Measuring Software Reliability

From end user’s perspective

Mr. Saurabh Dhupkar
Mumbai, India, saurabh.dhupkar@yahoo.com

Abstract — In this author's opinion, following are the most important software quality attributes - Reliability, Usability, Performance and Availability. As said wisely, "Things that count most, can not be counted mostly." This paper tries to suggest how 'Reliability' can be measured from end user's perspective.

For any software system, end user is the most important actor. Measuring ‘Reliability’ is a way to quantify the usefulness of the software for and end user. Therefore, measuring Reliability from an end user's perspective is very important. Unfortunately, there doesn't seem to be much discussion about how much reliable the software is for end user.

Keywords— Software Reliability, Software Quality, Software Architecture, Software Engineering, Continuous Improvement

I. INTRODUCTION

In this author's opinion, a user will be simply repelled from the software system that is not reliable. Unlike Usability or Availability where user wants to use it, but can't. For example, end users are comfortable going to authorized dealer to buy something or even bill payment or prepaid recharge. If we are proposing an alternative, we have to make sure that it is simpler, trust worthy and should provide more features. That means, if our online prepaid recharge portal charges money but fails to activate the prepaid package selected by user, the user will avoid using online recharge option.

Therefore, Reliability of Software is utmost important in this author's opinion.

Currently, we have very good and clear processes setup to measure availability. It measures the percent of time the application was responding. However, mostly it is not equipped to measure correctness at every instance.

Therefore, we need to have some process to measure the Reliability so that we will be able to prioritize the issues, we will be able to measure / forecast the impact, quantify the 'Image' of our software in the minds of end users, study the trend and forecast the life expectancy of the software and identify the need for software retirement or decommissioning well enough in advance.

II. RELIABILITY

2.1. Definition

As per ‘IEEE Standard Glossary of Software Engineering Terminology’ approved on September 28,1990, following are the definitions

2.1.1. Software Reliability

Reliability is the ability of a system or component to perform its required functions under stated conditions for a specified period of time

2.1.2. Software Reliability Growth

The improvement in reliability that results from correction of faults.

Thus in short, Software Reliability is the probability of failure-free software operation for a specified period of time in a specified environment [1].
III. PERSPECTIVE

This measure is proposed completely from the ‘End User’s Perspective’. It measures the number of successful attempts out of total number of attempts made at production environment.

User chooses to use an application considering some particular activity in mind, which we call as a ‘Use Case’. So, in simple terms, user opens the application to perform some use case. For end user, it is one application and doesn’t care about number of components or third party systems involved in.

Therefore, any failure / error / surprise that user faces belongs to the application and not to the components. Therefore, in this author’s opinion, we can not simply get rid of our responsibilities by just saying it is a third party component and is not under our control. Selecting that third party component was our decision and we can not run away from the cost of the decision. It also doesn’t matter if we are accepting that responsibility or not, if fact it is our responsibility, even if we don’t want it.

Therefore, for complete satisfaction of end user, we have to work as one team and consider every component that works for our application.

This measure tries to quantify the ‘End User Satisfaction’ considering the fact that most of the users will not raise an incident or request unless they are obliged with. They will simply walk away and try for some other application, may be your competitor’s application.

IV. WHY TO MEASURE RELIABILITY ?

4.1. Quantification
Quantification makes visualization possible and gives the actual feel
It can provide a standard way to measure success rate of application and its usefulness for end user. It can also provide a shared vocabulary for technical as well as non-technical stakeholders. Instead of measuring application reliability by ‘number of crashes in last 10 years’, it can give real-time status for shorter period also. For analysing effectiveness, having a usable data in hand is extremely important. Therefore, ‘number of crashes in last 10 years’ doesn’t help much. For an end user, what matters more is the probability of successful transaction and that depends on so many factors. Therefore counting number of crashes will not be of much help in this case.

4.2. Perspective
It enables stakeholders to measure the usefulness of the system for an end user. End user being the most important actor for the software, it matters most what end user’s think about the software. Measuring reliability attempts to quantify the image of software in the end user’s mind.
To quantify the ‘image of software’, surveys can be conducted. However, even if all the end users participate in the surveys, they will answer the questions that are asked. On the other hand, measuring reliability gives you the right question to think about.

4.3. Wider Scope
It is assumed that software has passed all the tests and reliability measurement is no replacement for testing. Reliability measurement gives wider scope than testing or availability monitoring by logging and counting every request, which makes every corner of software accessible to reliability measurement irrespective of its priority.
Most of time, testing focuses based on business priority (which is not wrong). On the other hand reliability measurement assesses every use case irrespective of its business priority.

4.4. Unreported Failures
End users will not report an incident if it is not mandatory for them. If while using software, end user faces any error / exception, user may simply try with some other alternative and may opt for
services from competitor. Reliability measurement removes the dependency of incident logging by monitoring every incoming request.

