R&RATA # 4
(Vol.2) 2009, December

WEB SOFTWARE RELIABILITY ENGINEERING

G. Albeanu, A. Averian, I. Duda
•
Spiru Haret University, Bucharest, Romania

e-mail: g.albeanu.mi@spiruharet.ro

ABSTRACT

There is an increasing request for web software systems, some of them to be used very intensive. The customers ask not only for fast design and implementation, but also for a high quality product. Considering reliability as an important quality attribute, this paper describes the current state of the art in designing, implementing, and testing web software. An important attention is given to software vulnerabilities and how to deliver secure software.

1 INTRODUCTION

Recently, a special class of distributed software was born, and is used intensively by people working on their terminals, situated in office or at home. The object we are talking about is called *web application,* which “is a collection of servlets, html pages, classes, and other resources that can be bundled and run on multiple containers from multiple vendors”, according to Davidson & Coward (1999). However, the term is used generic for web sites (web servers), and every software application using Internet environment.

According to Pickering (2005), “in most server architectures, the failure of any one system or service in the path between server and user will in effect cause failure of the entire application as far as the user is concerned”. The term *failure* is used according to (Randell et al. 1986), i.e. “the event of a system deviating from its specified behaviour”.

Another important aspect deals with software security. Even the security is managed separately; before the security hole is patched any failures of the application will have great impact on the application reliability. This makes difficult the usage of the standard software reliability growth models for insecure systems.

When speaking about web-servers, we have to take into account many technologies (hardware and software), each one having its own failure modes and sources of delay and unreliability, as proved by Pickering (2005).

According to the study of (Randell et al. 1986), the reliability of the web-based applications can be considered as a special case of distributed software running on distributed computer systems, over different kind of networks (local area networks, wide area networks etc.). If the nodes of the network are assumed to be perfect and the connections among nodes are assumed to fail in a statistically independent manner, then the network reliability can be computed as presented in (Shier 1991). However, if the nodes are imperfect, then the usage of the algorithm provided by (Lin et al. 1999) is an efficient solution.

The aim of this paper is to describe the relevant aspects of web software coding, testing and reliability analysis and to outline some best practices when thinking in terms of web software reliability engineering.

The above ideas motivate us to organize the paper as follows. The second section considers the web-based software design for reliability, and covers the state of the art in implementing and testing web applications. The management of the software vulnerabilities is described in the third section. For a secure web-server, aspects concerning the reliability growth modeling are considered in the fourth section. Two case studies are discussed. The first one refers to a Virtual Campus project (VC), while the second considers the DISTeFAX project. Finally some concluding remarks are formulated.

2 WEB-BASED SOFTWARE ENGINEERING

As a general rule, the web-based software is built in order to provide some functionality using different web services protocols and frameworks oriented to a specific application, as mentioned by Chu & Qian (2009). For instance, *E-business XML* (or ebXML) is a useful protocol when processing electronic business information over various platforms. Also, *ApacheAxis2* is a framework supporting many protocols, including SOAP (*Single Object Access Protocol*) for exchanging information in a decentralized distributed environment. We refer to SOAP, because “web services usually use SOAP over HTTP”, as Maeda et al. (2003) remarked.

Some years ago, speaking about the future of software reliability engineering, Lyu (2007) said: “the traditional solution that software designers adopted – carefully elicit change requests, prioritize them, specify them, design changes, implement and test, then redeploy the software – is no longer viable.” Nowadays, agile methodologies based on software components, including open source, are used to deal with rapidly software releasing, increasing reliability and diminishing the software costs (Averian et al. 2009).

According to Wasserman (2005), “the most heavily used websites are characterized by high reliability, high availability, high security, and rapid interactive response”. The discussion, in (Wasserman 2005), is oriented to the following design principles: abstraction, modularity, multi layer architecture, and logging for analyzing and testing. It is important to notice that these principles are independent on the web services provided by the web application.

The *abstraction* is used in all web-application life cycle (Schneidewind 2003, Wasserman 2005): requirements’ specification (use case diagrams, scenarios, work flow models, conceptual data models etc.), project design (by objects: images/audio/video, menus, buttons, text fields etc.), coding (based on templates), and testing (failure trees, root cause analysis, etc.).

