

Risk Assessment and Adaptive Group Testing of Semantic Web Services

Xiaoying Bai · Ron S. Kenett · Wei YU

Received: date / Accepted: date

Abstract Testing is necessary to ensure the quality of web services that are loosely coupled, dynamic bound and integrated through standard protocols. Exhaustive testing of web services is usually impossible due to the unavailable source code, diversified user requirements and the large number of possible service combinations delivered by the open platform. The paper proposes a risk-based approach for selecting and prioritizing test cases to test service-based systems. We specially address the problem in the context of semantic web services. Semantic web services introduce semantics to service integration and interoperation using ontology models and specifications like OWL-S. It is conceived to be the future in WWW evolution. However, observations show that semantic errors are more difficult to detect compared to the syntactic errors due to the complex ontology relationships. In this research, we analyze the semantics from various perspectives such as ontology dependency, ontology usage and service workflow to identify the factors that contribute to the risks of the services. Risks are analyzed from two aspects (failure probability and importance) at three layers (ontology data, service and composite service). With this approach, test cases are associated to the semantic features and scheduled based on the risks of their target features. Risk assessment is used to control the process of Web Services progressive group testing, including test case ranking, test case selection and service ruling out. The paper also discusses the key techniques to enable the adaptation mechanism: adaptive measurement and adaptation rules. As a statistical testing technique, the proposed approach aims to detect, as early as possible, the problems with highest impact on the users.

Keywords Semantic Web Services · Risk-Based Testing · Adaptive Testing · progressive group testing

1 Introduction

Service Oriented Architecture (SOA) and its implementation Web Services (WS) introduce an open architecture for integrating heterogeneous software through standard internet protocols [16] [33]. From the provider perspective, proprietary in-house components are encapsulated into standard programmable interfaces and delivered as reusable services for public access and rent. From the consumer perspective, applications are built following a model-driven approach where business processes are translated into control flow and data flow of which the constituent functions can be automatically bound to existing services discovered by service brokers. In this way, it enables large scale software reuse of Internet available

Xiaoying Bai
State Key Laboratory of Software Development Environment, Beijing University of Aeronautics and Astronautics, China
Department of Computer Science and Technology, Tsinghua University, China
Tel.: 86-10-62794935
E-mail: baixy@tsinghua.edu.cn

Ron S. Kenett
KPA Ltd., Israel
Department of Statistics and Applied Mathematics, Universita di Torino, Italy
Tel.: +972-522-434-491
E-mail: ron@kpa.co.il

Wei Yu
State Key Laboratory of Software Development Environment,
Department of Computer Science and Technology, Beijing University of Aeronautics and Astronautics, China
E-mail: yu_wei@ss.buaa.edu.cn

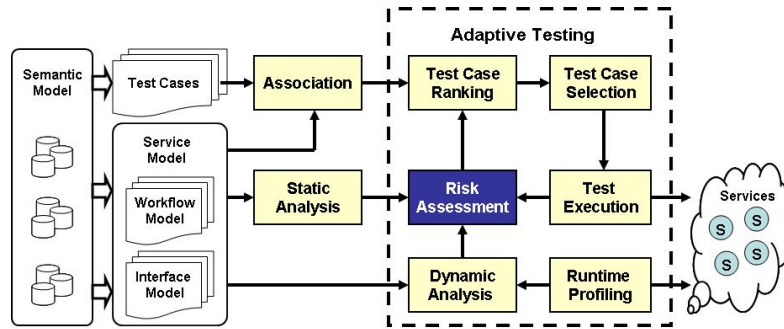


Fig. 1 Approach overview

resources to support agile and fast response to dynamic changing business requirements. SOA is believed to be the major trend of software paradigm shift.

Due to the potential instability, unreliability, and unpredictability of the open environment in SOA and WS, they present challenging quality issues compared with traditional software. Traditional software is built and maintained within a trusted organization. In contrast, service-based software is characterized by dynamic discovery and composition of loose-coupled services that are published by independent providers. A system can be constructed, on-the-fly, by integrating reusable services through standard protocols [30]. For example, a housing map application can be the integration of two independent services: Google Map service [48] and housing rental services [49]. In many cases, the constituent data and functional services of a composite application are out of the control of the application builder. As a consequence, service-based software has a higher probability to fail compared with software developed in-house. On the other hand, as services are open to all Internet users, the provider may not envision all the usage scenarios and track the usage status at runtime. Hence, a failure in the service may affect a wide range of consumers and result in unpredictable consequence. For example, Gmail reported a failure of "service unavailable due to outage in contacts system" on November 8th, 2008 for 1.5 hours, millions of customers were affected.

Testing is thus important to ensure the functionality and quality of the individual services as well as the integrated services, in order to ensure that the selected services can best satisfy the users' needs and that services, that are dynamically composed in to the processes, can interoperate with each other. However, testing is usually expensive and confidence in a specific service is hard to achieve, especially in the open Internet environment. The users may have multiple dimensions of expected features, properties and functional points that result in a large number of test cases. It is both time and resource consuming to test a large set of test cases on the large number of service candidates.

To overcome these issues, the concept of group testing was introduced to services testing [3][4][41][42]. With this approach, test cases are categorized into groups and exercised group-by-group. In each group-testing stage, the failed services are eliminated through a pre-defined ruling out strategies. With effective test case ranking and service ruling out strategies, a large number of services can be removed at the early stage of testing, and thus reduce the total number of executed tests. Test cases ranking and selection is the key problem in progressive group testing to enhance test efficiency.

This paper extends our work in previous research and proposes a risk-based approach to quantitatively evaluate services and rank test cases. Risk-based testing was proposed to select and schedule test cases based on software risk analysis [1][2][13][22] [29][31][32] [36]. With this approach, test cases can be prioritized and scheduled based on the risk of their target software features. When time and resources are limited, test engineers can select a subset of test cases with the highest risk in order to achieve good enough quality with affordable testing effort. This approach has similarity with the studies of Taleb on risks of rare events ("Black Swans") in the financial industry [37][38].

Figure 1 presents an overview of the proposed approach. As shown in the Figure, the service-based system is specified from three perspectives: the interface model of the exposed functionalities of individual or composite service, the workflow model of service composition logic, and the semantic model to define the concepts and domain knowledge for service mutual understanding and interoperation. Particularly, the paper addresses the problem of risk-based test ranking and selection in the context of semantic WS. Risks are assessed based on the semantic model of service data, interface operation, and workflow. The risks of the services are identified in two ways, static and dynamic analysis. Static analysis is based

on service dependence and usage topologies. As services may re-composed online, dynamic analysis is introduced to detect the changes at runtime and re-calculate the service risks based on runtime profiling of service composition, configuration, and operation. Test cases are associated to their target features under test and are ranked based on the risks of the services.