If the area of incident is not critical or can be ignored, end user might not bother about reporting it and it may go unnoticed.

4.5. Analysis
Reliability measurement provides concrete and quantified data which can be analysed to discover some trends or hidden issues which mostly can not be seen.

4.6. Forecasting
Reliability measurement and trend analysis can also provide some estimate about reliability trend and enable stakeholders to take necessary actions well in advance. It also enables stakeholders for forecasting life expectancy of software and find cure as early as possible wherever required.

4.7. Vision and Mission
It helps to set S.M.A.R.T. goal for the software. Instead of just qualitatively improving quality, it provides actual case study to go through. It can provide more focused priority and can set desired target and also measure it. It enables stakeholders to set Key Performance Indicators.

If it can be measured by using scientific means, it also be improved by using engineering techniques.

4.8. Related Quality Attributes
Reliability Impact Vs Component in change can also be tracked to understand the complexity and modifiability of the components. Reliability is inversely proportional to complexity. i.e. higher the complexity, lower the reliability while reliability is directly proportional to modifiability. i.e. higher the modifiability, higher the reliability.

V. HOW TO MEASURE RELIABILITY

5.1. Unit
Reliability should be measured as percentage per unit time to make it effective.

In layman language, reliability is % of successful attempts out of total number of attempts in a unit time.

   e.g. – Consider, your app is getting average 100 hits per min, out of which average 97 are successful and 3 end up in error. That means, reliability of your app in 97%.

Unit time can differ with scenario. E.g. – for a 24 * 7 app, the unit time can be hour, for an app with service hours of 8:00 – 18:00, unit time can be day, while for a real time system, unit time can be minute. It is stakeholders’ responsibility to wisely select the appropriate unit time.

Use of percentage, keeps the old values valid even if amount of traffic changes with time. e.g. – if 2 years ago, the app used to get 100 hits per hour, but now it is getting 1000 hits per min. However, if you are calculating reliability in percent, you can still compare with old data for trend analysis. On the other hand, if reliability is being measured as no. or failures in unit time, it may become irrelevant with time.

5.2. Inclusion
What is 'Reliability' from the end user's perspective? The software should every time complete the task given to it completely, on time and without any problems / failures / surprises and should give proper, adequate complete and timely response.

Thus, the software reliability also includes Responsiveness, Transaction Atomicity and Fault Tolerance. That also indicates that overall Reliability of software is combination of reliabilities of all the functionalities considering their relations with each other.
5.3. **Assumptions**  
This metric is to measure the real-life software and not a Software testing methodology. Therefore, the software is assumed to have passed all the testing methods satisfactorily and has gone live for end users.

5.4. **Basic**  
In simplest way, a reliability can be calculated as –

\[
\text{Reliability} = \frac{\text{No. of successful attempts}}{\text{Total no. of attempts}} \text{ unit time}
\]

Thus, this metrics will use real-world data from production servers. That also indicates that this value is volatile and changes with time. Every single change deployed at production environment will have some impact on the reliability of the software.

**VI. USE CASES**

For an end user, a software is a group of inter-related use cases. Therefore, measuring a software reliability according to use cases makes it effective and useful. Before attempting to measure reliability, finalizing the components in necessary.

6.1. **Function**  
A function is considered to be a complete, atomic and standalone group of activities which performs some business logic. 

- e.g. – Login, make payment etc.
- A use case is a series of functions executed to fulfill the user requirements. 
  - e.g. – Use Case – Bill Payment is a series of following functions –
    - Login
    - Select the bill amount
    - Select the payment method
    - Payment
    - Refresh user account

6.2. **Sub-Function**  
A sub-function is considered to be group of activities which perform one or more tasks for the function in coordination with other sub-functions and may or may not be standalone.  

- e.g. – LOGIN function can have following sub-functions –
  - VERIFY USERID AND PASSWORD
  - VERIFY CAPTCHA
  - VERIFY ROLES ASSIGNED TO THE USER

6.3. **Structure of a Function**  
Considering a function or a sub-function as a ‘Black Box’, this is how a function or a sub-function should be –
In this structure of function or a sub-function, every instance of output will be considered as a successful attempt, while every instance of error will be considered as failed attempt.  

e.g. – Successful login or ‘Invalid Credentials’ both will be successful attempts as function didn’t experience any failure. However, any unhandled error or exception will be considered as failed attempt.