Modularity promotes the component-based paradigm and the reuse principle, a smart usage of reusable components for constructing quality software by reducing the verification costs, increasing the software reliability, and reducing the development time (Albeanu et al. 2009a, Averian et al. 2009).

Recently, the application software follows a *multi-layer architecture*, being developed similar with some parts of the operating systems. Three-tier architecture is based on the following entities: client, server, and database. As Wasserman (2005) said, web applications have “well-known and widely followed n-tier site architecture” based on pattern design (configuration modes described using XML or other pattern languages), plug-ins (assuring an extensible architecture), with a modular structure using a specific user interface (based on languages and technologies like HTML, Flash, JavaScript, etc.).

For analyzing and testing web software, there are available a large collection of tools (components), ready to be embedded into the web application, or activated in order to monitor different aspects related to the website activity. These tools generate log files useful for “studying system performance, identifying errors, and determining general patterns of use”, as mentioned by Wassweman (2005).

Web services are offered by different web servers for specific activities. This is the reason for Chu & Qian (2009) to say: “e-business application development has certain characteristics that make it different from traditional software development”. This observation is also valid for other fields asking for high security assurance.

According to (Chu & Qian 2009), the following requirements should be taken into account for specific web applications, like those from e-business field: *service composition* (developed based on a complete system model), *formal semantics* (in order to use automated tools for service design and verification), and *systematic service design methodology* (for supporting service reuse). In this way, service reuse at different levels of granularity is also provided.

Zaupa et al. (2008), using the product line concept, proposed a web application development strategy oriented on services. In this manner, there are three stages to be followed during the development process: 1) Application domain definition; 2) Services development, and 3) Application generation.

The set of requirements has to be stable when classical software development methodologies will be used, according to Schneidewind (2003). However, in an agile framework, the requirements of a web application could be easy updated during the starting period of any iteration (when applying an iterative prototyping approach). The above three stages can be iterated when an agile methodology is used and will consider the requirements obtained in one of the following methods: classical, using UML notation, navigation-based templates, hypermedia modeling based on object thinking, and other ad-hoc, but documented models. According to (Averian et al. 2009), such an agile methodology will increase the quality of the software under development.

In order to decrease the number of faults (local or sub-network faults), the software team have to be experienced with existing vulnerabilities and security improvement mechanisms. Based on the preliminary study of Albeanu (2009), and the investigation described by (Albeanu et al. 2009a), such aspects will be detailed in the next section.

Another important aspect of web applications deals with *interoperability*. Many web applications accept as input and produce as output different objects. In order to be used/viewed/printed/listen, the object format will be a known one, and secure plug-in components will be available for clients. Here, we think about a web application in a multi server – multi client architecture. The multi server architecture is required for increasing reliability and availability by sharing connections (in a *round-robin* fashion and/or by *load balancing*). This is also the case of all software intensive systems where the *availability* is an important quality attribute, according also to (Pickering 2005, da Silva Filho 2005).

Web application *robustness* is another quality requirement: "the property of a system or a component that is totally correct in respect to a complete specification, thus its behavior is predictable for all possible operational environments", as defined in (Calori & Stalhane 2007). In order to obtain a robust web application, software engineering plays an important role. The analysis of robustness can proceed according to some methods, like those discussed by Calori & Stalhane (2007), but for critical applications like *e-business* or *e-campus total management*, the operational environment, including the security profile, will be simulated in order to test all specified requirements.

3 SOFTWARE VULNERABILITIES

If omitting the failures generated by cyber attacks, we refer to the intrinsic reliability. In large, the software reliability covers also aspects related with security holes that permit to attackers the crashing of the web application. These security holes are generated by software vulnerabilities as defined in the following. Software vulnerability deals with insecure programming and the possible insertion, by mistake, of the following classes of bugs, as identified by Albeanu (2009) and described in (Albeanu et al. 2009a, b):

– *memory-management* (buffer/stack overflow, format string vulnerability, boundary condition checking);

– *concurrency-management* (e.g. race condition involving a security check);

– *I/O-management* (e.g. input validation mistake, SQL injection, incomplete application protocol validation and verification);

– *inconsistent integration of security technologies* (e.g. configuration errors, environmental errors, incomplete access control procedures);

– *numerical inconsistencies* (e.g. integer overflow, division by zero, XOR based encryption);

– *vulnerable entry points* (command-line parameters, the environment array of strings, default input files, default passwords, inherited file data structures, inherited attributes when working with extended classes, incorrect specification of web graph nodes).