The rest of the paper is organized as follows. Section 2 briefly introduces the research background of risk-based testing, progressive group testing, and semantic WS. Section 3 discusses the problem of adaptive WS testing. Section 4 analyzes the characteristics of semantic WS and proposed the method for estimate and predicate failure probability and importance. Section 5 discusses the methods of adaptive measurement and adaptation rules. These techniques are the basis for incorporating the dynamic mechanism into the WS group testing schema. Section 6 presents the metrics and the experiments to evaluate the proposed approach. Section 7 surveys the related work. Section 8 concludes the research.

2 Background

This research is related to risk-based testing, service progressive group testing, and semantic web services.

2.1 Risk-Based Testing

Testing is expensive, especially for today's software with its growing size and complexities. Exhaustive testing is not feasible due the limitations in time and resources. A key test strategy is to improve test efficiency by selecting and planning a subset of tests with a high probability to find defects. However, selective testing usually faces difficulties in answering the questions like "What should we test first given our limited resources?" and "When can we stop testing?".

Software risk assessment identifies the most demanding and important software aspects, and provides the basis for test selection and prioritization. In general, the risk of a software feature is defined by two factors: the probability to fail, and the consequence of the failure. That is,

$$Risk(f) = P(f) * C(f) \quad (1)$$

Where f is software feature, $Risk(f)$ is its risk exposure, $P(f)$ is the failure probability and $C(f)$ is the cost of the failure. Intuitively, a software feature is risky if it has a high probability to fail or its failure will result in serious consequence. Intuitively, a risky feature deserves more testing effort and has a high priority to test. Risk-based testing was proposed to select and schedule test cases based on software risk analysis [1][2][13][22][29][31][32][36]. The general process of risk-based testing is as follows:

1. Identify the risk indicators;
2. Evaluate and measure the failure probability of software features;
3. Evaluate and measure the failure consequence of software features;
4. Associate the test cases to their target software features;
5. Rank the test cases based on the risks of their target features. Test cases with risky features should be ranked for exercised earlier; and
6. Risk-related coverage can be defined to control the testing process and test exit criteria.

Practices and case studies show that testing can benefit from the risk-based approach in two ways:

1. The reduced resource consumption; and
2. Improved quality by spending more time on critical functions.

2.2 Web Services Progressive Group Testing

Group testing technique was originally developed at Bell Laboratories for efficiently inspecting products [34]. The approach was further expanded to general cases [14][44]. It is routinely applied in testing large number of blood samples to speed up the test and reduce the cost [15]. In this case, a negative test result of the group under test indicates that all the people in the group do not have the disease; otherwise, at least one of them is affected. Group testing has been used in many areas such as medical, chemical and electrical testing, coding, etc., using either combinational or probabilistic mathematical models.

Tsai et al [41] introduces the ideas of progressive group testing to Web Services testing. We define a test potency as its capability to find bugs or defects. Test cases are ranked and organized hierarchically according to their potency to detect defects, from low potency to high. Test cases are exercised layer-by-layer, following a hierarchical structure, at groups of services. Ruling out strategies are defined so that web services that fail at one layer cannot enter the next testing layer. Test cases with a high potency is exercised first with the purpose to remove as many as web services as early as possible. Group testing is by nature a selective-testing strategy, which is beneficial in terms of reduced number of test runs and shortened test time.

2.3 Semantic Web Services

Sir Tim Berners-Lee brought the vision of the semantic Web as a new form of Web content in which the semantics of information and services are defined and understandable by the computers [9]. Ontology techniques are widely used to provide a unified conceptual model of Web semantics [17][35]. In WS, OWL-S provides a semantic model for composite services based on the OWL ontology specification language. OWL-S specifies the intended system behavior in terms of inputs, outputs, process, pre-/post-conditions and constraints using the ontology specifications of data and services [27][45][46]. Spies [35] and the MUSING project [47] introduce a comprehensive approach of ontology engineering to financial and operational risks.

However, semantics introduce additional risks to WS. First, the ontologies may be defined, used, and maintained by different parties. A service may define the inputs/outputs of its interface functions as instances of ontology classes in a domain model that is out of the control of the service provider and consumer. Due to the complexity of conceptual uniformity, it is hard to ensure the completeness and consistency, and the unified quality of the ontologies. For example, AAWS (Amazon Associate Web Services) is an open service platform for online shopping. Its WSDL service interface provides 19 operations with complicated data structure definition. In this research, we translated the WSDL data definition to ontology definition for semantic-based service analysis. We identified 514 classes (including ontology and property classes) and 1096 dependency relationships between the classes.

Moreover, ontologies introduce complex relationships among the data and services which result in the increased possibility of misuse of the ontology classes. For example, in the *BookStore* domain, two ontology classes "CourseBook" and "EditBook" are defined with inheritance from the "Book" ontology. From the domain modeler perspective, the two categories of books are used for different purposes and mutual exclusive. However, from the user perspective, such as a courseware application builder, the two classes could be overlapped because an edited book can also be used as reading materials for graduate students. Such conflicting views will result in a possible misuse of the class when developing education software systems.

In our previous research, we analyzed the problem of ontology misusing and investigated the problem in mutation testing [6]. This paper discusses the issues from another perspective, ontology-based risk analysis.

3 Problem Statement

Given a set of services $S = \{s_i\}$ and a set of test cases $T = \{t_i\}$, selective testing is the process of finding an ordered set of test cases to detect bugs as early as, and as many as possible. However, testing and bug detection are like a "moving target" shooting problem. As testing progresses, bugs are continuously detected and removed. As a result, the bug distribution and service quality are changed. On the other hand, the bug detection potency of each test case is also changed, as each test case may be effective in identifying different types of bugs. Adaptive testing is the mechanism to calculate the ranking and ordering of test cases, at runtime, so as to reflect the dynamic changes in the services, test cases potencies and bugs.

Suppose that $\forall s \in S, B(s) = \{b_i\}$ is the set of bugs in the service, $T(s) \subseteq T$ is the set of test cases for the service s , $\forall b \in B(s), \exists T(b) \subset T(s)$ so that $\forall t_i \in T(b), t_i$ can detect b . Ideally, the potency of a test case t , $\rho(t)$, is defined as the capability of a test case to detect bugs. That is,

$$\rho(t) = \frac{|B(t)|}{\sum |B(s_i)|} \quad (2)$$

where $B(t)$ is the set of bugs that the test case t can detect, $|B(t)|$ is the number of bugs t can detect, $\sum |B(s_i)|$ is the total number of bugs in the system. The problem is to find an ordered subset of T so that $\forall t_i \in T$ and $t_j \in T$, $0 \leq i, j \leq n$, if $i < j$, then $\wp(t_i) > \wp(t_j)$.

However, it is usually hard to obtain the accurate number of bugs present in the software that the test case can detect. In this research, we transform the bug distribution problem to a failure probability. We further consider the failures' impact and combine the two factors into a risk indicator for ranking services. In this way, testing is a risk mitigation process. Suppose that $Risk(s) = P(s) * C(s)$ is the risk of a service, the potency of a test case is defined as:

$$\wp(t) = \frac{\sum Risk(ts_i)}{\sum Risk(s_i)} \quad (3)$$

where ts_i is the set of services that t tests. To define the process, we make following reasonable assumptions:

1. a service can have at most n bugs;
2. all the bugs are independent to each other;
3. a bug is removed immediately after being detected.

