An Emerging Experience Factory to Support High-Quality Applications Based on Software Components and Services

(Invited Paper)

Jeff Tian
Computer Science & Engineering Dept., Southern Methodist University, Dallas, Texas 75275, USA
Email: tian@lyle.smu.edu

***Abstract*— Software components and services (SCS) are playing an increasingly important role in software engineering, particularly as building blocks of systems that demand high quality and dependability. A major impediment to advances in developing such systems is the difficulty of providing objective evaluations and conducting rigorous experiments to determine the efficacy of selected SCS and the resulting systems. This paper presents a framework that facilitates such experimentation to measure SCS quality in the target usage environment, to provide unbiased quality assessment, and to support effective usage of SCS in high quality applications. We are building a comprehensive collection of 1) usage scenarios in a set of target operational profiles and 2) a flexible defect classification and analysis framework and related quality analyses. Our work will form the first step of an operational experience factory for SCS. Resulting repositories and supporting facilities from applying our approach to web-based applications are included to demonstrate its viability.**

***Index Terms*— software quality assurance, experience factory, software components, software services, usage-based testing and operational profiles, web-based applications**

I. Introduction

Software components and services (SCS) are used as building blocks for may software systems and software-intensive systems ranging from embedded systems to general heterogeneous systems for net-centric operations (NCO) and service-oriented architecture (SOA). Commercial-off-the-shelf (COTS) and open-source (OS) SCS play an essential role in such systems. Some of these applications are mission-critical and must meet stringent dependability requirements, including high reliability, availability, safety, and security; while others require high resilience, fast response, preservation of confidentiality, etc. A major impediment to advances in developing such systems is the difficulty of conducting rigorous experiments to determine the efficacy of selected components or services and the resulting systems.

This paper describes a recent attempt at developing such an approach and related supporting infrastructure that facilitates such experimentation. A key differentiating feature of our approach is the ability to measure SCS quality in the target usage environment and to provide unbiased quality assessment, unlike testing and measurement in an environment envisioned by SCS developers. The overall infrastructure will form the first step of an operational experience factory [1], including a rich repository of data, models, packaged experience, and other software and hardware artifacts and supporting facilities, that will permit researchers to experimentally validate their SCS and systems for realistic application environments as well as under extreme conditions, and to experiment with various system design decisions to optimize the performance and quality of the resulting systems.

As the initial increment of this comprehensive experience factory, we have developed some core functions and related supporting facilities, and applied them to the web application domain. Our supporting facilities consist of 1) *target environment and usage capturing facilities* that capture real-world application scenarios in operational profiles to allow for testing of SCS under realistic usage environments and to test SCS's performance under hostile environments via boundary extension and fault injection, and 2) *quality and defect analysis facilities* that provides integrated instruments to evaluate specific quality attributes of SCS and relate them to observable defects from different perspectives. Resulting repositories and supporting facilities from applying our approach to web-based applications are included to demonstrate its viability.

II. The Need for an SCS Experience Factory

One major goal in software engineering is to deliver high-quality software products under budgetary, schedule, and other constraints, or to minimize risks of undesirable consequences such as in-field failures, budget overruns, schedule delays, and project cancellations. The concept of software quality is generally associated with good user experience characterized by the absence of observable problems or failures caused by internal faults and/or environmental disturbances. A quantitative measure of quality meaningful to both the users and the developers is product *reliability*, which is defined as the probability of failure-free operations for a specific time period or input set under a specific environment [2]. Dependability is a broader concept that encompasses reliability, availability, safety, security, etc. [3].

This work was supported in part by NSF Net-Centric Software and Systems I/UCRC.

doi:10.4304/jsw.6.2.289-297

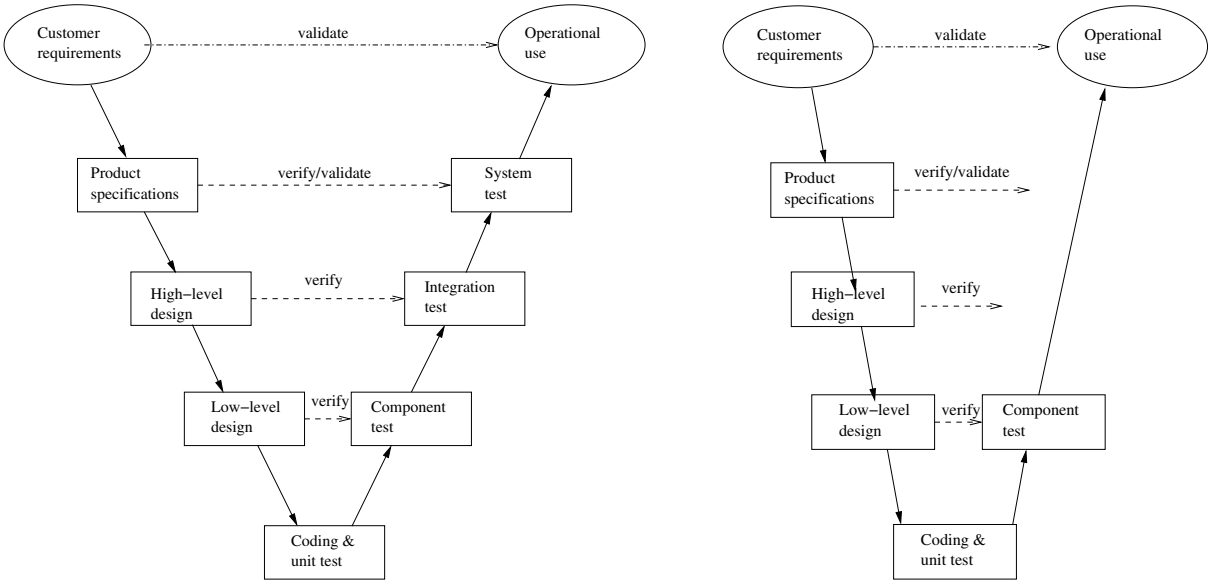

Figure 1. V-model for software products (left) and reduced V-model for SCS (right)

