Reaching and Maintaining High Quality of Distributed J2EE Applications – BeesyCluster Case Study***

Paweł Czarnul

Faculty of Electronics, Telecommunications and Informatics
Gdansk University of Technology, Poland
pczarnul@eti.pg.gda.pl

Abstract. The paper presents design recommendations, selected and representative implementation and configuration errors encountered during development of BeesyCluster – a J2EE component-based system for remote WWW/Web Service file management, task queuing, publishing services online for other users with credential management and team work support. Based on a QESA methodology developed previously, we build a quality tree by including the aforementioned but generalized recommendations, errors, and solutions for multi-tiered distributed J2EE applications. This allows to validate other similar applications in the future against errors we have identified and solutions we recommend thus creating a quality checklist for other J2EE developers.

1 Introduction

Although the market offers applications in a variety of fields, there is a growing need for high quality software. This is true especially in view of a large collection of open source code available on the Internet but of variable quality. The latter can be used or embedded into larger projects to solve specific tasks (within the limitations imposed by licences).

It is the quality of the development process, the methodology used, design practices and implementation techniques that contribute to the final quality of the product.

For complex applications, designers and programmers might reuse solutions to similar problems faced by others before which is often expressed as design patterns. Certainly a check-list of typical implementation errors, especially for distributed Internet-based applications, would also be useful to eliminate bugs quickly. Of equal importance are activities and issues that show up during software configuration, deployment and maintenance, usually very time-consuming but nevertheless required.

2 Motivations and Goals

Based on the facts derived above, we can conclude that every effort that classifies recurring design/implementation/deployment/maintenance problems and solutions can help improve new projects.

* partially covered by the Polish National Grant KBN No. 4 T11C 005 25
** calculations carried out at the Academic Computer Center in Gdansk, Poland

Please use the following format when citing this chapter:

Czarnul, P., 2006, in IFIP International Federation for Information Processing, Volume 227, Software Engineering Techniques: Design for Quality, ed. K. Sacha, (Boston: Springer), pp. 179–190.

A research team led by the author of this paper has successfully designed, implemented and deployed a large Web-based portal for accessing HPC (High Performance Computing) clusters, file and task management, queuing, making tasks available to others via WWW with a virtual payment subsystem and a team work environment, described in detail in paragraph 4 and [1–3]. BeesyCluster was deployed at Academic Computer Center, Gdansk, Poland as an access portal to HPC clusters including a 288-processor IA-64 holk, a 64-processor SGI Altix 3700 system and others[1]. 21 designers, programmers and documentation writers have contributed to the project over 3 years. The goal of this paper is to use the experience we have gained during the development of BeesyCluster (ca. 100 KLOC) and turn it into a concise check-list in the form of a quality tree. The paper identifies and suggests solutions to:

1. selected design problems – this will include comments on the usage of existing patterns and possibly identification of new recommendations,
2. selected implementation errors – especially useful since provides a check-list of problems the programmer might face in own applications,
3. system configuration/management/deployment problems – can be non-trivial, time-consuming and require much experience for complex J2EE and distributed systems.

Since J2EE imposes API and the multitiered architecture, this serves as a common denominator for applications considered which in turn makes this approach viable.

The quality tree which includes common J2EE problems and implementation errors is defined to automate the process of checking other applications against errors identified in BeesyCluster and making it easier to eliminate them. Each application can be evaluated in a special QESA tool, codeveloped by the author before.

3 Related Work

Firstly, existing J2EE design patterns are directly related to our work here as provide reference solutions to typical design problems encountered during development of J2EE applications. As [4] suggests the patterns are:

- reusable – can be used for several applications, are also expressed in general terms so can be applied to problems in various areas,
- developed and improved by knowledgeable designers and programmers.

[5] lists various design patterns for J2EE applications important of which are: Intercepting Filter, Front Controller, Session Facade and Web Service Broker for exposing selected services for SOAP calls.