The general process of risk-based test selection is as follows:

1. Calculate the risks of each service, and order the services in a sequence s_i such that $\forall s_i, s_j \in S$, $0 \leq i, j \leq n$, if $i < j$, then $Risk(s_i) > Risk(s_j)$.
2. Select the sets of test cases for each service and order the test sets in sequence T_i such that $T_i = T(s_i)$ and $\forall T_i, T_j \subseteq T$, $0 \leq i, j \leq n$, if $i < j$, then $Risk(s_i) > Risk(s_j)$.
3. Rank the test cases in each test set according to their potencies. That is, $T_i = \{t_{ij}\}$ such that $\forall t_{ij}, t_{ik} \in T_i$, $0 \leq j, k \leq n$, if $j < k$, then $\wp(t_{ij}) > \wp(t_{ik})$.
4. Select the service s_i with the highest risk and select the corresponding set of test cases T_i . Exercise the test cases in T_i in sequence until the n bugs are detected or all test cases have been run out.
5. Re-calculate the service risks and test case potencies.
6. Repeat 4-5 until certain criteria are met. We can define different exit criteria such as the percentage of services covered, the percentage of test cases exercised, the number of bugs detected, etc.

4 Risk Assessment

Based on the analysis of semantic services, risks are assessed at three layers (data, service and composite service) from two perspectives (failure probability and importance).

4.1 Semantic Web Services Analysis

A failure in a service-based software may result from many factors such as misused data, unsuccessful service binding, and unexpected usage scenarios. To calculate the risk of services-based software, this section analyzes the features of semantic web services from following three layers:

1. The domain layer, that is, how ontologies are defined and dependent to each other;
2. The service layer, that is, how ontologies are instantiated and used as the input/out data to each services; and
3. The composition layer, that is, how services are organized and affect each other in a control structure in a workflow.

4.1.1 Ontology Dependency Analysis

Ontology techniques are widely used to specify the conceptual model and semantic restrictions of a service domain. A domain usually contains a large number ontologies with complex relationships. Such relationship specification is usually a manual labeling process. An error in a class definition may affect others and propagate to a large scale along the dependencies. To analyze the robustness of the model, we need to understand how ontologies relate to each other. In this research, the dependency relationships are identified from the following three perspectives:

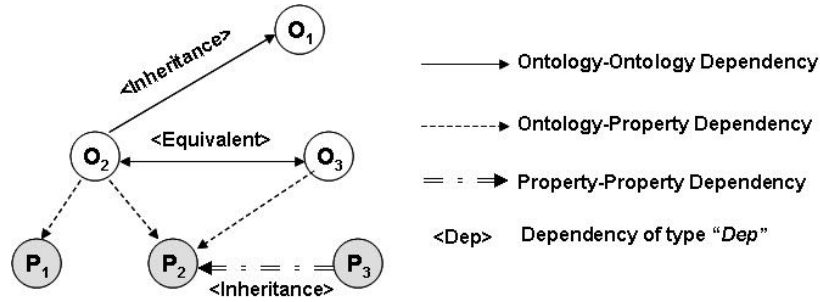


Fig. 2 Ontology dependency graph

1. **Inheritance** OWL and RDFS allow classes to have multiple inheritance relationship defined by *rdfs:subClassOf*. A subclass inherits all the properties of its super-class and can extend the super-class with its own definitions. Generically, the subclass has more restrictions than its super class. Because of the generalization relationship, an instance in the subclass should also belong to the super-class.
2. **Collection Computing** Based on the set theory, OWL define the correlations among ontology classes, such as
 - **Equivalence** Equivalent classes must have precisely the same instances. That is, for any two classes C_1 and C_2 and an instance c , if $C_1 \equiv C_2$ and $c \in C_1$, it implies that $c \in C_2$, and vice versa. OWL uses the *owl:equivalent* constructor to declare the equivalence relationship.
 - **Disjointness** The disjointness of a set of classes guarantees that an individual of one class cannot simultaneously be an instance of another specified class. That is, for any two classes C_1 and C_2 , if they are two disjoint classes, then $C_1 \cap C_2 = \emptyset$. OWL uses the *owl:disjointWith* constructor to declare the disjointness relationship.
3. **Containment** OWL supports complex class definition using set operations, such as intersection, union, and complementary. The constructors for the set operation can be combined and nested in complex class definitions.

In addition, the property of an ontology class can also be defined as classes and have the relationships listed above.

A dependency graph is defined to model the dependency relationships among the ontologies. In this directed graph, a node represents a class of ontology or property and a link represents a dependency between two classes. Different types of links are defined to represent various types of dependency relationships. Figure 2 illustrates an example dependency graph.

4.1.2 Ontology Usage Analysis

Ontologies can be used to define the inputs, outputs, operation, process and collaboration of services [43]. An ontology class can be misused in many ways such as different scope, restrictions, properties, and relationships, which may cause a failure of the service-based software. It is necessary to trace how an ontology is used in a diversified context so as to facilitate software analysis, quality control and maintenance. For example, in case an error is detected in an ontology definition, all the affected atomic and composite services can be located by tracing the usage of the ontology in those software artifacts.

Given an ontology domain D , we define $Ont(a) = \{o_i\}$ as the set of ontology classes $\{o_i | o_i \in D\}$ used in an artifact a ; and $Art(o) = \{a_i\}$ is the set of service artifacts that are affected by an ontology class o , $o \in D$ and a_i could be any type of service artifacts such as message, operation, interface, service endpoint and workflow.

Figure 3 gives an example to show the ripple effect of an ontology class. In this example, ontology classes are defined for the publication domain. A class *Book* is defined as a subclass of *Publication*. Different *BookStore* services may use the *Book* ontology to specify the input parameters of the operation *BookDetails()* in the interface *BookQuery*. An application builder defines a business process *BookPurchase* as a workflow of services. In the workflow, it first checks the prices of the book from different *BookStore* services, then orders the book from one with lower price, and finally check out the order from the selected service. We can see from the example that a domain model can be used by various services and workflows. Therefore, the quality of the domain model has significant impacts in the services and applications in the domain.

