Building Robust Web-based Systems by Managing Exceptions Through Logging, Reporting and Analysis

SHARIL TUMIN
University of Bergen
IT Dept
P.O. Box 7800, 5020 Bergen
NORWAY
edpst@it.uib.no

SYLVIA ENCHEVA
Stord/Haugesund University College
Faculty of Technology, Business and Maritime Sciences
Bjørnsonsg. 45, 5528 Haugesund
NORWAY
sbe@hsh.no

Abstract: Software systems will never be free from bugs since dynamic operating environments will introduce operating errors. Deployed Web-based applications could be made more useful and robust by planning exceptions handling during development phase. Under deployment, errors should be log for reporting and analysis. Error messages should make sense to the users and should not expose the system internals. Exception handling needs to be carefully planned.

Key Words: Web-based systems, exception handlers, error management, sustainable deployment

1 Introduction

The robustness of a computer program is a measure of how well it handles error situations. It is a well known fact that all non trivial computer programs are not error free. These errors can be categorized under 1) logical, 2) invalid references, 3) limit overruns, 4) input parameters and 5) data format, occur during runtime of the computer programs which were not perceived in advance under modeling, designing and implementation processes.

In order to produce a useful piece of software, programmers need to assume certain operational conditions and its runtime environment. To perceive all eventualities in advance is impossible due to the dynamic of these environments. Under certain dynamic runtime conditions these design assumptions will be no longer valid, thus producing operational exceptions. This is especially the case for a Web-based application build on top of multi-layered, independent and loosely coupled sub-systems.

It was reported, that more then 90% of total software cost is devoted to software maintenance and evolution [3]. A computer program in active usage is very seldom static and acting alone. The dynamic of usage exposes 1) runtime errors, 2) unfulfilled requirements, 3) integration problems and 4) enhancement potentials. The usefulness of a computer program is its ability to adapt and evolve, which is also the criterion for sustainable deployment.

The strictly sequential processes of system analysis, design, development and deployment (the why, how, what and where) in the task of creating a piece of computer program is a myth. These working tasks are usually run in parallel and are not considered completed, not until that piece of software joins the millions like it in the virtual graveyards of disuse, non-functional and forgotten computer software.

Programming language theoreticians debated about formal forms and constructs, such that language implementers can provide programmers with a perfect tool for their trade of building computer programs. No such a tool after six decades of modern computing has ever materialized and probably will never be. A programming language taken as a tool should assist not resist the work of developing software regardless of size and degree of complexity.

A programming language for general programming work must be 1) simple, 2) assist programmers productivity, 3) supports software modularity and 4) promotes software maintainability. Simple means simplicity of its syntax and constructions. Programs written in such a language are easy to write and read, while there are no obscure feature and unsafe constructions. The language runtime environment supports rich set of standard and system interfacing library. New interfaces and modules can be
developed easily. The language supports modern features such as object-oriented programming styles and modular software development. Most importantly, the language must support effective exception handling mechanism, after all a piece of software is not considered to be complete as long as it is in used and no software is bug free.

Software developers working within an organization write computer programs to facilitate organizational business processes, constrain by the organizational policies, resources, infrastructure and culture. No software were build in isolation. New software must conform to its running environment and interface with the existing systems and users.

2 What is a System?

What is a system? Different people will have different answers to this question depending on theirs professions, for examples; Eco system, weather system, political system and computer system. A system can be either naturally occurring or engineered by humans. Within the context of this paper, a system is meant to be a Web based engineered system.

A system; 1) is an integral set of inter-operable components performing specific functions on data read from input interfaces and write to output interfaces, 2) these functions are working in a coordinated ways to provide capabilities to reach some predetermined goals, 3) these goals are services, facilitating actors in performing specific value-added activities, 4) these services are modeled, designed and implemented by a team of competent designers, 5) the implementation and the deployment are bounded by technological and operational environments, and 6) the operation, maintenance, and adaption for sustainable period of usefulness by a team of competent maintainers. A system performs useful work of transforming raw data or material to a value-added finished product. A very simple system can be represented as a single module as shown in Figure 1. A more complex system is usually composed of many cooperating modules. A system module \( W \) can be described using a triplet of non-empty sets; 1) data, control and material \( D \), 2) transformers \( T \), and 3) users \( U \). The elements of set \( D \) can be any of; 1) raw material, 2) raw data, 3) information and control, and 4) intermediate and final products. The users of a system (or sub-systems) \( U \) are; 1) co-routines, 2) co-processors, 3) independent modules, and 4) human users. The transformer set \( T \) contains; 1) tools, 2) functions, 3) procedures, and 4) services. A system with a label \( n \) at a level \( i \) can be described by,

\[
\mathcal{W}^i_n = \langle D, T, U \rangle^i_n, \text{ where } D, T, U \neq \emptyset.
\]

Therefore, a module without interactions with other modules is not considered as a part of a system. By this top-down definition, a system must have users to interact with through its input/output interfaces. During modeling and design phase, these three sets can contain conceptual elements. Implementation of a system means converting conceptual elements of a model into real elements in deployment.

