the execution of the algorithm the temperature decreases and as a consequence the number of accepted worse orders will become very small. For the Simulated Annealing algorithm three parameters have to be set before it can be executed: initial temperature $t_{initial}$, final temperature t_{final} and the number of modifications with the same temperature *N*. The initial temperature is the temperature the algorithm starts with. The higher the initial temperature the higher is the probability to accept worse orders. The final temperature t_{final} is the exit condition of the algorithm. If the current temperature is smaller than t_{final} the algorithm stops. *N* describes the number of modifications of current orders with the same temperature. Every time the temperature decreases *N* tries are made to create new orders.

C. Ideal Test Focus Consideration (ITFC)

All six algorithms of the previous sections only take into account the simulation effort when deriving an integration test order. To derive an order that considers the test focus perfectly, the two sets $\mathbf{C_{TF}}$ and $\mathbf{C_{NTF}}$ have to be integrated one after another starting with the components in $\mathbf{C_{TF}}$ followed by the components in $\mathbf{C_{NTF}}$. The order within both sets does not make a difference for the test focus consideration. Existing approaches can be used to optimize the integration test order within both sets. In our approach we use the graph-based algorithm of Briand et al., [6] because our experiment results have shown that it is the best algorithm to minimize the simulation effort. This algorithm is applied to both sets independently. After the application of Briand's algorithm both ordered sets are combined to one integration test order. The derived order perfectly satisfies the criterion of test focus consideration, but breaks at least all dependencies where the independent component is assigned to the set $\mathbf{C_{NTF}}$ and the dependent component to $\mathbf{C_{TF}}$.

D. First Experiment

In a first experiment the seven algorithms are applied to nine open source software systems. The software systems[1] are *Eclipse (1)*, *Apache ANT (2)*, *Apache FOP (3)*, *Chemistry Development Kit (4)*, *Free Network Project (5)*, *Jetspeed (6)*, *JMol (7)*, *OSCache(8)* and *TVBrowser (9)*. The goal is to identify how well the algorithms consider the test focus and the simulation effort. To measure the test focus consideration we use the number of *wrongly integrated dependencies*. For the simulation effort we measure the number of *stubbed classes*, *stubbed dependencies*, *broken inheritances*, *stubbed service calls*, *stubbed attribute accesses*, and the *fitness*. Every algorithm was only applied once to the nine software systems. An overview of the size of all software systems is shown in Table I.

The dependencies of the software systems and their properties are identified by using the source code analyzer SISSy [2] and a small self developed tool called MetricAnalyzer. SISSy creates an abstract model of the source code and exports the model into a data base. The MetricAnalyzer extracts the dependencies and their properties from the data base. The required dependency properties are the *number of service calls*, the *number of attribute accesses* and a flag indicating an *inheritance*. The analyzed components are source code files. The mapping from classes to source code files is described in [3].

For Eclipse the test focus has already been computed by the test focus selection approach described in [3]. The results of the case study in [3] are used in this experiment. For the remaining eight software systems we compute the test focus by using the number of faults per component. This number was computed by Timea Illes-Seifert in her work in [10] and [11]. The dependencies where 20% of components with the highest number of faults are involved are selected as test focus.

TABLE I. SIZE OF THE SOFTWARE SYSTEMS

	Source Code Files	Dependencies	Inheritances	Associations
OSCache	110	298	43	282
JMol	323	1480	172	1455
Freenet	456	2019	262	1852
TVBrowser	818	4320	393	4053
Apache FOP	1006	4728	773	4468
Apache CDK	1022	5745	751	5289
ANT	1053	4524	1027	4236
Jetspeed	1347	4315	991	3733
Eclipse	10133	96476	10375	91455

For the comparison of each algorithm and each metric an average rank for all nine software systems is computed. For example the graph-based algorithm of Briand et al. computed for five software systems the order with the smallest number of stubbed components (rank 1) and for four software systems it reaches the second rank. The

[1] (1) Eclipse, www.eclipse.org, (2) http://ant.apache.org, (3) http://xmlgraphics.apache.org/fop/, (4) http://sourceforge.net/projects/cdk/, (5) http://freenetproject.org, (6) http://portals.apache.org/jetspeed-2/, (7) http://jmol.sourceforge.net/, (8) http://www.opensymphony.com/oscache/, (9) http://www.tvbrowser.org/

[2] http://sissy.fzi.de/

average rank for all software systems is 1.4. The average rank for each algorithm and metric is summarized in Table II. The average rank of the best algorithm according to a metric is highlighted.