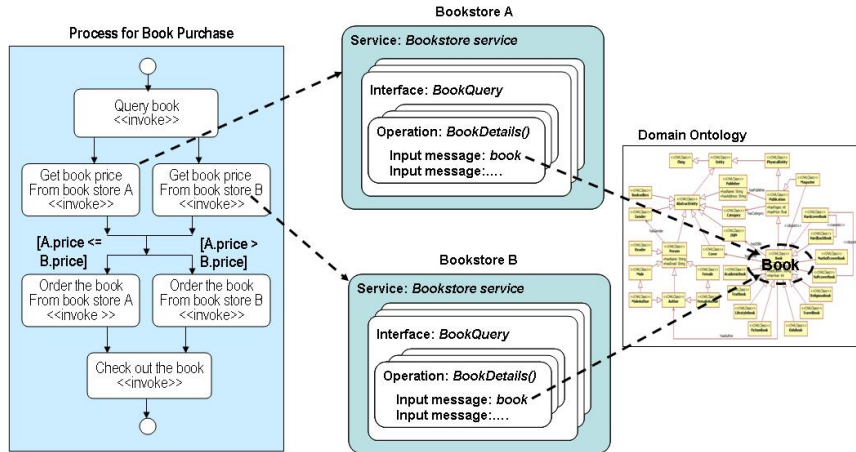


Fig. 3 Ontology Usage

4.1.3 Service Workflow Analysis

A composite service defines the workflow of a set of constituent services. For example, in OWL-S, the ServiceModel is modeled as a workflow of processes including atomic, simple and composite processes. Each composite process holds a *ControlConstruct* which is one of the following: *Sequence*, *Split*, *Split – Join*, *Any – Order*, *Iterate*, *If – Then – Else*, and *Choice*. The Constructs can contain each other recursively, so the composite process can model all of the possible workflows of WS.

The risk of the composite service depends on that of each constituent services and the control constructs over them. A service may be conditioned (e.g. *If – Then – Else*) or unconditioned (e.g. *Sequence*) executed based on the control constructs. The contribution of each constituent is proportionate to its execution probability in the composition. In an unconditioned construct, each constituent service will be executed with probability 1; while in a conditioned one, each conditional branch has a certain probability to execute. The higher the path is executed, the more the services in the path contribute to the whole composition [7].

4.2 Failure Probability Estimation

The failure probability of the service-based system is estimated as follows:

1. Estimate of the initial failure probability of each ontology class in the domain.
2. Adjust the estimation of each class by taking its dependencies into consideration.
3. Estimate the initial failure probability of an interface service.
4. Adjust the service's estimation by taking into consideration the failure probability of its input/output parameters defined by the domain ontology.
5. Given the failure probability of a set of functions and their input/output parameters, estimate the failure probability of the composite service based on its control construct analysis.

Here, we use Bayesian Statistics to analyze the initial failure probability p of each ontology class and service function. Given a certain artifact a , suppose that we test a for n times with x times failure. Suppose that X is the distribution of x and assume that X follows binomial distribution, that is, $X \sim b(n, p)$. Assume that the failure probability follows uniform distribution, that is $p \sim U(0, 1)$ and its density function is $\pi(p) = 1(0 < p < 1)$. Then, the posterior probability $P(p|X)$ follows $\beta(x + 1, n - x + 1)$ and the expectation of the posterior with respect to p is the estimator of the initial failure probability as follows:

$$E_{P(p|X)}(p) = \frac{x + 1}{n + 2} \quad (4)$$

4.2.1 Ontology Analysis

The initial failure probability is adjusted with the ontology dependency relationships. The dependence graph (DG) is transformed into a Bayesian Network (BN) for inference of the probability of each class.

The nodes in the BN represent the ontology classes and the links represent the dependency relationships. Depending on the types of class dependencies, the transformation rules are defined as follows:

1. An ontology class in DG is mapped directly to a node in a BN.
2. The property classes in DG are not shown in the BN. However, the failure probability of those property classes are used to estimate the ontology classes that the property belongs to. As an error in a property will cause an error of the ontology class, we use the product of all the properties as the affecting factor. The adjusted failure probability of the ontology class is defined as follows:

$$P_{adj}(o) = 1 - (1 - P(o)) \times \prod_{j=1..n} (1 - P(p_j)) \quad (5)$$

Where - $P(o)$ is the estimated failure probability of the ontology o and P_{adj} is the adjusted probability taking the relationships into considerations. - $p_j \in Prop(o)$ where $Prop(o)$ is the set of properties of o of length n and $P(p_j)$ is the failure probability of the property class p_j .

3. For two classes with *Inheritance* relationship, that is $Inherit(o_1, o_2)$ where o_1 is the parent of o_2 , a directed link is added to BN between o_1 and o_2 starting from o_1 and ending at o_2 to denote that o_2 is affected by o_1 .
4. For two classes with *CollectionComputing* relationship, that is $Equiv(o_1, o_2)$ or $Disjoin(o_1, o_2)$, then 1) two nodes o'_1 and o'_2 are added to BN with $P(o_1) = P(o'_1)$ and $P(o_2) = P(o'_2)$; and 2) two links are added from o_1 to o'_2 and from o_2 to o'_1 to denote the mutual dependence relationship.

Once the BN is created, the standard BN formulas can be used to calculate the probabilities as follows:

$$P(o_i|E_c) = \frac{P(o_i, E_c)}{P(E_c)} \quad (6)$$

where E_c is the current evidence (or the current observed nodes) and o_i is the node of ontology class.

4.2.2 Service Analysis

The failure probability of a service is calculated by multiplying that of its functions and its ontologies, as follows:

$$P(s) = 1 - (1 - P_f(s)) \times \prod_{i=1..n} (1 - P_{adj}(o_i)) \quad (7)$$

where

- $P(s)$ is the failure probability of a service.
- $P_f(s)$ is the failure probability of service functionality.
- $o_i \in Ont(s)$ is the set of ontology classes used in the service definition.
- $P_{adj}(o_i)$ is the adjusted failure probability of each ontology class.

4.2.3 Composite Service Analysis

The failure probability of the composite service is based on its control construct. For unconditioned execution, the failure of each service in the construct will result in the construct failure, hence the product formula is used to calculate the construct failure probability. For conditioned construct, we use the weighted sum formula where the weight denotes the execution probability of each branch that the service is located.

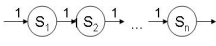
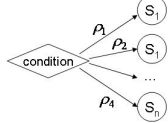
We use cc to denote control construct, and $S(cc) = \{s_i\}$ is the set of services in cc . ρ_i is the execution probability of a service s_i and $\sum \rho_i = 1$. Table 1 shows the formulas to calculate.

4.3 Importance Estimation

We use importance measurement to estimate service failure consequence. A failure in important services may result in high loss, thus the importance of a service implies the severity rank order of its failures. The importance of an element is evaluated from two perspectives:

1. Based on dependence analysis, that is, the more an element is dependent upon by others, the more important it is.
2. Based on usage analysis, that is, the more an element is used in various context, the more important it is.