As for avoiding implementation errors, there exist Code Conventions for the Java Programming Language ([6]) to save on software maintenance (80% of the lifetime cost of software according to [6]). Java practices are collected in [7] including issues for servlets/JSPs, coding exceptions, input/output, collections and common practices

[1] https://beesycluster2.eti.pg.gda.pl/ek/Main from anywhere, https://karawela.task.gda.pl:8443/ek/Main from Gdansk University of Technology

like defensive copying, using testing frameworks like JUnit etc. Still, J2EE specific errors are not addressed.

Secondly, we try to automate the process of checking the quality of design, implementation, configuration by including the identified practices, errors into a quality tree. This is related to existing general software quality models and defect classification methods.

There are several general quality approaches available. The Goal-Question-Metric (GQM, [8]) method first specifies goals to achieve, formulates questions which help achieve the goal, defines metrics for which data is collected and answers the questions ([8]). COCOMO ([8]) and Function Points ([8]) can be used to measure the required effort and software size. Software Process Improvement and Capability Determination (SPICE, [8], [9], published as ISO/IEC TR 15504) is an international initiative aimed at the standard of software process assessment, used in the context of process improvement or process capability determination either of an organization or a supplier. SPICE defines a framework for performing evaluation, required activities, defines how to conduct software evaluation. [10] presents system-level quality metrics for component-based systems that can help managers decide whether existing components should be reused.

In this work a QESA approach, introduced by us in [11] for improving design of an application for management of ship containers, will be used to build a quality tree including design practices, errors and recommended solutions. Paragraph 6 discusses the QESA methodology and compares it to defect classification methods like IBM's Orthogonal Defect Classification and HP's Company-Wide Software Metrics ([12]).

4 BeesyCluster

BeesyCluster can be seen as an access portal to a network of clusters/supercomputers/PCs with WWW and Web Service interfaces. Figure 1 depicts the architecture of the system with main modules and relationships (described in detail in [1]). The user sets up an account in BeesyCluster through which (single sing-on) can access accounts on many different clusters/supercomputers/PCs. Users can manage files and run sequential or parallel tasks (interactively or queued) on their accounts on clusters/supercomputers/PCs via WWW and Web Services. Furthermore, users can publish their services (applications, sequential or parallel, run interactively or queued on clusters/supercomputers as well files) to other users of BeesyCluster. For the use of services (if not specified as free of charge), users-providers earn points which can be spent on running services published by others. Users can register new clusters or individual PCs in the system just by providing a login/password to any system account and can run tasks, edit files and publish services from there.

BeesyCluster is representative in terms of:

distributed architecture – the user connects to BeesyCluster via WWW or Web Services while the system uses SSH to connect to accounts on remote clusters/PCs and run tasks there – we run several demanding parallel applications using this system (described e.g. in [13]),

access via multiple popular interfaces – WWW and Web Services (its efficiency in BeesyCluster tested in [2]),