Software testing and quality assurance activities play a central role in assuring product quality by executing software, observing its behavior, analyzing its artifacts to locate and fix related problems or by analyzing the working product directly using either static or dynamic means. These activities aim to *validate* software against user requirements under the actual or simulated usage environments or to *verify* that product specifications and designs are followed in the implementation product. During the software development process, they are typically captured in the so called V-model, as illustrated by the left diagram of Figure 1.

Most traditional testing techniques attempt to cover major functions, execution paths, partitions and boundaries, or new and modified features [4]. They use coverage information as the stopping criteria, with the implicit assumption that higher coverage means higher quality. Alternately, product reliability goals can be used as an objective criterion to stop testing, which requires statistical usage-based testing under an environment that resembles actual usage by target customers [2]. Another important factor that affects the choice of testing techniques is the increasing size and complexity of software systems, which makes many coverage goals infeasible or impractical to achieve.

The cost and schedule constraints for modern software development, maintenance, and service integration dictate that not everything be developed from scratch. SCS are reused and integrated to form new systems or to deliver required functionalities via dynamic composition. The former approach can be characterized as component-based development that has been around for decades but has gained new momentum in the last few years, particularly with the maturation of the market for commercial-off-the-shelf (COTS) components and the proliferation of open source SCS. The latter approach can be characterized by the net-centric operations (NCO) capabilities and/or service-oriented architecture (SOA) paradigm where system capabilities are dynamically composed at runtime. However, improper reuse and integration of SCS into target systems and environments may lead to operational failures and sometimes severe consequences, such as the improper reuse of software for the Ariane 5 rocket, primarily due to the mismatch between the intended usage of SCS and their actual usage environment [5]. Therefore, a key concern for effective reuse lies in characterizing the actual usage environment and scenarios so that proper selection and/or adaptation of SCS can be carried out to ensure high quality.

In either of these SCS-based approaches, system elements must meet certain quality goals to ensure the overall system quality. On one hand, SCS are typically subjected to limited internal testing that only includes unit and component testing subphases performed by the component developers, due to the non-existence of the composite system yet. The integration and system testing subphases are left out, as illustrated by the right diagram of Figure 1. On the other hand, integration testing and system testing subphases for component-based systems are usually the responsibility of system integrators. without the benefit of the expertise of component developers. There is a big gap between the target usage environment under which the SCS need to be tested by system integrators and what the SCS was subjected to in their unit and component testing.

In addition, different users and stakeholders have different concerns, and their expectations of quality will be expressed in different quality attributes at different expected levels. Furthermore, most of the quality and dependability attributes are environment-sensitive [6]. For example, reliability is not only related to the number of internal faults, but also the usage scenarios that trigger the observed external failures [2]. To provide realistic evaluations of quality for SCS, we need to test and

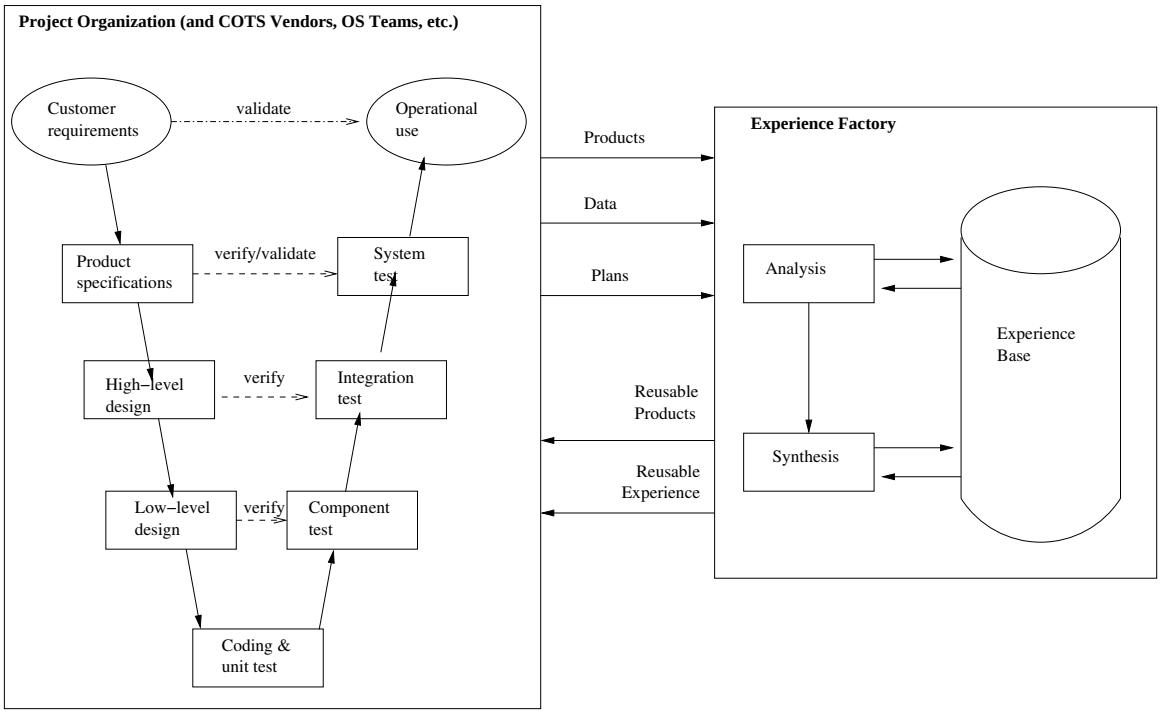

Figure 2. Experience factory (EF) for supporting software development using V-model

evaluate SCS under an environment that resembles the realistic usage environments.