Table 1 Failure probability of control construct

Category	Graphic Expression	OWL-S Example	$P(cc)$
Unconditioned		Sequence, Split, Any-Order	$P(cc) = 1 - \prod_{i=1..n} (1 - P(s_i))$
Conditioned		If-The-Else, Choice, Iterate, While-Repeat	$P(cc) = \sum p_i P(s_i)$

4.3.1 Dependence-Based Estimation

Given a domain D , the importance of ontology class o is calculated as a weighted sum of the number of its dependent classes, including both directed and undirected dependent classes. For any two classes o_1 and o_2 , if there is a path between them in DG , we define the distance $Dis(o_1, o_2)$ between them as the length of links between them. Assume that there exists at least one dependence relationship in the domain, that is, $\exists o_1, o_2 \in D$ such that $Dep(o_1, o_2)$. Then, the dependence-based importance estimation of an ontology class is calculated as follows:

$$D_{da}(o) = \sum e^{1-i} |Dep_i(o)| \quad (8)$$

$$D_{dr}(o) = \frac{C_{da}(o)}{\max_D C_{da}(o_j)} \quad (9)$$

Where

- $o \in D$ is an ontology class in the domain and $C_{da}(o)$ is the absolute importance of o while $C_{dr}(o)$ is the relative importance of o using the dependence-based approach.
- $Dep_i(o) = o_j$ is the set of ontology classes that dependent upon o with distance i , that is, $o_j \in D$, $Dep(o, o_j)$ and $Dis(o, o_j) = i$.
- $|Dep_i(o)|$ is the length of the set, that is, the number of dependent ontology classes.
- $\max_D C_{da}(o_j)$ is the maximum absolute importance $\forall o_j \in D$.

4.3.2 Usage-Based Estimation

As shown in Figure 3, an ontology class can be instantiated in various services and a services can be integrated in various business process. The usage model tracks how an ontology or a service is used in different contexts and measures the importance of the element as a weighted sum of the count of the context. Suppose that, for an element e (e could be an ontology class or a service), $Context(e) = \{ct_i\}$ is the set of context that e is used in. Assume that the element is used in at least one context for at least once, the importance can be measured as follows:

$$C_u(e) = \frac{\sum_{i=1}^{|Context(e)|} w_i Num(e, ct_i)}{\sum_{i=1}^{|Context(e)|} Num(e, ct_i)} \quad (10)$$

where

- $C_u(e)$ is the importance of e using the usage-based approach.
- $|Context(e)|$ is the number of contexts that e is used in.
- w_i is the weight for a context $ct_i \in Context(e)$.
- $Num(e, ct_i)$ is the number of e 's usage in a context ct_i .

Given a large number of measured elements, we can also use the static models to normalize the values, such as the Bayesian model for collective choice as follows:

$$C_{ub}(e) = \frac{\frac{1}{N} \sum_{i=1}^N C_u(e_i) + C_u(e)}{\frac{1}{N} \sum_{i=1}^N |context(e_i)| + |Context(e)|} \quad (11)$$

where

Table 2 Example service composition profile

Time Interval	Execution Sequence	s_1	s_2	s_3	s_4	s_5	s_6	s_7
[t_1, t_2]	$\{s_1, s_2, s_4, s_5\}$	100	80	20	80	80	20	-
	$\{s_1, s_2, s_5, s_4\}$							
[t_2, t_3]	$\{s_1, s_3, s_6\}$	50	25	25	25	-	25	-
	$\{s_1, s_2, s_4\}$							
[t_3, t_4]	$\{s_1, s_2, s_4, s_7\}$	200	140	60	140	-	60	140
	$\{s_1, s_2, s_7, s_4\}$							
	$\{s_1, s_3, s_6\}$							

- $E = \{e_i\}$ is the set of measured elements.
- $N = |E|$ is the number of measured elements.
- $|Context(e)|$ is the number of contexts that an element e is used in.

5 Risk-Based Adaptive Group Testing

A key technique of WS group testing is to rank the test cases and exercise them progressively. However, as testing progresses, changes can occur in the service-based system in various ways:

- The services could change as the providers maintain the software, update its functionalities and improve its quality.
- The service composition could change. The application builder may select different service providers for a constituent service and may change the workflow of the composite service to meet the changed requirements of business processes.
- The risks of the software could change due to changes in services and service compositions.
- The potency of the test cases could change due to changes in services and service compositions.
- The quality preference could change. For example, for a safety-critical or mission-critical usage context, it may require to have a comprehensive coverage of test cases for a service to be accepted. While otherwise, less strict criteria can be used to reduce the time and cost of testing.

To accommodate these changes, adaptation is necessary to continuously adjust the measurement of software and test cases, and the rules for test cases selection, prioritization and service evaluation. In the proposed risk-based approach, as shown in 1, a dynamic mechanism is introduced in order to enable adaptive risk assessment and test case ranking based on the runtime monitoring and profiling of the target services and systems.

5.1 Adaptive Measurement

With support of runtime monitoring, it can gather the information of the ontology dependencies, services usage, and service workflows. Profiling and statistical analysis of the logged information can facilitate to detect the changes in the system and adjust the measurement of risks and test case potencies.

For example, in an experiment, a sensor is instrumented in the process engine of composite service [5] to monitor service calls. The number and sequence of calls to each external services are recorded. Table 2 lists the typical sequence of service invocations and the number of invocations to each service in the three time intervals. Observation of the logged data show that:

- In the interval [t_1, t_2], s_2 and s_3 are conditioned executed after s_1 and the execution probability of each service are $\rho_{s_2} = 0.8$ and $\rho_{s_3} = 0.2$. s_4 and s_5 are parallel executed after s_2 . s_6 is executed after s_3 .
- In the interval [t_2, t_3], service s_5 becomes unavailable and there is no subsequent invocation to s_5 after s_2 . In addition, the call distribution between s_2 and s_3 is changed from $0.8 \sim 0.2$ to $0.5 \sim 0.5$.
- In the interval [t_3, t_4], service s_7 is newly bound to the system and invoked after s_2 in parallel to s_4 . The call distribution between s_2 and s_3 is resumed to $0.7 \sim 0.3$.

The changing process of the composition structure is shown in Figure 4. Similarly to the monitored composition changes, the system can also detect the changes in ontology domain and ontology usage. Such changes trigger a re-assessment of the risk probabilities and importance. Considering this example, Table 3 shows the changes in the risk of the composite service.

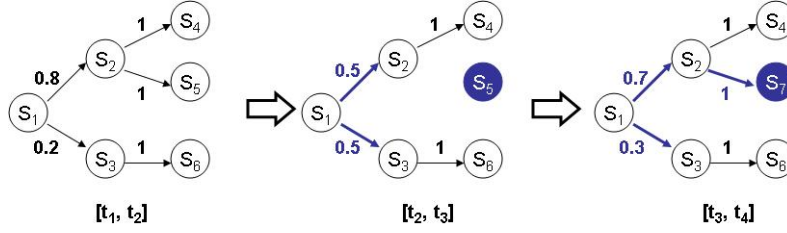


Fig. 4 Example service composition changes

Table 3 Example adaptive risk assessment

Time Interval	Risk Factors	s_1	s_2	s_3	s_4	s_5	s_6	s_7	Composite Service
$[t_1, t_2]$	$P(s)$	0.1	0.3	0.1	0.3	0.7	0.3	-	0.654
	$C(s)$	1	0.8	0.2	0.3	0.3	0.3	-	
$[t_2, t_3]$	$P(s)$	0.1	0.3	0.1	0.3	-	0.3	-	0.840
	$C(s)$	1	0.5	0.5	0.5	-	0.5	-	
$[t_1, t_2]$	$P(s)$	0.1	0.3	0.1	0.3	-	0.3	0.3	0.751
	$C(s)$	1	0.7	0.3	0.3	-	0.3	0.3	

5.2 Adaptation Rules

Rules are defined to control the testing process. In the generic WS group testing process, rules are defined to control the following testing activities:

- Risk levels to categorize the test cases and arrange them hierarchically into different layers.
- The strategies of ranking the test cases, such as cost, potency, criticality, dependency, etc.
- The strategies for ruling out services after each layer of testing.
- The strategies of ranking the services, such as importance, failure rates on the test case, etc.
- The entry and exit criteria for each layer of group testing.