TABLE II. AVERAGE RANK PER ALGORITHM (FIRST EXPERIMENT)

	Fitness	**Stubbed Files**	**Stubbed Dependencies**	**Stubbed Service Calls**	**Stubbed Attribute Accesses**	**Broken Inheritances**	**Wrongly Integrated Dependencies**
ITFC	5.9	5.2	5.0	4.9	5.2	5.9	**1.0**
Tai & Daniels	4.2	4.6	5.4	5.3	4.2	**1.0**	5.4
Simulated Annealing	1.9	2.8	2.6	2.0	2.4	1.8	4.1
Genetic	3.2	4.9	4.1	3.7	3.6	2.1	4.4
Briand	**1.4**	**1.4**	**1.0**	**1.6**	**1.4**	**1.0**	6.4
Le Traon	4.3	1.9	2.9	3.6	4.0	4.4	2.3
Random-Based	7.0	7.0	7.0	7.0	7.0	7.0	3.8

The last column of the table shows that our algorithm considers the test focus best. However, our algorithm does not very well take into account the simulation effort. This is shown by the average rank of the fitness, of the stubbed elements (Files, Dependencies …) and of the broken inheritances. As shown in Table III our algorithm considers 100% of the test focus dependencies. The second best algorithm is the algorithm of Le Traon et al. However, this algorithm integrates only about 34% of the test focus dependencies correctly. The algorithm that considers the test focus worst is the algorithm of Briand et al. Only about 10% of all test focus dependencies were integrated correctly.

TABLE III. CORRECTLY INTEGRATED TEST FOCUS IN PERCENT (FIRST EXPERIMENT)

	ITFC	**Tai & Daniels**	**Simulated Annealing**	**Genetic**	**Briand**	**Le Traon**	**Random-based**
Correctly integrated test focus in %	100	13.26	19.27	13.48	10.13	33.78	20.62

Yet, the algorithm of Briand et al. performs best in minimizing the simulation effort (average rank of 1.0 to 1.6). Our results are consistent with the results of Briand et al. in [6], who compared the three graph based algorithms mentioned above to analyze how well they consider the simulation effort.

IV. TEST FOCUS AND SIMULATION EFFORT

In this section we analyze whether the existing algorithms can be adapted to consider the test focus as well as the simulation effort. The graph based algorithms cannot be adapted to consider the test focus as they do not have parameters which can be applied to consider additional optimization criteria. Only the heuristic approaches are extendable, because their cost function describes whether an order fits better than another. All we have to do is to expand the cost function by an additional value. This value represents the test focus consideration **TFC**. It is added to the overall simulation effort **OSE** of an integration order. The goal is to find a value that represents the test focus consideration and is equivalent to the value of the overall simulation effort. In several experiments we tested equations for TFC. First we used the number of not correctly integrated dependencies. However, the influence of the TFC to the fitness of an order was too high. A better value representing the TFC in our cost function is the number of not correctly integrated dependencies divided by the average number of dependencies per component.

The fitness of an integration test order is computed by the weighted arithmetic mean of **OSE** and **TFC**:

$$\textbf{Fitness} = (W_{OSE} * \textbf{OSE} + W_{TFC} * \textbf{TFC}) / 2$$

The sum of W_{OSE} and W_{TFC} has to be "1". Both weights can be used to parameterize the algorithms. If an integration order is required that considers the simulation effort in the same way as the test focus, both weights have to have the same value. In our second experiment we use $W_{OSE}=W_{TFC}=0.5$.

A. Second Experiment

In our second experiment we apply the adapted versions of the heuristic approaches to the nine software systems. Similar to the first experiment the approaches are only applied once. The results of the other four algorithms are taken from the first experiment. The goal is to show that the test focus as well as the simulation effort can be considered in an integration order. The results are shown in Table IV and Table V. For the

Beside the seven metrics of the first experiment two additional metrics are collected. The metric **TFC** (column 3) represents the test focus consideration and **OSE** (column 4) the overall simulation effort. The average rank of the fitness computed by the cost function is represented in column 2 and indicates how well the derived integration test orders consider the test focus and the simulation effort. As expected, the ITFC algorithm reaches the best **TFC** in all nine software systems, but the average rank of OSE is 5.9. The algorithm of Briand et al. reaches the best **OSE** in all nine software systems, but the average rank of **TFC** is 6.7. The algorithms considering the simulation effort and the test focus are the two heuristic approaches "Simulated Annealing" and "Genetic". These two algorithms compute orders with the best fitness (average rank 1.4 respectively 2.3, Table IV). In the test focus consideration only the ITFC algorithm is better than these two algorithms which reach an average rank of 2.1 respectively 2.9. As can be seen in Table V the Simulated Annealing algorithm integrates about 88% of all test focus dependencies correctly, the Genetic algorithm about 67%. In minimizing the simulation effort only the algorithm of Briand et al. is

better than the Simulation Annealing algorithm (average rank 2.6). The Genetic algorithm reaches an average rank of 4.0 after the algorithm of Tai and Daniels (average rank 3.4) in minimizing the **TFC**.

TABLE IV. AVERAGE RANK PER ALGORITHM (SECOND EXPERIMENT)

	Fitness	**TFC**	**OSE**	**Stubbed Files**	**Stubbed Dependencies**	**Stubbed Service Calls**	**Stubbed Attribute Accesses**	**Broken Inheritances**
ITFC	3.4	**1.0**	5.9	4.8	4.7	4.7	4.9	5.9
Tai&Daniels	5.4	6.2	3.4	3.8	4.7	5.0	3.6	**1.0**
Simulated Annealing	**1.4**	2.1	2.6	3.8	3.2	2.8	2.8	2.1
Genetic	2.3	2.9	4.0	5.7	5.1	4.9	4.8	3.3
Briand	4.0	6.7	**1.0**	**1.4**	**1.0**	**1.0**	**1.2**	**1.0**
LeTraon	4.3	4.3	4.1	1.6	2.3	2.8	3.7	4.3
Random	7.0	4.7	7.0	7.0	7.0	6.9	7.0	7.0

The second experiment has shown that an integration test order can consider the test focus and minimize the simulation effort. The best algorithm in considering both is Simulated Annealing followed by the Genetic algorithm. The disadvantage of the heuristic algorithms is their long duration to compute an integration test order. In all nine software systems they took the longest time to compute the integration test order.