The labeling schema promotes hierarchical system design methodology, for example \( \mathcal{W}^{1.2.3}_{authn} \) and \( \mathcal{W}^{1.2.4}_{authn} \) can be used to label sub-modules of \( \mathcal{W}^{1.2}_{authn} \) located at one level up in the hierarchy. The transformer set \( T \) of a module can contain other sub-modules or sub-systems. While \( U \) defines usage from top-down view, \( T \) decomposes usage from bottom-up. Sometimes it is not possible to specifically define the \( U \) in a \( \mathcal{W}^i_n \) and in this case the \( U \) is a universal set, for example when a module \( \mathcal{W}^i_n \) implements a low level general library routines.

Two interrelated, independent and loosely coupled modules within a system can behave as 1) client-server, and 2) peer-to-peer.

The client-server model describes the server as a resource or service provider and the client as a consumer. The server waits for the client. All communications are initiated by the clients. In its simplest form, a request is sent by the client and a response is sent back through the same communication channel as a reply. This type of communication pattern is called request-response.

There is no clear distinction of who is the client
and who is the server in the peer-to-peer. Peers are both providers and consumers of services and resources. All peers can act as a client, i.e. initiate connection, and a server, i.e. wait for connection, at the same time. In its simplest form, peers communicate by message-passing.

Modules communicate with each others to exchange information (data orand control). In client-server model a client gets (pull) data from a server. In the peer-to-peer model an initiating peer puts (push) data to another peer. Data is either push to the destination or pull from the source, thus can be duplicated. Data can be saved in a long term data-store. Stored data can be; 1) searched, 2) retrieved, 3) copied, 4) modified, and 5) deleted.

Information is represented by data. Data becomes information when it is meaningfully understood by the users, whether a human or a program. The structure of data is defined by its syntactic rules, while its meaning is understood by its semantic rules. Informations are needed for intelligent actions, for example making decisions.

3 Web-based Systems

Web-based systems depend their inceptions due to the advancement of Internet technologies. Web-based systems are based on the model of division of labor, responsibility and control for a common purpose.

![Figure 2: Web-based System Multitier Architecture](image)

There are three important functional attributes that made a system Web-based; 1) presentation, 2) processing, and 3) data-storages, implemented in a three levels multitier system architecture of corresponding tiers; 1) Presentation, 2) Logic, and 3) Data, see Figure 2. These three tiers are correspondingly deployed by; 1) Web server, 2) Application server, and 3) Database server. Software components in each tier are developed and maintained as independent software product. A Web-based system deployment is free to mix and match any technologies for each of the three tiers, correspondingly for example; 1) Apache [11], 2) PHP [8], and 3) MySQL [6]. This popular combination is normally deployed on a Linux server and is widely known as LAMP [4].

The Logic tier (applications) implements the business logic, information processing and the middleware [5]. The development of these applications follows the Model-View-Controller (MVC) [7] software architectural model, as shown in Figure 3.

![Figure 3: Model-View-Controller Process Flow](image)

The one most significant difference between a graphic windows application and a Web-based application is the reliance of the Internet of the later. Each of the tier can be deployed on a separate server. The Logic can be and is most often deployed as a group of loosely coupled software applications running on different hardware machines. Internetworking system components of separate and cooperating units introduces communication lag time on top of the processing lag time.

Let $\Delta_{appl}$ be the time taken for an application to respond to a user’s request (i.e $t_f - t_0$ of Figure 3), and $\Delta_{appl} = \Delta_{proc} + \Delta_{comm}$, where $\Delta_{proc}$ to be the total processing time and $\Delta_{comm}$ the total communication time for the whole application. In Figure 3, the gray bars represent the time periods needed for computing of eight processing instances at different stages, while
communication is assumed to be instantaneous (for simplicity). For a Web-based application, $\Delta_{\text{comm}} \geq 0$ always. The main goal of a Web-based application developers and implementors is to minimize $\Delta_{\text{appl}}$, especially when a work process within the application includes human users. For data intensive application, $\Delta_{\text{comm}} > \Delta_{\text{proc}}$ and for compute intensive application, $\Delta_{\text{proc}} > \Delta_{\text{comm}}$.