In this research, testing is controlled as a risk mitigation process. That is, the test cases that have a high probability to detect a risky bug should be exercised first so that the risky bugs can be detected and removed early and that the risk of the whole system can be reduced. As bugs are detected and removed, the rules of the strategies and criteria are also adapted to reflect the changes in the risks of the services. The rule adaptation is by nature a problem of dynamic planning. Each layer in the WS progressive group testing is a stage in decision making, and the goal is to select a set of test case with maximum potential risks.

6 Evaluation

Suppose that $T = \{t_i\}$ is the set of test cases, $TE = \{t_i\} \subseteq T$ is the set of exercised test cases, $B = \{bug_i\}$ is the set of all bugs in the system, and $BD = bug_i \subseteq B$ is the set of bugs that detected. To evaluate the test results, we define the following two metrics:

1. **Test Cost.** That is, the average number of test cases required to detect a bug in the system.

$$Cost(T) = \frac{|TE|}{|BD|} \quad (12)$$

2. **Test Efficiency.** That is, given a number of exercised test cases, the ratio between the percentage of bug detected and the percentage of test cases exercised.

$$Effect(T, TE) = \frac{|BD|}{|B|} / \frac{|TE|}{|T|} \quad (13)$$

The goal of testing is to detect as many as possible bugs with as few as possible test cases [28]. That is, low test cost and high test efficiency is preferred.

Two evaluation experiments were exercised on case studies. For simplicity, we only consider the bug in the ontology classes. Assume that each ontology class has exact 1 bug, which could be detected by 1 test case. The test cases are categorized based on their target ontology classes. Each ontology class is

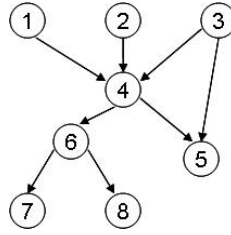


Fig. 5 The Bayesian Network of experiment 1

Table 4 Adaptive risk assessment of the ontology classes in experiment 1

nodes	1	2	3	4	5	6	7	8
Round1	0.3000	0.5000	0.8000	0.5300	0.4130	0.8060	0.7612	0.8418
Round2	0.3015	0.5036	0.8000	0.5478	0.4148	0.8617	0.7723	observed
Round3	0.3052	0.5124	0.8000	0.5918	0.4192	observed	0.7723	observed
Round4	0.3052	0.5124	0.8000	0.5918	0.4192	observed	observed	observed
Round5	0.3052	0.5124	observed	0.5918	0.4592	observed	0.7723	observed
Round6	0.3396	0.5943	observed	observed	0.4130	observed	observed	observed
Round7	0.3333	observed	observed	observed	0.4130	observed	observed	observed
Round8	0.3333	observed	observed	observed	observed	observed	observed	observed

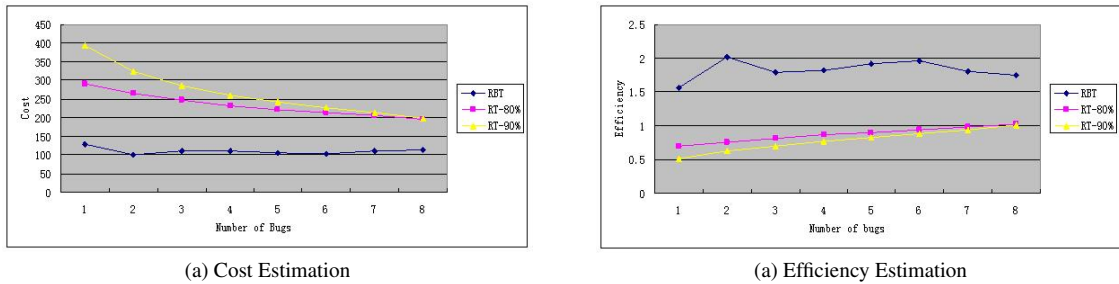


Fig. 6 Comparison of test cost and test effectiveness between RBT and RT of experiment 1

assigned 200 test cases. All the test cases are independent and they are designed for exactly 1 ontology class.

In Experiment 1, 8 ontology classes are identified and the corresponding BN network is shown in Figure 5. We simulate the joint distribution of the BN and Table 4 shows the calculated risk of each node during the iterations of nodes risk analysis and test selection. Figure 6 shows results of the experiment which compares the proposed risk-based approach (RBT) with random testing approach (RT). In the Figure, the horizontal axis lists the number of bugs detected. The curves show the trend of test cost (Figure 6(a)) and test efficiency (Figure 6(b)) with increasing number of bugs detected. Here, the RT approach is estimated with two probability levels, 80% and 90%, that is, to detect n bugs with certain probability, the number of test cases required.

In Experiment 2, 16 ontology classes are identified and the corresponding BN network is shown in Figure 7. Taking experiment 1 as a training process for experiment 2, we introduce the adaptation mechanism to rank test cases based on their defect detection history. Figure 6 shows the experiment results which compares test cost (Figure 6(a)) and test efficiency (Figure 6(b)) with increasing number of bugs detected.

From the experiment we can see that the risk-based approach can greatly reduce test cost and improve test efficiency. The learning process and the adaptation mechanism can greatly improve the test quality.

7 Related Work

More and more researchers begin to realize the unique requirements of WS testing and propose innovated methods and techniques from various perspectives [11][16], such as collaborative testing architecture, test case generation, distributed test execution, test confidence, model checking, etc. In addition to the traditional issues, the open and dynamic nature of WS impose new challenges to software testing techniques.