There are possible mistakes not only during design, but also during testing and implementation phases. Environmental and administrative mistakes are common when speaking about web-servers.

The vulnerabilities are possible to be identified: (1) manually (by experts), (2) automated (by bottom up and/or top down testing) (3) by black box testing, (4) by white-box analysis, (5) using scanners, and (6) combined various methods. The software trustability will be increased by testing the software using environment perturbation (taken different actions on files, other processes, network etc.).

According to Knight & Elder (2001), when coordinated security attacks are identified, "additional protection mechanisms such as closing connections over a wide area together with longer term measures such as changing cryptographic keys" are required for such faults. Non-local fault tolerance can be implemented using a specialized cryptographic protocol implemented on a cluster of servers.

If the development is based on the component-based approach, and inadequately secure components are embedded, the wrapping technique will be used for the components accommodation. Such an approach was described early in (Randell & Dobson 1986): "design means of masking or of detecting and recovering from, the security errors which might arise", and used also for the project presented in (Albeanu et al. 2009a, b).

In order to minimize the security type vulnerabilities, the prevention of the cyber attacks is the best strategy and may use the following technologies (Maeda et al. 2003): security tokens, digital signatures, encryption, and other security tools according to the security management procedure. Taking into account the above classes of bugs and the mentioned security technologies, the following types of web application attacks will be rejected: imposture (impersonation), repudiation (refusing acknowledgment), information disclosure (without permission), information altering, denial of services, and gaining the privilege of administrators or owner applications.

According to Guo & Sampath (2008), the following classes should be taken into consideration: *data storage* class covering all possible faults related to data structures, *logic faults* generated during implementing algorithms and the application control flow (some of them being related to session/paging faults, inconsistent browser interaction parsing faults, mistakes in coding encoding/decoding and encryption/decryption algorithms), *data input* faults generated by input validation mistakes related to files and forms, *appearance* faults generated by inappropriate coding for controlling the display of the web-pages, and *linking* faults due to mistakes in controlling the transfer to different locations in the World Wide Web (URL – Uniform Resource Locator). The last class is reach for the case of web applications working with URL data bases. Comparing the two classifications mentioned above we found that the taxonomy detailed by Albeanu (2009) and described in (Albeanu et al. 2009a, b) is rich enough and contains also cookies' manipulation, communication encryption, user authentication, account management, and accessing/using resources without permission.

During web software development, a model that accurately describes the vulnerabilities is required. As mentioned by (Albeanu et al. 2009a, b), "the most used vulnerability models use VCG (Vulnerability Cause graphs), C/DFG (Control/Data Flow Graphs), and decision trees". For the web applications investigated in section 4.3, the VCG approach was used. This is similar to root cause analysis method.

A VCG structure for a specific vulnerability can be built along five steps per vertex (Byers et a. 2006): validity analysis, split requirement analysis, conversion to compound vertex analysis, predecessor identification, predecessor post-processing. The VCG is stored in the vulnerability database in order to establish a complete relation between vulnerabilities and causes. The vulnerability database considers also other supplementary information like: flaw description, example of code containing the flaw, and preventive advice, as mentioned by Albeanu(2009).

Other methods uses FMEA and soft computing techniques as those described in (Calori & Stalhane 2007).

A global analysis considers both hardware and software fault categories when studying the web application reliability (Littlewood & Strigini 2000, Lyu 2007, Pham 2003). A separate analysis can be developed in the case of software faults only.

4 WEB SOFTWARE RELIABILITY

4.1 Network reliability and performability

As Pickering (2005) already identified, an important factor influencing the web-server reliability is the network reliability and availability. As measures of availability the most important are the connectivity and the performability. When a failure occurs, the network could not be able to perform at the same parameters as when working without failure. In this way, there is a strong relation between the network failure performability and the network reliability.

Various services are provided over multiple interconnected networks with different technologies and infrastructure by different suppliers (providers).