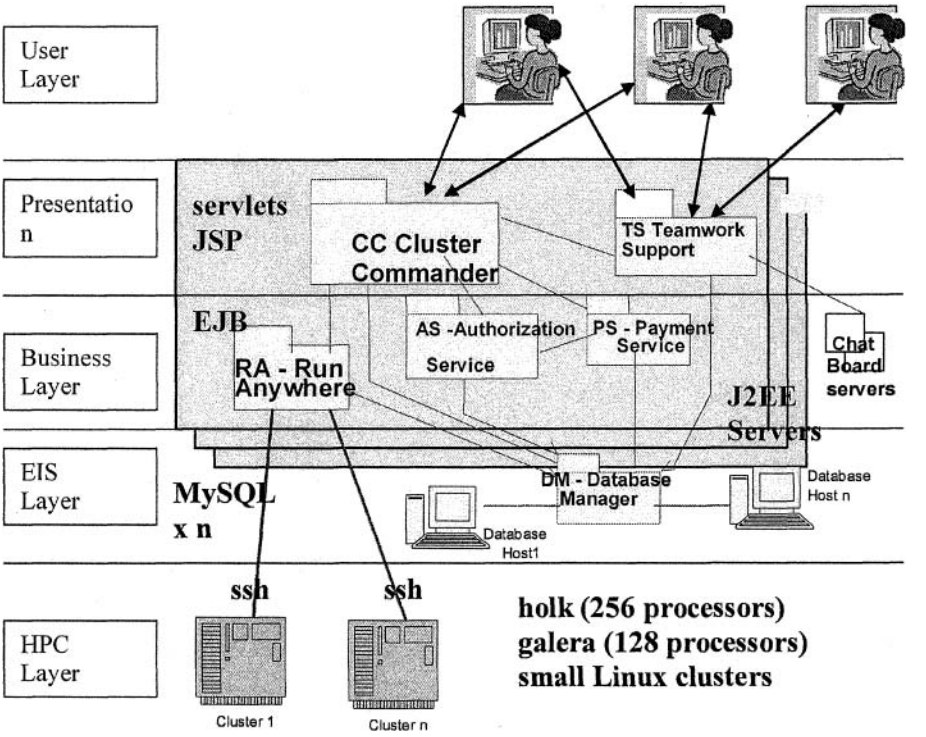

Fig. 1. Architecture of BeesyCluster

grid computing – the user can mark an application to be available as a service via WWW or Web Services from both accounts on clusters as well as even desktop PCs – this implements controlled resource sharing i.e. grid computing,

data replication – uses data replication in several databases for which consistency must be maintained and is handled by a custom-built distributed database replication mechanism on MySQL outside of J2EE,

clustering – uses multiple J2EE servers to increase availability and reliability,

session and security handling using WWW and Web Services (described in paragraph 5.1) – handling security identities and rights to the resources (digital signatures with asymmetric cryptography are used),

modular design – the system is composed of modules which can be implemented independently and share the same top-level compilation scripts,

variety of interactive services via applets – BeesyCluster uses two dedicated Java servers for chat and a board shared by users for interactive collaboration, another applet implements an online remote shell on clusters,

building scientific workflows – services on clusters can be combined into complex scientific workflows ([3]).

5 Classification of Patterns/Solutions to Typical Errors Identified during Development of BeesyCluster

5.1 Selected Design Problems/Solutions in BeesyCluster

In this paragraph, we distinguish selected design problems and their solutions in BeesyCluster (Table 1, [1]). This is done in view of the existing J2EE design patterns, also in a broader context of current and future technologies which are suggested for implementation.

Tab. 1: Selected Design Problems/Solutions in BeesyCluster

Problem	Solution
Portable Authorization and Session Management for Various System Interfaces and Clients	Since complex applications can use various interfaces like WWW, Web Services, listen on sockets using a proprietary protocol, wait for a file system change etc., a portable and compatible way of authorization and session management between calls must be used. BeesyCluster suggests a way in which the user logs in with a username/password and obtains an encrypted token which is passed with following calls (steps analogous to publishing data in UDDI). In the system there is a dedicated business component for authorization based on a database. Then for: 1. WWW requests: authorization can be done within an Intercepting Filter ([5]) which verifies the token by calling the business component per each request before delegating the request to following components, possibly Front Controller. Although J2EE has a way of defining roles that may access Web components and J2EE server users may be mapped to these roles, this way is more flexible since can employ e.g. runtime variables as time of day or IP into granting access. The token which handles session information may be stored in a cookie or in a session object on the server and be identified by a cookie. 2. Web Services or other interfaces: a method for logging in is a first required step which returns a token which is then used as an additional parameter to successive calls ([3] explains the way it is implemented in BeesyCluster). "Business" Web Services (which call EJBs) call a business component to verify access. Similarly, the proprietary protocol for TCP communication might use the same token. This means that the user could possibly start a session using WWW and finish using Web Services from another device.
Separation of Java Code from Web Pages and Instant Review of Page Changes	Although the J2EE standard defines the presentation layer (servlets, JSPs) and business logic layer (EJBs), still servlets and especially JSPs can contain control statements (patterns like Front Controller or Composite View [5]) as well as formatting for Web pages. It can be recommended to use a technology purely for presentation/formatting output. In our case, we used Velocity which displays (using proper templates) output variables (from proper business methods) or arrays set in servlets. Furthermore, changes in templates do not require recompilation which speeds up the development.