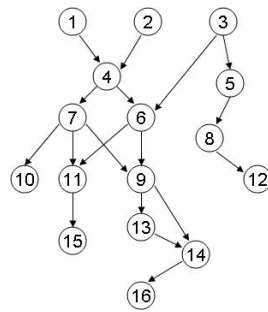
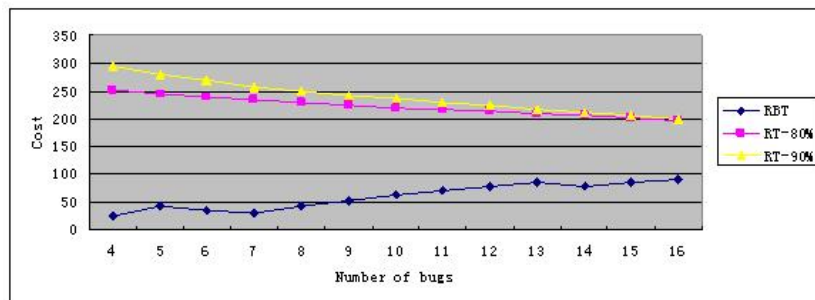
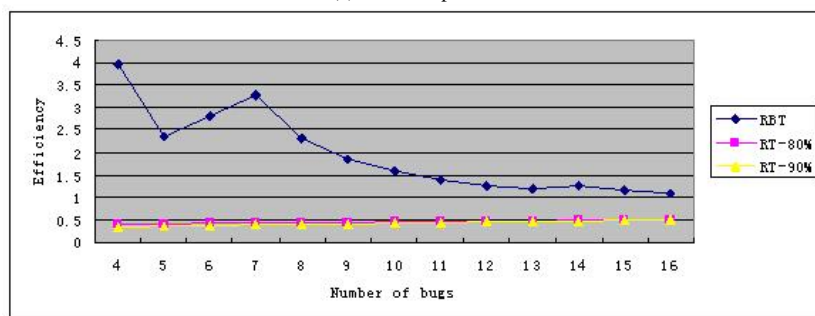


Fig. 7 The Bayesian Network of experiment 2



(a) Cost comparison



(b) Efficiency comparison

Fig. 8 Comparison of test cost and test effectiveness between RBT and RT of experiment 2

In the open platform, the service consumer cannot fully control services that are remotely managed and maintained by some unknown providers. For example, in an early research of software reliability, Kenett et al. [20] discovered that software reliability decays overtime due to the side effects of bug correction and software evolution. In a service-based system, such decaying process may not be aware to the consumers until a failure occurs. Services are changed continuously online (the "The Perpetual Beta" principle in Web 2.0 [30]). The composition and collaboration of services are also built on-demand. As a result, testing could not be fully acknowledged beforehand and has to be adaptive so that tests are selected and composed as the services change.

WS progressive group testing is proposed to enable heuristic-based selective testing in an open service environment [39][40] [3]. Group testing is by nature a selective-testing strategy, which is beneficial with reduced number of test runs and shortened test time. To refine the model, Bai et al [3] discuss the ranking and selecting strategy based on test case dependence analysis. Two test cases tc_1 and tc_2 are dependent if a service that fails tc_1 will also fails tc_2 . Hence, for any service, a test case will be exercised only if the service passes all its dependent test cases. They further propose the windowing technique that organizes web services in windows [42] and incorporates an adaptive mechanism [4][8] which follows software cybernetics theories [10] [12], in order to dynamically adjust the window size, determine web service ranking and derive test case ranking. Several challenges need to be addressed in setting up progressive group testing strategies, including

1. Estimation of probabilities;
2. Specification of dependencies;
3. Dynamic updating of estimates; and

4. Sensitivity evaluation of group testing rule parameters.

Measurement is the basis for defining the control and adaptation rules. It affects directly the test efficiency. In counter to the unique features of WS, statistical research provides promising techniques for qualitative measuring and ranking of test cases and services. Kenett et al. [21] [22] present the applications of statistics and DOE (Design of Experiments) methods in modern industry in general, and in software development in particular [26]. In particular, Bayesian techniques such as Bayesian inference and Bayesian Networks can help for estimation and prediction. Harel et al [18] apply such procedures in the context of Web Usability assessment. They further refine the controlled process with usability change tracking, service indices, usability diagnosis, and corrective adjustments [25].

Kenett et al. propose a convergence between risk engineering and quality control from the statistical perspective [24]. In software engineering, risk-based software testing is gaining attention since the late 1990s. Amland [1] established a generic risk-based testing approach based on the Karolak's risk management process model [19]. Bach [2] identifies the general categories of risks during software system development including complexity, change, dependency, distribution, third-party, etc. An important issue in the risk-based approach is the measurement of risk. In most cases, the risk factors are estimated subjectively based on experts' experiences. Hence, different people may produce different results and the quality of risk assessment is hard to control. Some researchers began to practice the application of ontology modeling and reasoning to operational and financial risk management, combining statistical modeling of qualitative and quantitative information on a SOA platform [35] and [47].

This paper is an extension of our previous research on the WS progressive group testing and semantic WS testing. It is part of the more general convergence between quality engineering and risk methodologies. Most of current WS testing research focus on the application and adaptation of traditional testing techniques. This research makes an early attempt to address the uniqueness of WS and discusses WS testing from system engineering perspective, taking the advantage of interdisciplinary research. Compared with existing works in this area, the paper contributes in the following aspects:

1. It proposes an objective method to assess software risks quantitatively based on both the static structure and dynamic behavior of service-based systems.
2. It analyzes the quality issues of Semantic WS, measures the risks of the ontology-based software services based on semantic analysis using stochastic models like Bayesian Network.
3. It improves WS progressive testing with effective test ranking and selection techniques.

8 Conclusion and Outlook

Testing is critical to assure the quality properties of a service-based system so that services can deliver their promise. To reduce test cost and improve test efficiency, the paper proposes a risk-based approach to ranking and selecting test cases for controlling the process of WS group testing. Adaptation mechanism is also introduced so that the risks can be dynamic measured and the control rules can be dynamic adjusted online. Motivated by the unique testing issues, the research shows the convergence among various disciplines including statistical, service-oriented computing, and semantic engineering. Some preliminary results illustrate the feasibility and advantages of the proposed approach.

Future work include: 1) application and experiments with an real application domain, such as the Musing project of finance ontology and operational risks; 2) service behavior analysis to identify the typical fault models and risky scenarios; 3) model enhancement to achieve better test evaluation results by using more sophisticated mathematical models and reasoning techniques.

Acknowledgements This research is supported by National Science Foundation China (No. 60603035), the Open Fund of the State Key Laboratory of Software Development Environment (No. SKLSDE-2009KF-2-0X) of Beijing University of Aeronautics and Astronautics, and the National Basic Research Program of China (973 Program) under Grant (No. 2005CB321901). The second author was partially supported by the FP6 MUSING project (IST- FP6 27097). Special thanks to Dr. Jinhua Lei from Zhou Pei-Yuan Center for Applied Mathematics of Tsinghua University, for his discussions and suggestions.

References

1. S. Amland, Risk-based testing: risk analysis fundamentals and metrics for software testing including a financial application case study, *Journal of Systems and Software*, 53(3), p.287-295 (2000).
2. J. Bach, Heuristic Risk-based testing, *STQE Magazine*, 1(6) (1999).
3. X. Bai, Z. Cao and Y. Chen, Design of a trustworthy service broker and dependence-based progressive group testing, *Int. J. Simulation and Process Modeling*, vol. 3, Nos. 1/2, p. 66-79 (2007).