TABLE V. CORRECTLY INTEGRATED TEST FOCUS (SECOND EXPERIMENT)

	ITFC	**Tai & Daniels**	**Simulated Annealing**	**Genetic**	**Briand**	**Le Traon**	**Random-based**
Correctly integrated test focus in %	100	13.26	88.07	66.69	10.13	33.78	27.56

The results of our experiments show that it is very important to select the algorithm that fits best to the current project context. If the aim is to derive an integration test order that minimizes the simulation effort, the algorithm of Briand et al. should be used. If an integration order is required that integrates the dependencies selected as test focus, the ITFC algorithms should be used. Both algorithms are very fast and derive an order that fits the corresponding criterion. If one is interested in an order that considers both, the heuristic algorithms are the only choice.

V. SUMMARY

In this paper we presented an approach to measure the test focus consideration of a given integration test order. Within a first experiment we compared several algorithms how well they consider the test focus and the simulation effort. Afterwards we adapted the three heuristic approaches to better consider the test focus and the simulation effort and applied them again to software systems. The results show that the heuristic approaches Simulated Annealing and Genetic algorithm can be used to derive integration test orders that optimize the test focus as well a the simulation effort.

The disadvantage of both heuristic approaches is the long duration for deriving an order. In our current work, we analyze how the duration can be reduced by using better starting orders instead of random starting orders. In further experiments we will try to use orders computed by the ITFC algorithm or the algorithm of Briand et al. as starting orders and reduce the number of iterations.

REFERENCES

[1] R. Binder, Testing Object-Oriented Systems". Addison-Wesley, 2000

[2] L. Borner, and B. Paech, "Testfokusauswahl im Integrationstestprozess" in Liggesmeyer P, Engels G, Münch J, Dörr J, Riegel N (Ed.): Software Engineering (SE 2009) Fachtagung des GI-Fachbereichs Softwaretechnik in Kaiserslautern, LNI P-143, pp. 139-150, GI 2009

[3] L. Borner, B. and Paech, "Using Dependency Information to Select the Test Focus within the Integration Testing Process", to appear at Taic Part 2009

[4] L. Borner, T. Illes, and B. Paech, "The Testing Process - A Decision Based Approach" In: Proceedings of The Second International Conference on Software Engineering Advances (ICSEA 2007), Cap Esterel (France), p. 41, IEEE Computer Society 2007

[5] L.C Briand, J. Feng, and Y. Labiche, "Using Genetic Algorithms and Coupling Measures to Devise Optimal Integration Test Orders", 14th International Conference on Software Engineering and Knowledge Engineering (SEKE 2002), IEEE Computer Society, 2002

[6] L.C. Briand, Y. Labiche, and Y. Wang, "An Investigation of Graph-Based Class Integration Test Order", IEEE Transactions on Software Engineering, IEEE Press 2003

[7] R. E. Burkard, and F. Rendl, "A thermodynamically motivated simulation procedure for combinatorial optimization problems" European Journal of Operational Research, Volume 17, Issue 2, August 1984, Pages 169-174

[8] Y. Le Traon, T. Jéron, J.M. Jézéquel, and P. Morel, "Efficient Object.Oriented Integration and Regression Testing", IEEE Transactions on Reliability, IEEE Computer Society 2000

[9] G.J. Meyers, "The Art of Software Testing", John Wiley & Sons, New York, 1979

[10] T. Illes-Seifert, and B. Paech, "Exploring the relationship of a file's history and its fault-proneness: An empirical study " In: Proceedings of the Testing: Academic & Industrial Conference - Practice and Research Techniques (TAIC-PART), Windsor (UK), August 29-31, 2008, pp. 13-22, IEEE Computer Society 2008

[11] T. Illes-Seifert, and B. Paech, "Exploring the relationship of history characteristics and defect count: an empirical study" In: Devanbu P T, Murphy B, Nagappan N, Zimmermann T (Hrsg): Proceedings of the 2008 Workshop on Defects in Large Software Systems (DEFECTS 2008), held in conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2008), Seattle, (USA), July 20, 2008, pp. 11-15, ACM 2008

[12] K. Tai, and F. Daniels, "Test Order for Inter-Class Integrations Testing of Object-Oriented Software", 21st International Computer

Software and Applications Conference (COMSAC 1997), Rio de Janeiro (Brazil), ACM 1997

[13] R. Tarjan, “Depth-First Search and Linear Graph Algorithms”, SIAM J. Comput. Volume 1, Issue 2, pp. 146-160. 1972