continued on next page

Problem	Solution
Multiple Extensible Interfaces to the System and Business Layer Separation	In today's world, apart from the WWW interface for human-system interaction over the Internet, complex applications need means to communicate among themselves. We used Web Services (based on AXIS), currently an element of J2EE, to provide such possibilities. In fact the Web Service Broker pattern suggests this approach. Still, other interfaces might be needed like more efficient proprietary protocols over TCP etc. J2EE is well prepared for this as business methods may be called by endpoints handling these interfaces e.g. servlets/JSPs for WWW, Web Service for SOAP, a server listening on sockets etc. From this perspective, it seems crucial that business methods are sufficiently isolated (Session Facade [5]).
Minimizing latency to data layer and external systems	This should be done by proper caching of data: 1. when fetching data from external systems or the database, part of it should be reused for following client requests if possible (e.g. reloading the left panel of the file manager does not cause querying of the right panel of another cluster), 2. in the presentation layer: technologies like AJAX allow to exchange XML data with the server without reloading the entire page.
Uniform Logging Facility	Logging can be incorporated into an Intercepting Filter but only for presentation layer components. A dedicated logging component (e.g. bean) is suggested recording the id of the calling module, time, the user who has requested the operation, users whom the operation affects, priority, description. It is recommended to define logging levels to reflect the J2EE layers (presentation, business). Logging in the presentation layer should be turned off when EJBs already log detailed information.
Transparent Parallel and Reliable Access to Data	Usually data would be retrieved from a database by entity beans (BMP or CMP). Still, it is desirable that there is a mechanism, transparent to the programmer, that hides potentially parallel access to several databases for both increasing the throughput of e.g. SELECT queries and reliability (if some database nodes fail). This can be configured in both commercial engines and e.g. MySQL where a master node and slave database server nodes can be configured. Within BeesyCluster, an extension to the MySQL solution was implemented which changes a slave to the master if the current master fails. Additionally, synchronization algorithms can be changed to e.g. quorum consensus and others easily ([1]). This in fact suggests a more complex sequence diagram for the standard Data Access Object pattern ([5])
Client-aware Interface	Although fast broadband Internet connections have become mainstream, the client-system data transfer should be client-aware because of mobile devices like palmtops or mobile phones with limited memory and processing capabilities (MIDP 2.0 requires 128KB for the Java runtime heap, 8KB for persistent data, a screen of 96x54 pixels). Crucial Web, Web Service or other resources should take the maximum returned data size parameter. This can be done with the standard request e.g. by: 1. another request parameter for HTTP transfer, 2. another header in a SOAP message for Web Services ([14]). Revert to a basic but functional interface for less capable browsers.
Minimize Response Time by Advance Queries	Periodic calls with output to be used by user queries (e.g. monitoring the state of remote systems or databases to be queried next) should be done by threads in the background (threads or separate servers). The output (possibly somewhat out-of-date) is fetched when the user request is handled. JMS communication with threads is suggested.

5.2 Selected Implementation Errors Identified during Development of BeesyCluster

Table 2 lists selected implementation errors or recommendations identified during the development of the system. These are likely to occur in other complex applications.

Tab. 2. Selected Implementation Errors Identified during Development of BeesyCluster