Modelling the network as an undirected simple graph, the network reliability is studied, to assure, at least theoretically, a solution to the following problems: (1) Compute the probability that there is a path between two distinguished vertices a, and b [terminal connectivity]; (2) Compute the probability that all vertices remain connected. It is clear that both combinatorial and statistical methods are mixed in order to compute the network reliability. Analysing network reliability is more important for the case of content replications motivated by requests for decreasing the answer time to a large number of simultaneously queries.

Considering the most used types of distributed web-servers (DWS) the following architectures are possible: cluster based (with virtual IP address depending on the web service visible to the clients, and a real IP address of the cluster nodes (CN), but hidden to clients), virtual cluster (the nodes sharing the same IP, and only one node will keep a message from clients), and distributed cluster (every node having its IP, and the message being redirected by a dynamic procedure applied related to the Domain Name System). The redistribution is implemented in a switching (SW) system.

The web based system reliability can be computed as in the case of serial systems: R(DWS) = R(SW) x R(CN). It is clear that R(CN) depends on the cluster topology, but experimentally we found that the estimation of R(CN) depends also on the method of content mirroring, the best results being obtained for complete replication.

Suppose G represents the network of cluster nodes that can perform if and only if it is connected, and Gr a random subgraph of G. If every edge e of G has associated a failure probability pe, then the probability that Gr remains connected is the same as G still perform. The network reliability computation can be realized using the classical results presented by Shier (1991) and Shooman (2002).

The web applications are composed by a large number of software components, many of them used in a reusable manner. The most used protocol for inter-component communication is the client-server mechanism. In this case, a component-dependency graph is built, and the component reliability is estimated (for white-box components) or approximated (in the case of black-box components) based on its average execution time, using the methodology described by Hu (2007). Based on the architecture style (sequencing, looping, concurrency supporting, fault-tolerant style, refinement, or a mixed style), and taking into account the transition probabilities among the components (estimated during a benchmarking period) the overall system reliability can be computed using methods as described in (Davila-Nicanor & Mejia-Alvarez 2004, Suri & Bhushan 2007, Tsai et al. 2004).

For the virtual campus project it was found that tree based architecture provides a high degree of reliability, and the computing of architecture reliability was fast using the MFST method described in (Lin et al. 1999). For a cluster having five nodes the best performance was obtained in a partition of type (2, 3), being also a strong fault-tolerant architecture. Actually, the web application is distributed using an architecture of type (1, 1), the availability of service being 99%.

The DISTeFAX software was designed and implemented as a secure software system facilitating the processing of meta-faxes (multipart documents obtained from individual files generated by different applications like text editors, spread sheet processors, image processing applications, etc.)

The DISTeFAX application has two main parts according to the client-server design methodology and based on the reusing principle as an agile approach for short-time releasing of software to the customers. The server part is based on the *sqLite* open source software assessed as a very fast database machine. The activities related to sending and receiving faxes are managed by a component built on top the *eFax* module (as faxing driver). The main functionality of the client module is connected with sending facsimiles and previewing and sending those received – without using any fax-modem; the faxmodems being installed only on the fax server. For managing the visualization of the facsimiles, a wrapped component based on *libtiff* was integrated into the client module.

4.2 Web software testing

It is a general assumption that fault removal is successful in the case of many software reliability growth models, as Littlewood & Strigini (2000) already remarked. For web software, only faults generating security holes are successfully removed (it is imperative necessary).

When speak about web application testing, there are two interpretations. The first one is related to software validation as mentioned by (Davila-Nicanor & Mejia Alvarez 2004, Eaton & Memon 2004, Suri & Bhushan 2007, Guo & Sampath 2008):

– *establishing the level of usability* (offering an easier navigation; conformity with standards related to content organization, and the visibility of the navigation graph);

– *checking for browser/platform compatibility* (for assuring also the portability at operating system level, and provide wide access to the web site, including by mobile technologies);

– *assuring functionalities* (the content of pages inclusive the scripts is syntactically correct and free of bugs, and all links are active; all inputs are validated, cookies are checked for correctness and security; if a database is maintained then testing all aspects related to storage, code, protocols is required);

– *communication interface testing* (checking the network/cluster connectiveness and the correctness of data transportation, including encryption protocols and acknowledgement mechanism);

– *load testing* (in order to establish the level of performance under stress testing).