4. X. Bai, Y. Chen and Z. Shao, Adaptive Web Services testing, IWSC, p. 233-236 (2007).
5. X. Bai, S. Lee, R. Liu, W.T. Tsai and Y. Chen, Collaborative Web Services monitoring with active service broker, COMPSAC, p. 84-91 (2008).
6. Shufang Lee, Xiaoying Bai and Yinong Chen, Automatic mutation testing and simulation on OWL-S specified Web Services, ANSS, p.149-156 (2008).
7. L. Wang, X. Bai, Y. Chen, L. Zhou, A hierarchical reliability model of service-based software system, COMPSAC, Vol.1, p. 199-208 (2009).
8. X. Bai, R.S. Kenett, Risk-based adaptive group testing of Web Services, COMPSAC, Vol. 2(IWSC), p.485-490 (2009).
9. T. Berners-Lee, J. Handler and O. Lassila, The semantic Web, Scientific American Magazine, May (2001). (Revised 2008)
10. Kai-Yuan Cai, Optimal software testing and adaptive software testing in the context of software cybernetics, Information and Software Technology, p. 44:841-844 (2002).
11. G. Canfora, and M. Penta, Service-Oriented Architectures testing: a survey. Lecture Notes in Computer Science, vol. 5413, Springer-Verlag, p.78-105 (2009).
12. J.W. Cangussu, S. D. Miller, K. Y. Cai and A. P. Mathur, Software cybernetics, in Encyclopedia of Computer Science and Engineering, to be published by John Wiley & Sons.
13. Y. Chen, and R.L. Probert, A risk-based regression test selection strategy, 14th ISSRE (2003).
14. R. Dorfman, The detection of defective members of large population, Annals of Mathematical Statistics, 14, p. 436-440 (1964).
15. F.M. Finucan, The blood testing problem, Applied Statistics, 13, p. 43-50 (1964).
16. M.P.Papazoglou and D. Georgakopoulos, Service-Oriented Computing, Communications of the ACM, Vol.46, No.10, p. 25-28 (2003).
17. A. Gomez-Perez, M. Fernandez-Lopez and O. Corcho, Ontological Engineering, Springer press (2005).
18. A. Harel, R.S. Kenett and F. Ruggeri, Decision support for user interface design: Usability diagnosis by time analysis of the user activity, COMPSAC (2008).
19. D.V. Karolak, Software engineering risk management, IEEE Computer Society Press, Silver Spring, MD (1996).
20. R.S. Kenett and M. Pollak, A semi-parametric approach to testing for reliability growth with an application to software systems, IEEE Transactions on Reliability, Vol. R-35, 3, p. 304-311 (1986).
21. R.S. Kenett and S. Zacks, Modern industrial statistics: design and control of quality and reliability, Duxbury Press, San Francisco (1998), Spanish edition (2000), 2nd paperback edition (2002), Chinese edition (2004).
22. R.S. Kenett and D. Steinberg, New frontiers in design of experiments, Quality Progress, p. 61-65, August (2006).
23. R.S. Kenett and E. Baker, Software process quality: management and control, Marcel Dekker Inc., New York (1999), Taylor and Francis, Kindle Edition (2007).
24. R.S. Kenett and C. Tapiero, Quality and risk: convergence and perspectives, <http://ssrn.com/abstract=1433490>, QPRC (2009).
25. R.S. Kenett, A. Harel and F. Ruggeri, "Controlling the Usability of Web Services", International Journal of Software Engineering and Knowledge Engineering, World Scientific Publishing Company, Vol. 19, No. 5, pp. 627-651, 2009.
26. R.S. Kenett and E. Baker, Process Improvement and CMMI for Systems and Software: Planning, Implementation, and Management, Auerbach Publications, Taylor and Francis, 2010.
27. D. Martin et al, Bringing semantics to Web Services: The OWL-S approach, SWSWPC (2004).
28. G. Myers, The Art of Software Testing, Wiley & Johns (1979).
29. I. Ottevanger, A risk-based test strategy, STARWest (1999).
30. Tim O'Reilly, What is Web 2.0, <http://oreilly.com/web2/archive/what-is-web-20.html> (2005).
31. F. Redmill, Exploring risk-based testing and its implications, Software Testing, Verification and Reliability, no. 14, p.3-15 (2004).
32. L.H. Rosenberg, R. Stapko, A. Gallo, Risk-based object oriented testing, 24th SWE (1999).
33. M. P. Singh, M. N. Huhns, Service-Oriented Computing, John Wiley & Sons (2005).
34. M. Sobel and P.A. Groll, Group testing to eliminate all defectives in a binomial sample, Bell System Technical Journal, vol. 38, no. 5, p. 1179-1252 (1959).
35. M. Spies, An ontology modeling perspective on business reporting, Information Systems (2009).
36. H. Stallbaum, A. Metzger, K. Pohl, An automated technique for risk-based test case generation and prioritization, ICSE, p.67-70 (2008).
37. N.N. Taleb, The Black Swan: The impact of the highly improbable, Random House, NY (2007).
38. N.N. Taleb, The Fourth Quadrant: A Map of the Limits of Statistics, Edge, http://www.edge.org/3rd_culture/taleb08/taleb08_index.html (2008).
39. W.T. Tsai, R. Paul, L. Yu, A. Saimi and Z. Cao, Scenario-Based Web Services Testing with Distributed Agents, IEICE Transactions on Information and Systems, vol. E86-D, no. 10, p 2130-2144 (2003).
40. W. T. Tsai, R. Paul, Z. Cao, L. Yu, A. Saimi and B. Xiao, Verification of Web Services using an enhanced UDDI server, Proc. of IEEE WORDS, p.131-138 (2003).
41. W.T. Tsai, Y. Chen, Z. Cao, X. Bai, H. Huang and R. Paul, Testing Web Services using progressive group testing, Advanced Workshop on Content Computing, p.314 - 322 (2004).
42. W.T. Tsai, X. Bai, Y. Chen and X. Zhou, Web Services group testing with windowing mechanisms, SOSE, p. 213-218 (2005).
43. W.T.Tsai, Q.Huang, J.Xu, Y.Chen and R.Paul, Ontology-based Dynamic Process Collaboration in Service-Oriented Architecture, SOCA, p. 39-46 (2007).
44. G.S. Watson, A study of the group screening method, Technometrics 3(3):371-388 (1961).
45. W3C, OWL Web Ontology Language Overview, available at: <http://www.w3.org/TR/2004/REC-owl-features-20040210/>.
46. W3C, OWL-S: Semantic Markup for Web Services, available at: <http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/>.
47. MUSING, Multi-Industry Semantic-Based Business Intelligence Solutions, available at: <http://www.musing.eu>.
48. Google Map API, available at: <http://code.google.com/intl/zh-CN/apis/maps/>.
49. Rent API, <http://www.programmableweb.com/api/rentrent>.
50. Metrics and QA Support, TopBraid Suite - Supporting the Compleat Semantic Application Lifecycle, TopQuarant Inc., 2009.