Layer	Errors or Recommendations to Avoid Errors
Presentation Layer	1. Initial values not filled in web forms. 2. Presentation layer servlets and JSP pages using hardcoded ids (e.g. clusters or users) not from the database thus making it inconsistent with ids used by the business layer components. 3. Specific parameters (text boxes) cause problems (e.g. spaces in the names of directories). 4. Access to specific servlets or JSP pages should not be granted to users with restricted privileges (missing conditional instructions). 5. Test functionality of the interface using 1 client, always use 20+ concurrent client requests from various nodes to test response times, isolation of transactions, potential deadlocks when referring to same resources. 6. Always disable display of exception details for production version, log details to a log, always print information to a log in catch blocks. 7. Avoid a long sequence of page reloads (3+) to complete a task, could be completed within one page (using e.g. AJAX). 8. Use only one way of fetching session information in web components.
Presentation--business Layer Interaction	1. When processing in business method takes 5+ seconds, call it asynchronously, store a handle and allow to retrieve status or make the presentation layer show progress until results are available. 2. Data presentation not handled properly for certain input data to the business layer or error codes from the business layer not interpreted.
Business Layer	1. Errors in EJB components which are likely to be detected only during the real deployment of that module. Example: errors of task submission to a real cluster from the module (via the Jsch Java library). 2. Long response times or hangs when submitting many requests to an external system in a short time frame – configure external systems properly. On cluster holk command must be run via a proxy node - initially via rsh. rshd on holk refused connections in the case of many concurrent requests (ports up to 1023 can only be used). Using ssh solved the problem.

5.3 System Configuration/Deployment Errors and Solutions

Management of configurations especially in the case of multiple installations of a system, possibly on different architectures is challenging. BeesyCluster's official release runs on Solaris while the development version on Linux.

Tab. 3: System Configuration/Deployment Errors and Solutions

Issue	Items
Security	1. HTTP connection available after testing, should leave only HTTPS. 2. Errors with certificates in HTTPS access from certain browsers (error for self-signed certificates where Common Name (CN) of the issuer and CN of the entity the certificate was issued to are identical - Mozilla, Konqueror). 3. Securing physical access to servers (accidental restarts by other users). 4. Hide URLs for services where possible (e.g. by a proper Front Controller pattern passing parameters for selected URLs). 5. Write a client for exposed URLs requesting with random parameters and use it for testing.
Database Configuration	1. Error in scripts filling the database with initial data (SQL statements not accepted by later MySQL versions, worked correctly on the version, BeesyCluster was originally deployed on). 2. Modification of a single node of a cluster of replicated databases. During some tests using one node, only a single database was modified and another backed up as a master.
System Configuration	1. Problems with specific versions of required libraries e.g. xdoclet pre 1.2.2 caused compilation errors while newer versions worked correctly, 2. Problems with migration from Java 1.4 to 1.5, qualified names should be used in the code due to the conflict with classes from Java 1.5, 3. Inconsistent configuration (versions of software) and startup scripts across the cluster of servers, need for a tool updating all nodes or NFS, 4. Some services would not start properly after system was restarted although the core of the system worked correctly (Java chat/whiteboard servers). 5. Uniform configuration and compilation scripts for all modules are recommended. It is possible to define a top-level build.xml file so that a new module can simply be added by copying its directory into the existing sources and no or very few additional changes are required. 6. Failures of operating system servers cause selected servers used by the system to fail. Creation of a simple monitoring tool with restart of services is recommended.
Versioning	1. Components were updated on one of the J2EE servers instead of all the servers which resulted in errors on those servers. Use a distribution tool to distribute changes to all servers. 2. Submission of incorrect versions of components to a server for deployment – already corrected errors/bugs would show up again.

In view of clustering and replication to increase the number of clients the system can handle in parallel/concurrently and inconsistencies of configuration across the cluster, errors of this type in BeesyCluster (Table 3) can be applicable to other systems as well.

6 Quality Modeling and Evaluation in QESA

6.1 QESA Methodology

The QESA methodology ([11]) uses a generic QESA quality tree (Figure 2) to evaluate the quality of a product or phase by general top-level external quality attributes each of which is defined by either four or five quality factors at the second level (several translation functions are available). In the QESA methodology, these two levels are fixed since are thought to be general enough to suit any application, development phase or product. Depending on whether a development phase or a product is evaluated, factors will be further defined by more precise metrics at the lower and measures at the lowest level of a four-level quality tree – both chosen by the user to suit the application. As an example attribute *dependability* defined by factor *error-tolerance* could be defined by metric *presentation layer errors* and this by question *whether access to page tested when no user logged in.* Then answers to questions in measures or their numerical values propagate up the tree and generate final values for quality attributes.