– *vulnerability scanning for security assessment* (as mentioned above and detailed by Albeanu et al. (2009a, b));

The second one addresses the *unit testing* (by assertions on some regions of code – for assuring fault prevention) and the *debugging process* connected to failures, mainly by load testing (under heavy exposure). This kind of testing is useful to estimate the software reliability, and a database of failures (type, level of severity, etc.), bugs (identifier, type, location, if possible to generate security holes, …), as a time series database useful to establish both inter-failure time and cumulative numbers of failures in order to support fault/failure forecasting, will be recorded.

Even already established a user profile, the web application is, in general, open to many users. This is why we decompose the user profile in a *public profile* and a *private profile*. It is compulsory to release a bug-free web application according to the public profile (free access services), even the testing was stopped for the private profile (controlled access services) because of some schedule constraints.

When working with components, and for some of them creating some wrappers, a regression testing is required. In general, web applications are developed in an agile methodology (mainly extreme programming, or adaptive software development), and an agile testing approach is selected. In this case, the analysis of collected data is organized in batches, every batch corresponding to software life cycle iteration. For the VC project, three builds were analysed along their life cycle.

4.3 Case studies

In the following the web software reliability is analyzed using standard software reliability models (Pham 2003), as those provided by SMERFS (Farr 2003).

The web application implementing the university virtual campus was developed during three versions/builds. For all versions, the time series corresponding to test debugging were collected (Table 1), and analyzed using SMERFS, as described in (Albeanu et al. 2009b).

Table 1. VC - Time between failure data along three builds

Build No.	Number of failure	Time between failure data [days]
1	19	1, 2, 1, 3, 5, 1, 2, 4, 8, 6, 11, 17, 19, 35, 22, 52, 28, 62, 74
2	18	1, 1, 1, 2, 1, 3, 2, 4, 3, 14, 18, 11, 33, 24, 53, 71, 59, 72
3	16	1, 1, 1, 1, 2, 2, 5, 13, 12, 10, 21, 28, 38, 58, 74, 57

During analysis, for both projects, five models were selected, namely: Model 1 - Moranda's geometric model (assuming that the software is never error-free and as debugging progresses the faults become harder to detect, with the detection rate forming a geometric progression and being constant between error occurrences), Model 2 – Quadratic Littlewood-Verrall, Model 3 – Musa's basic, Model 4 – Musa's logarithmic, and Model 5 – Nonhomogeneous Poisson Process model (for execution time).

For every model the statistics concerning accuracy, bias, noise, and trend are computed. These statistics follow specific mathematical formulas depending on the model. The noise of the model is computed based on the Braun statistics (the Braun indicator is a measure of variability giving the quantitative inference about the model's noisy).

There are computed also important estimates obtained during models' execution, like: IIF – Initial Intensity Function (initial hazard rate), CIF – Current Intensity Function (current hazard rate), PrfLvl – the "Purity" Level (the ratio of the changing in the hazard rate function from the starting point to the ending and the initial value), MTBNF – Current Mean Time Between Next Failure, and KS – the measure for Goodness-of-fit calculation.

In the case of VC project, using the Goodness-of-fit measure we obtain three well-suited models for data fitting: Quadratic Littlewood-Verall, Musa's Basic, and the Nonhomegeneous Poisson Process model. These can also be identified in the pictures giving the raw and predicted data for the third builds of the project (Figures 1-3). It can be observed that Musa's logarithmic model is most pessimistic, while the best prediction is obtained using the fifth model, for short intervals of time.