Therefore, there is an urgent need to evaluate the quality of individual SCS and their contribution to the overall system quality. In addition, the analytical facilities and experimental support are needed to permit informed decision making in choosing appropriate SCS and system architecture to maximize the overall system quality.

On the other hand, substantial progress has been made in supporting software development and quality improvement through the use of the so-called experience factory (EF) [1] that consists of a collection of data, models, tools, and packaged experience supported by a dedicated group with the aid of related software tools, such as implemented in the the NASA Software Engineering Laboratory (NASA/SEL) [7]. Figure 2 illustrates the general operational relations of the EF matched to the software development process following the V-model. In this paper, we adapt EF to the specific application environment of SCS to support the specific quality goals. Some core functions and related supporting facilities of this experimental EF have already be implemented and applied to several web based applications, which are described in details after we outline our overall framework.

III. An Experience Factory for SCS: Architecture and Initial Application Domain

The overall infrastructure of our support facilities is derived from an extension and customization of an experience factory [1] depicted in Figure 3 for COTS components and open-source/other SCS. This infrastructure bridges the gap between what the component vendors provide and what the users of composite systems need. It helps the system integrators test, evaluate, and select proper SCS to compose the system prior to deployment, or to assemble the system on the fly. It also aids in validating systems in realistic environments and assuring overall system quality. It implements an open experience factory [1] customized for the community consisting of vendors of COTS/open-source components and individual services on the one side and system integrators, dynamic assemblers, and users of NCO/SOA systems on the other side, much like the Software Engineering Laboratory (SEL) did for NASA and its contractors [7]. This infrastructure consists of four interconnected units:

- a *testing facility* that recreates or simulates realistic target application environments and usage scenarios as well as usage under extreme and/or hostile conditions;
- an *evaluator* of component quality to empirically guide the selection and customization of SCS for system integration or service composition;
- an *estimator* of composite system quality based on that of its SCS and system architecture;
- an *optimizer* that takes a multivariate approach to recommend optimal system solutions.

The evaluator gets its primary data input from the testing facility. It may also use data from existing SCS and system testing/usage directly. The estimator utilizes the SCS evaluation results from the evaluator, and the optimizer utilizes the results from both the the evaluator and the estimator. The architecture of our EF and its general usage are illustrated in Figure 3. Currently, we have implemented the first two of the above (testing facility and evaluator) for a restricted web SCS, which

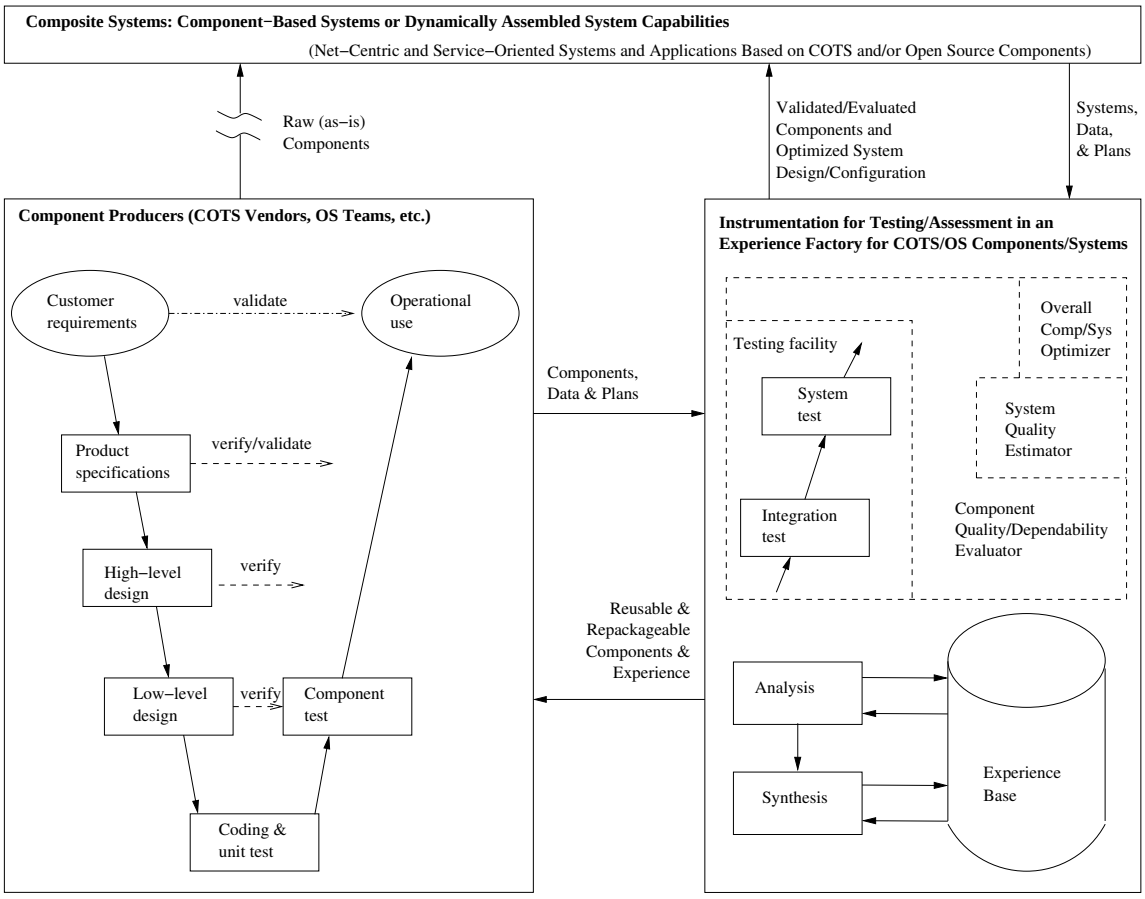

Figure 3. Overall support infrastructure for SCS in NCO/SOA as an experience factory

will be the focus of this paper. The last two (estimator and optimizer) are future extensions to the current work.