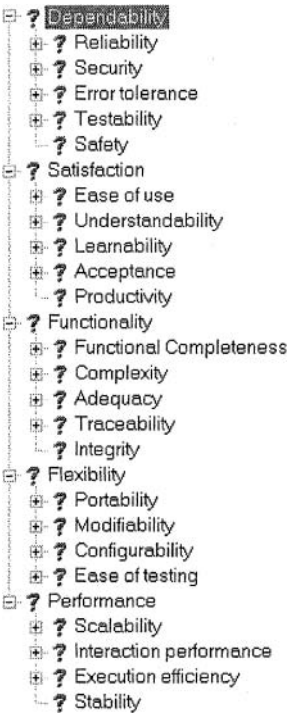

Fig. 2. QESA Quality Tree - Two Top Levels

QESA allows e.g. metrics to contribute to a factor by a decreasing function. Usually a more complex and fancy user interface improves visual effects while decreases interaction performance. Measures being in fact internal quality attributes are defined with values in their own domain (e.g. seconds or LOCs) and normalized into the [0,1] range. Quality attributes, factors and metrics are defined within the range [0;1], the higher value meaning better quality at the highest level.

In fact, as applied during classes on Software Quality courses at Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology, the QESA quality tree could be used in many ways, two of which are:

1. During the software development cycle, a new quality tree is created for each phase with metrics and measures specific for the given phase.
2. For the comparison of products e.g. complete applications, a reference quality tree is created with metrics and measures specific for the given type of product and evaluation is performed for each product. Values can be compared in the QESA system. In particular, an aggregate value for higher level factors and attributes can be compared.

6.2 Modeling Quality of BeesyCluster as a Template for New Applications

Modeling quality of BeesyCluster as a quality tree will allow other applications to be verified against the errors, deficiencies and design strategies suggested in paragraph 5.

For the BeesyCluster system, we have created quality models (trees) with metrics and measures specific for distributed and parallel applications which is our area of expertise ([13], [3]). Quality trees refer to:

design – measures are simply questions whether the design principles given in paragraph 5.1 are met (yes/no) or in what degree (numerical value),

implementation – whether the code has been validated against the errors listed in paragraph 5.2 and other basic coding standards,

testing – system tested for some implementation errors from paragraph 5.2 and configuration/deployment from paragraph 5.3.

As an example, the programmer/user of a new system specifies in the testing phase response times or whether form parameters have been tested. The values are processed by QESA which produces a final quality values for dependability, satisfaction, functionality, flexibility and performance. If the quality is satisfactory a new phase may start. This approach is similar to IBM's Orthogonal Defect Classification from 1992 ([12]) where in each phase numbers of defects of eight types are noted depending on the repair needed for the defect. Then the changes of distribution of defects between phases are compared to expected patterns. Process Inferencing Tree is built to track defect changes between phases. Similarly, in tracking quality QESA is similar to HP's Company-Wide Software Metrics from 1987 ([12]) which classifies defects into types depending on the phase and assigns mode e.g. *missing* for missing error checking. If other projects data is available, trends can be observed.

An exemplary part of the QESA quality tree for BeesyCluster's testing is shown in Figure 3 and includes the metrics and measures corresponding to items listed in paragraph 5.3. Resulting quality charts for BeesyCluster without the identified points (related to errors from paragraph 5.3) improved are shown in Figure 4 and after corrections in Figure 5. After the improvements the system can still be corrected e.g. a better interface can be engineered (as also reported by the attendees of a training course) or the response time can be reduced thanks to faster hardware.

For distributed J2EE applications such as BeesyCluster, the highest-level quality attributes given the largest weights (angles in Figures 4 and 5) are:

1. dependability especially error-tolerance i.e. how the system tolerates errors (here we assume that if several issues identified in BeesyCluster are not checked and tested for, the system may give undefined results), reliability (the system must be available and functional at all times) and security since providers must be certain their resources cannot be compromised beyond what they permitted,
2. functionality mainly functional completeness in the case of BeesyCluster being remote task execution, management, making resources available, receiving proper payments for the resources checked out etc.,
3. performance especially interaction performance (the system must respond in less than a few seconds for any request), scalability (must scale well with the number of servers and users).