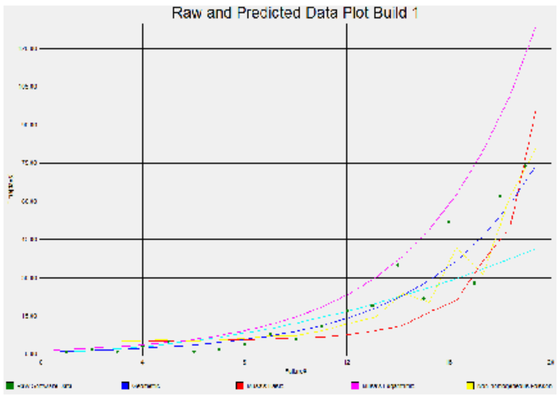


Figure 1. SRGM Analysis / VC-Build 1

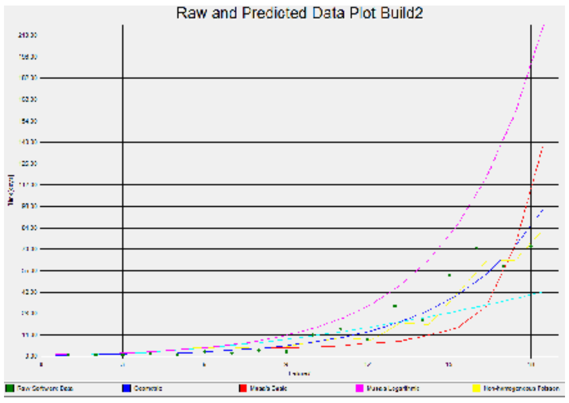


Figure 2. SRGM Analysys / VC-Build 2

The analyze using SMERF software established that for the sequence of failure data related to the DISTeFAX project (shown by value in Figure 4), the Musa’s Logarithmic model was ranked first (related to accuracy, bias, noise and trend), as revealed in Figure 5, but not recommended as the best for fitting when consider the Kolmogorov distance. The SMERF software indicates as adequate for fitting only Musa’s Basic model and the Non-homogeneous Poisson model.

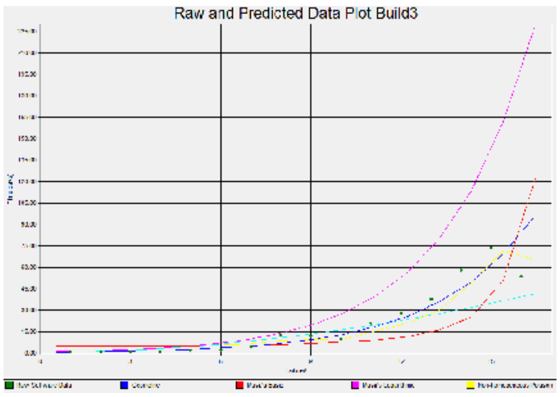


Figure 3. SRGM Analysis / VC-Build 3

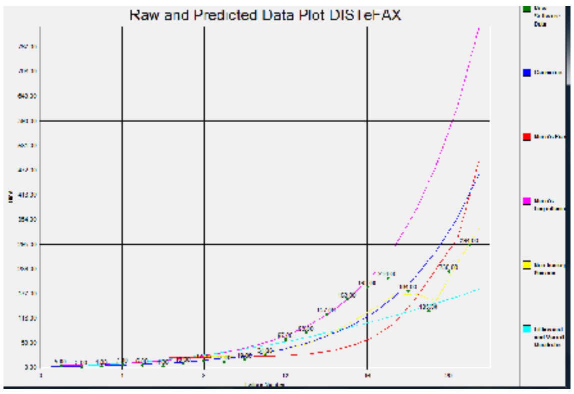
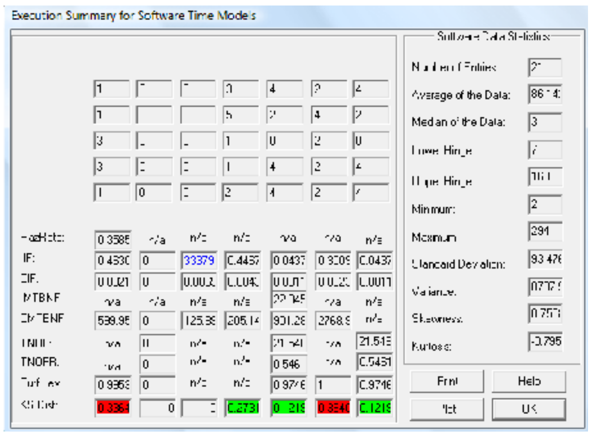