We applied the above approach to some web-based applications where web components and services are routinely used to implement and support various functions over the web. This series of case studies builds upon and extends our previous work applying and adapting traditional testing and quality measurement to the web application domain [8]–[11]. We started with an academic web site, Engineering School website at Southern Methodist University (SMU/SEAS, `http://www.seas.smu.edu`), where we have full access to the web server logs, source contents, and statistical reports produced by existing tools. Then, we gradually expanded our case studies to include three diverse web sites, including: the Open Source Project KDE website (`http://www.kde.org`), a online catalog showroom website for a small company (hereafter labeled SCC), and a commercial website for a large company in the telecommunications industry with extensive amount dynamic contents to support important of e-Commerce activities (hereafter labeled LTC). A final web site is a social networking site (SNH) that we have been working on since 2009. The results from these applications constitute the initial contents of our repository.

IV. Capturing SCS Usage Scenarios to Support Testing

The key feature of our testing facilities is the provision for realistic capturing and simulation of target usage environment under both normal and abnormal usage conditions.

A. A repository of operational profiles for realistic testing

An operational profile (OP) is a quantification of the way a software system is or will be used in field [2]. Two commonly used types of OPs are: 1) Musa's flat OP [2], [12], a list of disjoint set of operations and their associated probabilities of occurrence; and 2) Markov OP [13], [14], a finite state machine (FSM) with probabilistic, history-independent state transitions. For large systems, a collection of Markov chains may be organized into unified Markov models (UMM) [9].

We use both Musa OP and UMM in a hierarchical structure [15] to better reflect the heterogeneous nature of NCO/SOA systems. A diverse set of OPs are constructed to target a wide variety of operational environments, unlike in the traditional application of OP in testing where only one OP is needed for a generic customer usage environment. Musa OP can be constructed either through stepwise refinement [12] or supported data collection/analysis [2]. Ad hoc software tools and procedures

were also accumulated from our previous work in constructing Markov OP by first building the underlying FSM and then obtaining state transition probabilities for application domains ranging from defense to commercial and telecommunications [9], [15]. These tools are enhanced and integrated to form part of the basic collection of tools of our EF for SCS.

Data for OP construction can come from actual measurement of usage at customer installations, often with the help of some monitoring tools or system traces or logs, survey of target customers, or usage estimation based on expert opinions. As an independent party detached from either the customers or the component/service vendors, this EF for SCS will provide an objective way to build a collection of OPs to bridge the gap between the operational environments under which vendor testing was performed and the actuall usage environments where customer applications will be executed. This independence will enable us to overcome data sensitivity issues, and accumulating OPs from a wide user base, which will also help with "intended" usage from potential customers for new systems or applications. Each user will start with an analysis of her own application environment, and find the best OP available from our OP collection, to evaluate components comprising her system. If a close match cannot be found, a new OP will be constructed under the guidance of our EF for SCS.

B. A repository for testing the robustness and under extreme conditions

Various forms of testing for system robustness in handling unexpected input or scenarios can be supported by our EF for SCS. The simplest form of such testing will involve techniques such as boundary analysis and input domain testing [4], [16]. These techniques will be tailored to focus on anticipated and unanticipated extreme conditions, instead of internal boundaries and adjacent domains since they have already been tested extensively by the SCS providers.

Fault injection is a testing technique that purposefully introduces faults into systems to measure their response so that corresponding software design or implementation changes can be made to eliminate or tolerate similar faults [17]. These faults are typically related to rare-events or environmental disturbances, thus beyond the coverage scope or usage scenarios of normal testing. A closely related technique is mutation testing, where mutants, or small, localized software changes, can be introduced systematically [18]. Alternatively, software faults can be injected systematically based on defect classification data [19] or identified high-risk areas [16]. Most available tools are only suitable for a limited set of hardware or transient faults. There is limited availability of tools for injecting software faults, such as JACA and related tools (ATIFS, etc.) [20] for runtime data corruption, and Holodeck (http://www.securityinnovation.com/holodeck/) for Windows/web-service fault simulation. We are currently working on adapt these tools and other research tools for mutation testing, and develop new tools and capabilities for defect- or risk-based testing.

TABLE I.
USAGE FREQUENCIES AND PROBABILITIES BY FILE TYPES FOR SMU/SEAS

File type	Hits	% of total
.gif	438536	57.47%
.html	128869	16.89%
directory	87067	11.41%
.jpg	65876	8.63%
.pdf	10784	1.41%
.class	10055	1.32%
.ps	2737	0.36%
.ppt	2510	0.33%
.css	2008	0.26%
.txt	1597	0.21%
.doc	1567	0.21%
.c	1254	0.16%
.ico	849	0.11%
Cumulative	753709	98.78%
Total	763021	100%

C. Implementation: Web usage profiles

We have established some common patterns of web usage based on the web sites we studies, which also validated various related observations by other researchers that led to our hierarchical usage modeling [15]. At the top level, we use Musa's operational profiles, which allows us to pay special attention to highly usage types of functions and/or components. Table I gives such an OP for the SMU/SEAS web site, listing the number of requests for different types of files by web users over 26 days and the related probabilities. The adaptation and application of OP-based testing would ensure that frequently used web components are adequately tested, which in turn, would have a great impact on web site reliability improvement.