Earlier studies [1] reported that human users’ wait time threshold for a reply from a Web site before it becomes intolerable is between eight to ten seconds i.e. $8 < \Delta_{\text{appl}} < 10$ seconds. With the advancement of new and faster hardwares the tolerable wait time (TWT) had shrunk to between five to eight seconds [2]. Persons having access to broadband and faster PCs expect faster reply from Web sites i.e. $5 < \Delta_{\text{appl}} < 8$ seconds. It is important that Web-based applications are designed with the TWT in mind, since this constrain defines their practical usefulness to human users.

4 Exceptions in Applications

Simplistically speaking, an application map inputs to outputs. An application that does not interact with its external environment is not considered in this paper. A Web-based application has an internal environment and an external environment. A block of program has a start and a termination.

The internal environment is in the control of the program. During the course of its execution, the internal environment will remain consistent with its intended purpose. The internal environment changes its states during the running of the program. The external environment is beyond the control of the running program.

The only influence the program can exert to its external environment is through its outputs in direct relation to its inputs. A program will give identical execution traces if identical inputs are given to independent separate runs.

External environment to a program is never static. In order to write a piece of program, programmers need to make informed assumptions regarding the eventual external environment the program will be ultimately executed. Since the external environment is dynamically changing, run time conditions might negatively affect the executing program into an exceptional state and will halt prematurely.

Exception is something that occurs when the program does not know how to continue along its execution path due to an unforeseen or unplanned internal state. This condition can be introduced by logical errors in the programming, inputs error or unfulfilled assumptions. Exceptions can be caught and dealt with in which the contributing factors directly responsible to the fault situations can be resolved immediately or at a later time and saving the running program from aborting.

An exception can be described by the following set of equations, in which $T$ is data transformer, $f_{1n}$ is a function executing at a specific time $t_n$, and $D_m$ is an internal data state at a specific time $t_n$:

$$T = \{f_{10}, f_{11}, f_{12}, \ldots f_{1n}\}$$

$$f_{10}(D_{t0}) = D_{t1}$$

$$D_{t0} \Rightarrow (f_{10} \rightarrow f_{11} \rightarrow f_{12} \rightarrow \ldots \rightarrow f_{1n}) \Rightarrow D_{t0+n}$$

$$e_T : f_{10} \rightarrow f_{11}, D_{t0} \Rightarrow D_{t0+1}$$

The exception $e_T$ had occurred when the data transformer $T$ is not able to complete its transformation of $D_{t0}$ to $D_{t0+1}$.

5 Exception Handler

Different programming languages provide different constructs for handling exceptions, where some of them are summarized in Table 1.

Common to all the constructs is that an exception needs to be caught in order to be handled by the exception handler. A special code block within a program called a trap-block, $\tau()$ (e.g. try { ... }) will catch any exception should this happen within the block. A trap-block will be associated with one or more handler-block, $\epsilon()$ (e.g. catch { ... }).

Any exception happening outside of this trap-block will not be caught by the inner operating environment of the currently running procedure but instead will be passed to its outer operating environment, which is usually the operating environment of its invoking procedure.

Let a block of program $P = (P^a < P^b < P^c)$ stands for procedure $P^a$ invokes $P^b$ invokes $P^c$, where

$$P = \{f_{10}^a \rightarrow f_{11}^a \rightarrow f_{12}^b \rightarrow f_{t3}^c \rightarrow f_{t4}^c \rightarrow f_{t5}^c \rightarrow f_{t6}^c\}$$

and $e_{t3}^c : f_{t3}^c \Rightarrow f_{t4}^c$.

The program block $P$ goes into exception at $t_3$ when executing $f_{t3}^c$. If there is no trap-block declared in $P^c$, then the $e_{t3}^c$ will be passed to $P^b$. The $e_{t3}^c$ will then be passed to $P^a$ if no trap-block is declared in $P^b$. 