The presented quality tree is available from the author. The QESA (SOJO in Polish) system can be downloaded from http://fox.eti.pg.gda.pl/~pczarnul/SOJO-1.0.zip. A Web-based version of QESA is available at http://153.19.53.71/qes/page.tytul.php.

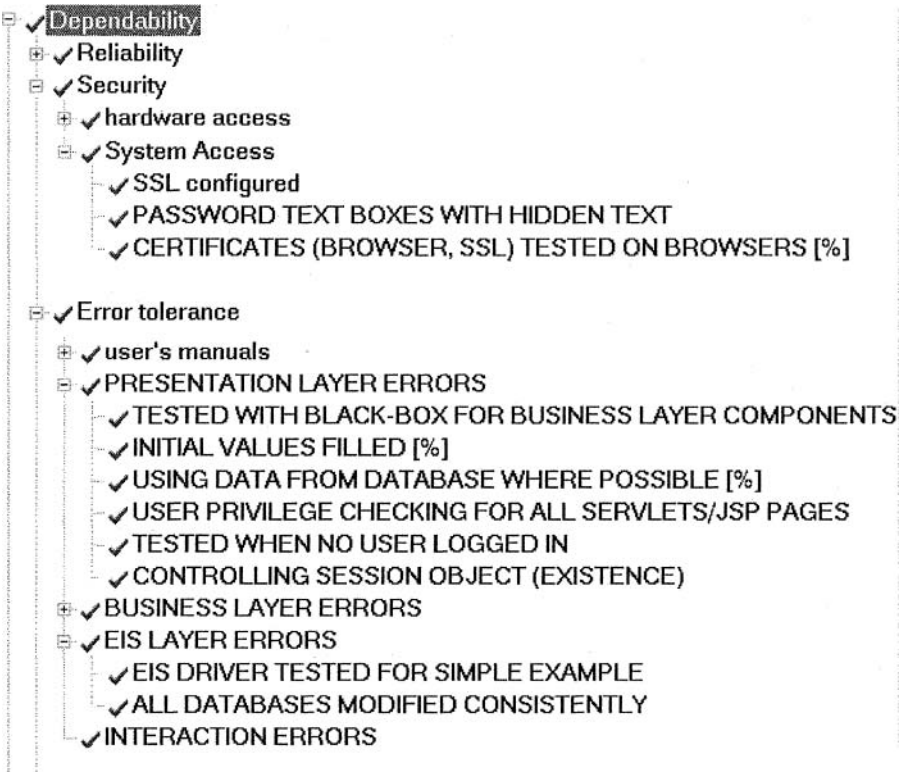

Fig. 3. Part of a QESA Quality Model for BeesyCluster's Testing

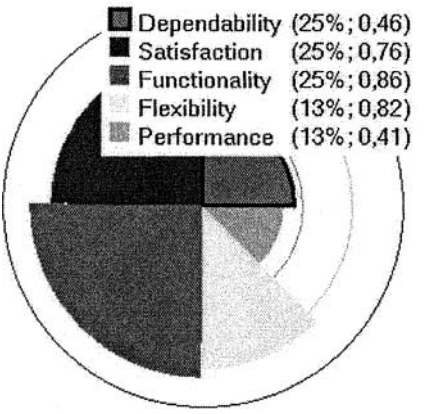

Fig. 4. Quality before Improvement

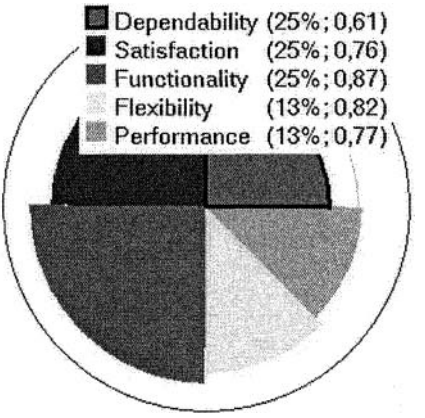

Fig. 5. Quality after Improvement

7 Summary

The model used, especially the quality issues specific for J2EE applications and identified above, can make design, implementation and development of other similar