Going beyond simple hit counts for individual components or elements, we can construct our UMMs to model the overall web navigation patterns by target users. In general, such navigations are clustered, such as plotted in Figure 4, showing the cross references of individual webpages of the same web site. The sorted names of individual official pages are used as indexes in Figure 4. Each point represents a cross-reference from a specific page indexed by its x-axis value to another specific page indexed by its y-axis value. A propositionally larger dot represents the number of duplicate cross-references. The references within a unit is then typified by the short distance between their indexes due to the same leading string in their names. The associated cross references would be represented by points close to the diagonal. The usage frequencies as well as the cross-reference frequencies are very unevenly distributed, as shown by the uneven distribution of points and masses in Figure 4, thus justifying our use of statistical testing to focus on high-risk/high-leverage areas.

Figure 5 shows the top-level Markov chain of the UMM for the SMU/SEAS web site. Each node is labeled by its associated web file or directory name, and the total outgoing direct hits calculated. Each link is labeled

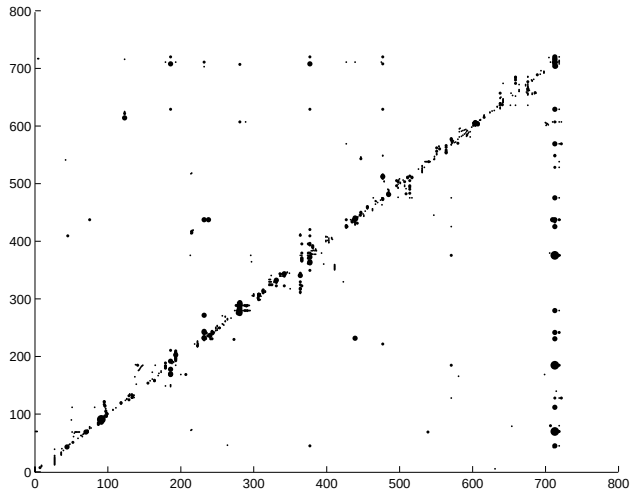

Figure 4. Cross-references for sorted individual web pages in SMU/SEAS

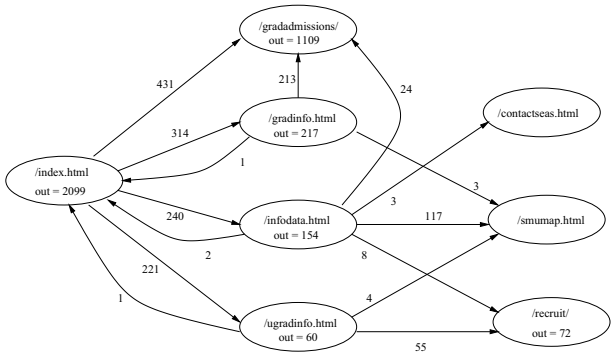

Figure 5. Top-level UMM for SMU/SEAS

with its direct hit count, instead of branching probability, to make it easier to add missing branching information should such information become available later. Infrequent direct hits to other pages are omitted from the model to simplify the model and highlight frequently followed sequences. Lower-level models are also produced for the nodes "`/gradadmission/`" and "`/recruit/`" in the top-level model. These models can be used to support our hierarchical strategy for statistical usage-based testing. Similarly patterns were also observed by the other web applications we studies, and also independently validated by other researchers [21].

V. Supporting SCS Quality Evaluation and Defect Analysis

To accommodate different users' concerns, expectations and application environments in our evaluator, quality evaluation for SCS and systems will be based on problem or defect characterization under the realistic usage environment.

A. External quality evaluation from a customer/user's perspective

For each application, the identification of relevant quality attributes will be carried out with direct involvement of customers, users, or domain experts. This will yield a list of quality attributes that are meaningful to specific customers under specific environments. Each attribute can be measured or evaluated using data from our testing facilities, or additional data from operational use, product development or existing testing. In some cases, external system logs, traces, and problem reports, such as access and error logs for many web-based applications [9], [10], [15], data from system monitoring tools [22], and defect repositories for open source software [23], also provide valuable information. All these data sources will be used for our comprehensive quality evaluation, supported by our EF for SCS.

Quality assessment typically takes the form of analyzing execution under this realistic usage environment and related failures, where a failure is an observable behavior deviations from user expectations [24]. For example, product *reliability* can be directly measured using our testing facilities, and captured by such measures as failure intensity and MTTF (mean-time-to-failures) estimated by fitting testing data to various reliability models [2]. For some quality attributes, such as safety and security, some rating levels in an ordinal scale can be used when a direct quantification is infeasible. The heterogeneous components in the NCO/SOA applications generally demonstrate vastly diverse operational behavior and characteristics. In such a system, interface failures due to erroneous interactions among different components may dominate [25]. The OPs of our testing facilities can be customized to model the interface/interactions among different components, with the rest of the system viewed as a generic "user" of a given component, and the testing results can be evaluated accordingly. In addition, system robustness can be evaluated using results from extreme or hostile testing described previously.

For each component under each OP, this evaluation will yield a quality vector whose individual elements are the corresponding values of assessment results for specific quality attributes, to form part of the data repository to be maintained and supported in our EF for SCS.