Table 1: Exception Handling Constructs

<table>
<thead>
<tr>
<th>Language</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smalltalk</td>
<td>`on: ExceptionClass do: [:ex</td>
</tr>
<tr>
<td>C++</td>
<td><code>try { ... } catch( Exception e ) { ... }</code></td>
</tr>
<tr>
<td>Java</td>
<td><code>try { ... } catch( ex_type_1 ex_id_1 ){ ... } catch( ex_type_2 ex_id_2 ){ ... } finally { ... }</code></td>
</tr>
<tr>
<td>Perl</td>
<td><code>eval { # try ... } or do { # catch ... }</code></td>
</tr>
<tr>
<td>Ruby</td>
<td><code>begin ... rescue SomeError rescue ... else ensure end</code></td>
</tr>
<tr>
<td>Python</td>
<td><code>try: ... except ex_type_1 : except ex_type_2 : except: # catch all else: # no exceptions finally # no exceptions ...</code></td>
</tr>
</tbody>
</table>

The exception will be passed to the operative system (OS), if no trap-block was declared in P.

The associated handler-block can empty and in this case the exception is caught and ignored. Such programming pattern will prevent a program block from aborting but is not very useful for managing error conditions. A handler-block can itself go into exception. The best way to handle such situations is by placing trap-block and handler-block at the outer most level of invocation.

To catch any eventual exception to the whole program block and also exception in the handler-block, $P^c$ can be covered by $\tau[]$ and $e[]$ as $P_c = \tau[P]e[f^c]$.

6 Exceptions Management

The management of exceptions within a Web-based application is crucial to its usefulness and robustness. We have seen in Figure 2 that a Web-based system is a multi-layers, multi-servers and distributive processing system and is dependent on the Internet technologies as the mean of communication between its distributed processing elements.

The different conditions for managing exceptions, within an operating environment of a Web-based system can be listed as:

- $e_{a}$— within a single name-space of a running procedure,
- $e_{b}$— across multiple name-spaces of multiple coroutines,
- $e_{\gamma}$— across multiple process-spaces of cooperative independent processes,
- $e_{\delta}$— across multiple loosely communicating servers.

Ultimately for human users, what is rendered on theirs Web-browser is all that matters. Exceptions originated from a Web-based application must be managed in order to ) ensure it usefulness and protect security.

The usefulness of a Web-based application under exception means; 1) providing meaningful error messages, 2) providing standardized Status-Code to Web clients [9] [10] and 3) providing timely responses. For human users, consistent and meaningful error messages are very useful in ensuring continual users’ confidence of the system. Users’ perception of system’s responsiveness is also important, the constrain set by the TWT (mentioned in Section 3) must be met. Exceptions occurring in the independent, cooperative and distributive sub-systems must not block the application into waiting state longer then the TWT.

Uncritical display of exception messages showing raw dump process stack is not only confusing for the users but it can also negatively contribute to weakening of system security. Process stack and data model raw messages are meant for system administrator as debugging information in the work to solve operational problems during deployment. Such raw system
error messages can contain critical detail information of the system which could be used by hackers to compromise system security.

Exceptions should be managed at the level in the application where users and application meet. In view of Figure 3, the exceptions should be managed by the **Control** of the MVC model. The **Control** communicates with the user Web-browser. In order to be displayed an error message must correlate with the original request from the user no matter what type of exception it was and where it was originated from. Exceptions happening in the lower sub-systems must be escalated to the higher enclosing systems and finally handled by the **Control**.

In the event of an exception the **Control** will do the following exception management tasks:

1. creates a unique timestamped ticket of the event,
2. saves in relation to the ticket, the error message from the exception,
3. saves in relation to the ticket, raw dump of process stack in to a database,
4. makes user friendly error message together with issue tickets,
5. alerts system administrators of the event,
6. presents the user with the user’s error message.

The exceptions are logged into a database where analysis of the errors and their patterns could be analyzed and solutions could be applied to eradicate the course of the problems.

7 Conclusion

Software applications and more recently Web-based applications will never be free from bugs. The dynamically changing operating environment makes it impossible to know all eventualities in advance. Inputs from human users and loosely couple natured of independently managed distributive cooperative subsystems can introduce data format incompatibility. Since error can not be predicted in advance and completely avoided then the next best thing to do is to manage them properly in order to contain their influence over the usability and robustness of the deployed applications. In order to be able to manage errors, eventual exception conditions need to be trapped by mean of programming language construct `try` (e.g. `try { ... }`) which catches exceptions and `catch` (e.g. `catch { ... }`) which handles the exceptions.

Modern programming languages have exception handling as their standard language constructs. To understand their usage conditions and to use these constructs properly to log, report and analyze errors is the first step for developers in order to provide users with robust, useful and sustainable Web-bases applications.

**References:**