applications easier and faster. Products for design, implementation and testing for other J2EE applications can be validated against items identified in this paper while QESA can produce a quantitative quality value which can be compared to other systems. One of the original goals of the QESA initiative was the creation of distinct models, including translation functions (how the values of lower level nodes are translated to higher levels), coefficients of translation functions, metrics and measures specific for the given application class and the given development phase. This is especially useful in case of distributed applications due to their complex nature. The model proposed in this work is based on real world errors encountered during the development of a large production J2EE-based application and can be either used as provided or improved.

References

1. Czarnul, P., Bajor, M., Banaszczyk, A., Buszkiewicz, P., Fiszer, M., Fraczak, M., Klawikowski, M., Rakiej, J., Ramczykowska, K., Suchcicki, K.: The architecture of beesycluster: a front-end to a collection of clusters accessible via www/web services. In: Proceedings of VI Conference on Computer Engineering (KKIO 2004), Gdansk, Poland (2004) 437–450 in Polish, ISBN 83-204-3051-8.
2. Czarnul, P., Bajor, M., Fraczak, M., Banaszczyk, A., Fiszer, M., Ramczykowska, K.: Remote task submission and publishing in beesycluster : Security and efficiency of web service interface. In Springer-Verlag, ed.: Proc. of PPAM 2005. Volume LNCS 3911., Poland (2005)
3. Czarnul, P.: Integration of compute-intensive tasks into scientific workflows in beesycluster. In: Proceedings of ICCS 2006 Conference,, University of Reading, UK, Springer Verlag (2006) Lecture Notes in Computer Science, LNCS 3993.
4. Sun Microsystems: BluePrints, Patterns (2006) http://java.sun.com/blueprints /patterns/index.html.
5. Alur, D., Crupi, J., Malks, D.: Core J2EE Patterns: Best Practices and Design Strategies. 2nd edn. Prentice Hall / Sun Microsystems Press (2003) http://www. corej2eepatterns.com/index.htm, ISBN:0131422464.
6. Sun Microsystems: Code Conventions for the JavaTM Programming Language (1999)
7. O'Hanley, J.: Collected java practices (2006) Canada, http://www.javapractices. com/Table-OfContents.cjp.
8. Fenton, N.: Ensuring quality and quality metrics. In: Software engineering. MIKOM (2000) ISBN 83-7279-028-0.
9. Emam, K.E., Drouin, J.N., Melo, W.: SPICE The Theory and Practice of Software Process Improvement and Capability Determination. Wiley (1997) ISBN 0-8186-7798-8.
10. Sedigh-Ali, S., Ghafoor, A., Paul, R.A.: Software engineering metrics for cots-based systems. IEEE Computer Society Press, Computer **34**(5) (2001) 44–50 ISSN:0018-9162.
11. Czarnul, P., Krawczyk, H., Mazurkiewicz, A.: Quality driven development methodology for network applications. In: ISThmus'2000 Conference, Poznan, Poland (2000)
12. Fredericks, M., Basili, V.: Using defect tracking and analysis to improve software quality. Technical report, Experimental Software Engineering Group, University of Maryland, College Park, Maryland USA (1998)
13. Czarnul, P., Grzeda, K.: Parallelization of electrophysiological phenomena in myocardium on large 32 & 64-bit linux clusters. In Springer-Verlag, ed.: Proceedings of Euro PVM/MPI 2004, 11th European PVM/MPI Users' Group Meeting. Volume LNCS 3241., Budapest, Hungary (2004) 234–241
14. Nilo Mitra, Ed.: SOAP Version 1.2 Part 0: Primer. W3C Recommendation. (2003) http://www.w3.org/TR/soap12-part0.