B. Defect analysis for both internal and internal quality evaluation

Since there is a causal relationship between faults and failures, where a fault is a problem in software implementation that may cause failures [24], indirect quality assessment can be carried out by assessing faults and their characteristics. For SCS studied in this paper, the usage environments characterized by our OPs are typically different from SCS vendors' testing environment, and a fault may trigger different failures under different application scenarios. By examining both the fault criticality and exposure via an OP-based assessment method we developed previously [8], [10], internal faults can be mapped into external failures. This approach will avoid the costs associated with testing for the same fault under different environments.

To systematically collect and analyze defect data, orthogonal defect classification (ODC) [19] developed by IBM and adopted by others can be customized and used. ODC has an extensive category of defect attributes, with data collected by testers or inspectors on defect discovery (external failure view) and by developers or maintainers upon defect fixing (internal fault view). Because of the uneven distribution of defects [26], analysis of ODC-like data can help us screen out low-quality SCS and select only high-quality SCS for inclusion in composite systems.

For open source components, we also have the opportunity to examine the source code so that an empirical relationship can be established between internal characteristics such as size, complexity, process and the external quality [27], [28]. This relationship can then be used to estimate external SCS quality and for SCS selection.

C. Implementation: Profiling reliability growth under usage-based testing

Under the idealized environment under usage-based testing using operational profiles, the fault that caused each observed failure can be immediately identified and removed, resulting in no duplicate observations of identical failures. This upper limit on potential reliability improvement can be measured by the reliability change (or *growth*) through the duration when such defect fixing could take place. Quantitative evaluation of the reliability growth potential can be captured by the purification level ρ [16] defined as:

$$\rho = \frac{\lambda_0 - \lambda_T}{\lambda_0} = 1 - \frac{\lambda_T}{\lambda_0}$$

where λ_0 and λ_T are the initial and final failure rates, respectively, estimated by a fitted software reliability growth model (SRGM). A larger ρ value is associated with more reliability growth, with $\rho = 1$ associated with complete elimination of all potential defects, and $\rho = 0$ associated with no defect fixing at all. $(1 - \rho)$ gives us the ratio between λ_T and λ_0, or the final failure rate as a percentage of the initial failure rate.

Figure 6 plots the reliability growth evaluation using Goel-Okumoto (GO) model [29] for the KDE data over 22 days, with usage time measured by the number of cumulative bytes transferred. It gives us a reliability growth potential of $\rho = 87.1\%$. When we used other usage time measurements, including hits, users, sessions, ρ values for KDE fall into a tight range between 86.7% and 88.9%. In other words, effective web testing and defect fixing equivalent to 22 days of operation could have reduced the failure rate to about 11% to 13% of the initial failure rate; or, equivalently, almost all the original problems could have been fixed. Similar results were also obtained for the other websites we studied.

D. Implementation: Defect classification and analysis for static and dynamic web applications

Inspired by ODC for traditional software systems [19], we developed a new defect classification and analysis technique for the web, mapping all the important ODC attributes to our framework [11]. We started this development by identifying defect impact as error type from error logs or response code from access logs [9]. The dominant "missing file" problems represent *interface* problems, which can be fixed by either supplying the requested files or correcting the referring links. The new web defect classification scheme we developed include the following attributes: *response code, file type, owner type, directory level, referrer type, agent type,* and *observation time period*. We successfully applied this classification scheme to classify and analyze web access logs for the SMU/SEAS and KDE web sites. The majority of these broken links were from internal links. Therefore, the identification and correction of the internal problems represent realistic opportunities for improved web software reliability based on local actions. The high-risk areas, or areas with substantially higher defect rate and/or defect share, were identified, with specific recommendations on how to deal with them for reliability improvement.

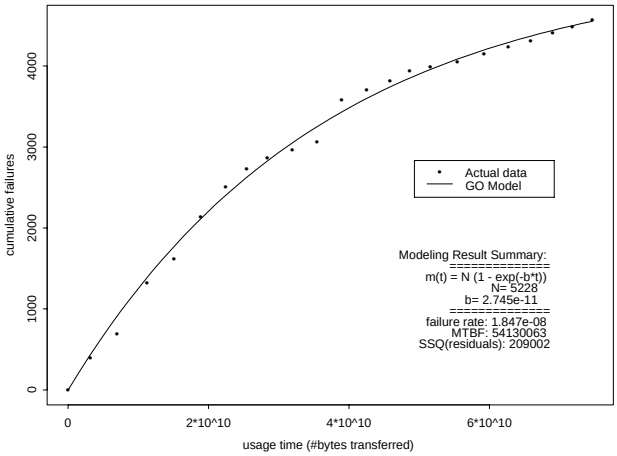

Figure 6. A sample SRGM fitted to KDE data

For dynamic web applications, such as LTC, many more "internal defects" were reported during development and maintenance activities. Although some HTTP responses carry successful response codes, they may not meet the software requirement specifications, and should be considered faults and reported as such. On the other hand, some HTTP faults may not be detected by testing. Significantly more information can be obtained if we can use both these data sources. The dynamic web ODC attributes are based on both the web log data and internal defect data from web development and maintenance activities, as modified below:

- Failure type is captured by two attributes, HTTP *response code* and domain specific failure types related to internal defects such as the 17 types [8] we used for LTC.
- *File type* is updated to include program files (Java, JavaScript, VBS, ActiveX, etc.), Cgi-Bin, etc.
- *Owner type* is expanded to include administrator,

programmer, user, etc.
- *Directory level* is also expanded to include a *directory type* attribute.
- *Referrer type* is further refined to reflect how the links were dynamically generated.