Figure 4. SRGM Analysis / DISTeFAX


Execution Summary for Software Time Models

	1	[illegible]	[illegible]	3	4	3	4
	1			5	2	4	2
	3	[illegible]	[illegible]	1	0	2	0
	3	[illegible]	[illegible]	1	4	2	4
	1	0	[illegible]	2	4	2	7
PrefRate:	0.3585	n/a	n/a	n/a	n/a	n/a	n/a
IF:	0.4530	0	33379	0.4457	0.0437	0.3209	0.0437
CIF:	0.0021	0	0.002	0.004	0.001	0.002	0.0011
MTBNF	n/a	n/a	n/a	n/a	22.745	n/a	n/a
CMTBNF	539.95	0	125.39	205.14	931.28	2768.9	n/a
TNOF:	n/a	0	n/a	n/a	21.741	n/a	21.513
TNOFR:	n/a	0	n/a	n/a	0.546	n/a	0.5451
Trend ex	0.9363	0	n/a	n/a	0.9746	1	0.9716
KS Dist:	🟥 0.3364	0	[illegible]	🟩 0.2731	🟩 0.219	🟥 0.3340	🟩 0.1219

Software Data Statistics

Number of Entries:	2[illegible]
Average of the Data:	86.[illegible]
Median of the Data:	3
Lower Hinge:	7
Upper Hinge:	1[illegible]
Minimum:	2
Maximum:	291
Standard Deviation:	93.476
Variance:	0[illegible]
Skewness:	0.75[illegible]
Kurtosis:	-0.795

Figure 5. DISTeFAX project - Execution Summary

Combinations of models of type Hsu & Huang (2009) are also possible, when they are shared basic common assumptions, and will be considered for future investigation.

5 CONCLUDING REMARKS

This paper considers a special case of distributed software, namely the web applications, which ask not only for basic quality characteristics of software, but also have to be vulnerability-free, that means be able to prevent, detect and recover a good state after a possible cyber attack.

Starting with the development of a virtual campus for a large size university the software team had to solve important problems related to web application software engineering, time releasing constraints and to provide a high quality product. The most part of practical aspects useful for finalizing such a project were covered and outlined above.

During DISTeFAX development, the team made use of many components (some of them open source), and was responsible for assuring a high degree of security by a smart approach in vulnerability identification and removal. Such activity improves also the DISTeFAX reliability.

Finally, we appreciate that a guide of best practices for web application software reliability engineering is necessary to be developed in short time to be available for students in software engineering, practitioners, and customers.

Acknowledgement. This version extends the material given in (Albeanu et al. 2009b) with more considerations about software vulnerability and a second case study, namely the DISTeFAX project.

The investigation has been realized according to the research program of the Faculty of Mathematics-Informatics at *Spiru Haret* University, Bucharest, Romania.

REFERENCES

Albeanu, G. 2009. Increasing software quality by software vulnerability engineering. *Optimum Q* 20(2): 13-24.

Albeanu, G., Madsen, H. & Averian, A. 2009a. On the influence of software vulnerabilities on software reliability: The case of open source component based software. In R. Bris, C. Guedes Soares & S. Martorell (eds.) *Reliability, Risk and Safety: Theory and Applications, Proceedings of ESREL Conference* 2:1341-1346.

Albeanu, G., Averian A. & I. Duda. 2009b. Towards web applications reliability engineering. In Krzysztof Kolowrocki & Enrico Zio (eds.) *Proceedings of SSARS* 1:15-22.

Averian, A., Duda, G., & Albeanu, G. 2009. Quality assurance for agile component-based software development. In H Pham, T Nakagawa (eds.) *Proceedings of the 15th ISSAT International Conference on Reliability and Quality in Design*: 100-104.

Byers, D., Ardi, S., Shahmehri, N., Duma, C. 2006. Modelling Software Vulnerabilities with Vulnerability Cause Graphs, *Proceedings of the 22th IEEE International Conference on Software Maintenance* (ICSM'06).

Calori, I.C. & Stalhane, T. 2007. FMEA and BBN for robustness analysis in web-based applications, In Aven & Vinnem (eds), *Risk, Reliability and Societal Safety*, *Proceedings of ESREL*: 2341-2347.

Chu, W. & Qian, D. 2009. Design Web Services: Towards Service Reuse at the Design Level, *Journal Of Computers* 4(3): 193-200.

Davidson, J.D. & Coward, D. 1999. *Java™ Servlet Specification*, v2.2, Sun Microsystems.

Davila-Nicanor, L. & Mejia-Alvarez, P. 2004. Reliability Improvement of Web-Based Software Applications, *Proceedings of the Fourth International Conference on Quality Software*, CINVESTAV-IPN: 180- 188.

Eaton, C. & Memon, A. 2004. Evaluating Web Page Reliability across Varied Browsing Environments, *Proceedings of the 15th IEEE International Symposium on Software Reliability Engineering (ISSRE'04)*, France.

Efax. *The efax home page*: http://www.cce.com/efax/

Farr, W. 2003. *SMERFS*. http://www.slingcode.com/smerfs/compare.php

Guo, Y. & Sampath, S. 2008. Web Application Fault Classification - An Exploratory Study, *Proceedings of ESEM'08*: 3003-305, http://doi.acm.org/10.1145/1414004.1414060.

Hsu, C.J. & Huang, C.Y. 2009, Reliability Analysis Using Weighted Combinational Models for Web-based Software. *Proceedings of the 18th International World Wide Web Conference (WWW 2009)*: 1131-1132.

Hu, H. 2007. Reliability Analysis for Component-based Software System in Open Distributed Environments. *International Journal of Computer Science and Network Security* 7(5): 193-202.

Knight J.C. & Elder, M.C. 2001. Fault Tolerant Distributed Information Systems, *Proceedings of the 12th International Symposium on Software Reliability Engineering*, IEEE: 132-137.

Leffler, S. 1997. *libtiff - TIFF Library and Utilities*: http://www.libtiff.org/ (last updated: 2003)

Lin, M.S., Chen, D.J. & Horng, M.S. 1999. The Reliability Analysis of Distributed Computing Systems with Imperfect Nodes, *The Computer Journal* 42(2), 129-141.

Littlewood, B. & Strigini, L. 2000. Software Reliability and Dependability: a Roadmap. *Proceedings of the Conference on the Future of Software Engineering*: 175-188.

Lyu, M.R. 2007. Software Reliability Engineering: A Roadmap, *Proceedings of FOSE*: 153-170.

Maeda, T., Nomura, Y. & Hara, H. 2003. Security and Reliability for Web Services, *FUGITSU Sci. Tech. J*. 39(2):214-223.

Pickering, R. 2005. *Web Server Reliability*, http://www.ipcortex.co.uk/wp/app-reliability3.pdf, Ipcortex.

Pham, H. 2003. Recent Studies in Software Reliability Engineering. In Pham Hoang (ed.), *Handbook of Reliability Engineering*: 285-302, London: Springer.

Randell, B. & Dobson, J.E. 1986. Reliability and Security Issues in Distributed Computing Systems, *IEEE Proceedings of the 5th Symposium on Reliability in Distributed Software and Database Systems*: 113-118.

Schneidewind N.F. 2003. Life Cycle Core Knowledge Requirements for Software Reliability Measurement, Reliability Review, *The R&M Engineering Journal* 23(2):18-29.

da Silva Filho, A. M. 2005. Toward More Available Software-Intensive Systems, *The Fifth International Requirements Engineering for High-Assurance Systems Workshop*, http://www. sei.cmu.edu/community/rhas-workshop/2005/ silvafilho.pdf.

Shier, R.D. 1991. *Network Reliability and Algebraic Structures*, Clarendon Press, Oxford.

Shooman, M.L. 2002. Reliability of Fault-Tolerant Computing Systems and Networks, *The Journal of Reliability Analysis Center* 4: 1-7&11-12.

SQLite. *SQLite home page*: http://www.sqlite.org/

Suri, P.K. & Bhushan, B. 2007. Reliability Evaluation of Web Based Software, *International Journal of Computer Science and Network Security* 7(9): 151-156.

Tsai, W.T., Zhang, D., Chen, Y., Huang, H., Paul, R. & Liao, N. 2004. A Software Reliability Model For Web Services, *8th IASTED International Conference on Software Engineering and Applications*: 144 - 149.

Unidrv. *Universal Printer Driver Components*. Microsoft: http://msdn.microsoft.com/en-us/library/ms801791.aspx

Zaupa, F., Gimenes, I.M.S., Cowan, D., Alencar, P. & Lucena, C.J.P. 2008. A Service-oriented Process to Develop Web Applications. *Journal of Universal Computer Science* 14(8): 1368-1387.

Wasserman, A.I. 2005. *Principles for the Design of Web Applications*. http:// www.se-hci.org/bridging/interact2005 /07_Wasserman.pdf.