We have a semi-automatic approach to Web-ODC data extraction, with internal defect data obtained similar to in original ODC while web logs based defect data obtained using our script programs. These data and script programs form part of the repository and support facilities of our EF for SCS.

E. Implementation: Profiling accelerated defect discovery via defect classification and prioritization

For test prioritization, we use *defect density*, the ratio of unique defect over unique files. So areas associated with high defect density are identified as high risk areas, and they are tested and fixed first. Log data are split into two parts: first part as training data and 2nd part as testing data. Risks are identified and prioritized by analyzing the training data. Then test cases are selected and executed in the order guided by the obtained risk profiles. To simulate this effect, we sort the file accesses in the testing data by the prioritized areas in the risk profile, and record and accumulate the corresponding web access problems. This prioritized testing by risk based on our analysis results can be compared to various coverage based testing by comparing their respective defect discovery profiles [30].

Coverage-based testing is similarly simulated on the testing data in a similar way to produce corresponding defect discovery profiles, but without using the prioritized list above. Instead, the testing data can be sorted in the directory levels meaningful to the web domain, because people commonly assume Web files in different directory level have different priorities and conduct coverage based testing by directory order schema. That is, Web files is categorized by directory level and each level is tested at a time by the assigned order. Some may test Web files in the ascending directory level order because they think Web files in the upper level are more important that those in lower levels. Others may conduct the test in descending directory order. Yet others may create and maintain a specific list of directory-level testing order and use it to guide coverage testing.

Figure 7 shows the defect discovery profiles of coverage-based testings guided by risks and several variations of directory orders for LTC. The horizontal axis represents the unique file accesses, and the vertical axis represents the unique failures. As we can see from this result, risk-based testing compares favorably to all other types of coverage-based testing in finding and fixing higher proportions of web defects early.

We also performed similar comparative studies in the other four web sites of quite different characteristics to cross-validate the results above and to draw some general conclusions. In addition to the directory-level orders, the testing data are also sorted in the following orders meaningful to the web domain: dictionary-order, reverse dictionary-order, ascending directory-level, descending directory-level, and random directory-level. Dictionary-order coverage testing is to test the Web files by the alphabetical order of their full file name which includes the file path. That means all the tests for Web files in the same sub-directory are finished before tests for other sub-directories. In other words, dictionary-order and other variations above simulate the test scenario which test the website one subsite a time.

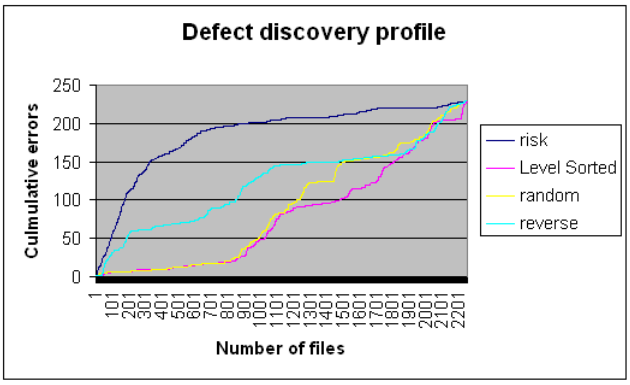

Figure 7. Defect discovery profiles comparison for LTC

In all these comparative studies, risk-based testing outperforms other coverage testing: Most of the time, the defect discovery curve for risk-based testing on top of other curves. An defect discovery curves over the other curves means given certain time constraint or resources, this testing methodology can find more defects than others under time and budget constraints.

VI. Conclusions and Perspectives

We have developed and initially validated an approach that will enable rigorous experimental quality assessment of software components and services (SCS). A key differentiating feature of our approach is the ability to measure SCS quality in the target usage environment and to provide unbiased quality assessment, unlike testing and measurement in an environment envisioned by SCS developers.

Our emerging experience factory for SCS consists of 1) *testing facilities* that capture real-world application scenarios in operational profiles to test SCS under realistic usage environments and to test SCS's performance under hostile environments via boundary extension and fault injection, and 2) *quality evaluating facilities* that provides comprehensive evaluation of specific quality attributes of SCS from different perspectives. Future work planned include an *estimator* that quantifies composite system quality based on that of its SCS and system architecture; and an *optimizer* that performs global optimization based on specific perspectives and associated value assessments using data envelopment analysis and other multivariate optimization techniques.

When fully implemented, this research will include a rich repository of data, models, packaged experience,

and supporting facilities, that will permit researchers to experimentally validate their SCS and systems for realistic application environments as well as under extreme conditions. It would substantially raise the quality, scale, and scope of experimental research in NCO/SOA systems based on COTS/OS SCS and help foster a strong scientifically based experimental paradigm for software and systems engineering. It will help parties ranging from COTS/OS SCS vendors to companies and organizations interested in NCO/SOA solutions achieve high reliability, dependability, and other quality goals.

REFERENCES

[1] V. R. Basili, "The experience factory and its relationship to other quality approaches," in *Advances in Computers, Vol.41*, M. V. Zelkowitz, Ed. San Diego, CA: Academic Press, 1995, pp. 65–82.

[2] J. D. Musa, *Software Reliability Engineering*. New York: McGraw-Hill, 1998.

[3] A. A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, Jan. 2004.

[4] B. Beizer, *Software Testing Techniques, 2nd Ed.* Boston, MA: International Thomson Computer Press, 1990.

[5] N. G. Leveson and K. A. Weiss, "Making embedded software reuse practical and safe," in *Proc. 12th Foundations of Software Engineering Conference (ACM Sigsoft 2004/FSE-12)*, 2004.

[6] S. L. Pfleeger, L. Hatton, and C. C. Howell, *Solid Software*. Upper Saddle River, New Jersey: Prentice Hall, 2002.

[7] V. R. Basili, M. V. Zelkowitz, F. E. McGarry, J. Page, S. Waligora, and R. Pajerski, "SEL's software process-improvement program," *IEEE Software*, vol. 12, no. 6, pp. 83–87, Nov. 1995.

[8] N. Alaeddine, "Anomalies analysis and classification for web quality and reliability improvement," Ph.D. dissertation, Southern Methodist University, Dallas, Texas, U.S.A., 2009.

[9] C. Kallepalli and J. Tian, "Measuring and modeling usage and reliability for statistical web testing," *IEEE Trans. on Software Engineering*, vol. 27, no. 11, pp. 1023–1036, Nov. 2001.

[10] Z. Li, N. Alaeddine, and J. Tian, "Multi-faceted quality and defect measurement for web software and source contents," *Journal of Systems and Software*, vol. (to appear), 2009.

[11] L. Ma and J. Tian, "Web error classification and analysis for reliability improvement," *Journal of Systems and Software*, vol. 80, no. 6, pp. 795–804, June 2007.

[12] J. D. Musa, "Operational profiles in software reliability engineering," *IEEE Software*, vol. 10, no. 2, pp. 14–32, Mar. 1993.

[13] H. D. Mills, "On the statistical validation of computer programs," IBM Federal Syst. Div., Tech. Rep. 72-6015, 1972.

[14] J. A. Whittaker and M. G. Thomason, "A Markov chain model for statistical software testing," *IEEE Trans. on Software Engineering*, vol. 20, no. 10, pp. 812–824, Oct. 1994.

[15] J. Tian and L. Ma, "Web testing for reliability improvement," in *Advances in Computers, Vol.67*, M. V. Zelkowitz, Ed. San Diego, CA: Academic Press, 2006, pp. 177–224.

[16] J. Tian, *Software Quality Engineering: Testing, Quality Assurance, and Quantifi able Improvement*. Hoboken, New Jersey: John Wiley & Sons, Inc. and IEEE CS Press, 2005.

[17] J. Voas, *Software Fault Injection - Inoculating Programs Against Errors*. New York: Wiley Computer Publishing, 1998.

[18] R. G. Hamlet, "Testing programs with the aid of a compiler," *IEEE Trans. on Software Engineering*, vol. 3, pp. 279–290, 1977.

[19] R. Chillarege, I. Bhandari, J. Chaar, M. Halliday, D. Moebus, B. Ray, and M.-Y. Wong, "Orthogonal defect classification — a concept for in-process measurements," *IEEE Trans. on Software Engineering*, vol. 18, no. 11, pp. 943–956, Nov. 1992.

[20] R. L. de Oliveira Moraes and E. Martins, "JACA - a software fault injection tool," in *in Proc. Int. Conf. on Dependable Systems and Networks*, 2003, p. 667.

[21] J. Hao and E. Mendes, "Usage-based statistical testing of web applications," in *Proc. 6th International Conference on Web Engineering*, Menlo Park, California, July 2006, pp. 17–24.

[22] N. Delgado, A. Q. Gates, and S. Roach, "A taxonomy and catalog of runtime software-fault monitoring tools," *IEEE Trans. on Software Engineering*, vol. 30, no. 12, pp. 859–872, Dec. 2004.

[23] A. G. Koru and J. Tian, "Defect handling in medium and large open source software projects," *IEEE Software*, vol. 21, no. 4, pp. 54–61, July 2004.

[24] *IEEE Standard Glossary of Software Engineering Terminology*, IEEE, 1990.

[25] D. Mackenzie, "Computer-related accidental death: An empirical exploration," *Science and Public Policy*, pp. 233–248, Aug. 1994.

[26] B. Boehm and V. R. Basili, "Software defect reduction top 10 list," *IEEE Computer*, vol. 34, no. 1, pp. 135–137, Jan. 2001.

[27] N. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, 2nd ed. Boston, MA: PWS Publishing, 1996.

[28] S. H. Kan, *Metrics and Models in Software Quality Engineering, 2/e*. Reading, MA: Addison-Wesley, 2002.

[29] A. L. Goel and K. Okumoto, "A time dependent error detection rate model for software reliability and other performance measures," *IEEE Trans. on Reliability*, vol. 28, no. 3, pp. 206–211, 1979.

[30] J. Tian, L. Ma, Z. Li, N. Alaeddine, and R. Geng, "Accelerated quality improvement through risk-based testing," in *Raytheon Information Systems and Computing Symposium (ISaCTN)*, Dallas, Texas, Apr. 2010.

Jeff (Jianhui) Tian received a B.S. degree in Electrical Engineering from Xi'an Jiaotong University in 1982, an M.S. degree in Engineering Science from Harvard University in 1986, and a Ph.D. degree in Computer Science from the University of Maryland in 1992. He worked for the IBM Software Solutions Toronto Laboratory between 1992 and 1995 as a software quality and process analyst. Since 1995, he has been with Southern Methodist University, Dallas, Texas, now as a Professor of Computer Science and Engineering, with joint appointment at the Dept of Engineering Management, Information and Systems. He is also Associate Director of the NSF Net-Centric Software and Systems I/UCRC (Industrial/University Cooperative Research Center). His current research interests include software testing and quality assurance, measurement, analysis and improvement of software reliability, safety, dependability, and complexity, risk identification and management, and applications in net-centric, commercial, web-based, service-oriented, telecommunication, and embedded software and systems. He is a member of IEEE, ACM, and ASQ Software Division.