This structure enables architects and developers to enable fault tolerance. As shown in Figure 2, in case if error thrown by sub-function 1 and sub-function 2, the sub-function 3 can be used to still try to provide the successful result.  

e.g. – if real-time request to a third party interface failed, it can be added in a request queue and process when the interface is up and running again

**VII. CALCULATIONS**

Thus, going back to calculations again –

\[
\text{Reliability} = \frac{\text{No. of successful attempts}}{\text{Total no. of attempts}}
\]

\[
\text{Reliability} = \frac{\text{No. of successful attempts for all sub-functions}}{\text{Total no. of attempts for all sub-functions}}
\]
Overall reliability of a function depends on reliabilities of sub-functions and their internal relations. Broadly, following are the types of relations between sub-functions:

### 7.1. Sequential

In this type of relation, sub-function2 gets executed after successful completion of sub-function1. If, execution of Sub-function1 fails, sub-function2 will not get executed at all.

In short, the function is successful only if both the sub-functions are successful.

\[ R = R(1) \times R(2) \]

![Sequential Sub-Functions](image)

### 7.2. Conditional

In this type of relation, either of the sub-functions get executed but not both. Based on some condition, any one of the sub-function is called. Therefore, the success of function depends on success of either of sub-functions.

![Conditional Sub-Functions](image)

\[ R = \sum \left( \frac{T_i}{T} \times R_i \right) \]

Where
- \( T_i \) = Number of requests for that sub-function
- \( T \) = Total Number of requests
- \( R \) = Reliabilities of respective sub-functions
7.3. **Fault Tolerant**

In this type of relation, one sub-function is meant to work as ‘Backup Plan’ or ‘Plan B’ for other sub-function. Its main role is to apply workaround or handle the exception cases of main sub-function.

![Fault Tolerant Sub-Function](image)

In this type of relation, the overall reliability will be –

\[
\text{Reliability} = R(1) + (1 - R(1)) \times R(2)
\]

7.4. **Example**

To elaborate more, let us go through following example of Login Use Case –

![Example for Reliability Calculation](image)
In this example –

\[ R = R(1) \times R(2) \times R(3) \]

Where
R(1) – Reliability of Verify UserID and Password
R(2) – Reliability of Verify CAPTCHA
R(3) – Reliability of Authorization

\[ R(3) = R(4) + \left( (1 - R(4)) \times (R(5)) \right) \]

Where
R(4) – Reliability of group of fetch roles, showAdminPage, showStandardPage
R(5) – Reliability of showMinimalPage

\[ R(4) = R(6) \times \left( \frac{T_7}{T} \times R(7) + \frac{T_8}{T} \times R(8) \right) \]

Where
R(6) – Reliability of group of fetch roles
T(7) – No. of requests to showAdminPage
R(7) – Reliability of showAdminPage
T(8) – No. of requests to showStandardPage
R(8) – Reliability of showStandardPage
T = T_7 + T_8

Therefore, combined formula will be -

\[ R = R(1) \times R(2) \times \left( \left( R(6) \times \left( \frac{T_7}{T} \times R(7) + \frac{T_8}{T} \times R(8) \right) \right) \right) \times \left( \left( 1 - \left( R(6) \times \left( \frac{T_7}{T} \times R(7) + \frac{T_8}{T} \times R(8) \right) \right) \right) \right) \times \left( R(5) \right) \]

Let us assume some values to complete the example –

<table>
<thead>
<tr>
<th>No. of input requests (min)</th>
<th>Function</th>
<th>No. of Successful requests (min)</th>
<th>No. of Failed requests (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1-‘verify userid and password’</td>
<td>996</td>
<td>4</td>
</tr>
<tr>
<td>996</td>
<td>2-‘verify captcha’</td>
<td>990</td>
<td>6</td>
</tr>
<tr>
<td>990</td>
<td>6 - ‘fetch user Roles’</td>
<td>975</td>
<td>15</td>
</tr>
<tr>
<td>900</td>
<td>8 - showStandardPage</td>
<td>897</td>
<td>3</td>
</tr>
<tr>
<td>75</td>
<td>7 – showAdminPage</td>
<td>72</td>
<td>3</td>
</tr>
<tr>
<td>21</td>
<td>5 - ‘show minimal page’</td>
<td>15</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 1: Example figures for Reliability Calculation Example
Therefore,

\[ R(4) = 98.5\% \times \left( \left( \frac{75}{975} \times 96\% \right) + \left( \frac{900}{975} \times 99.7\% \right) \right) \]

\[ R(4) = 98.5\% \times (0.0769 \times 96\%) + (0.9230 \times 99.7\%) \]

\[ R(4) = 98.5\% \times (7.38\% + 92.02\%) \]

\[ R(4) = 98.5\% \times 99.4\% \]

\[ R(4) = 97.9\% \]

Let us calculate Reliability of Authorization module as -

\[ R(3) = R(4) + \left( (1 - R(4)) \times (R(5)) \right) \]

\[ R(3) = 97.9\% + ((1 - 97.9\%) \times (71.4\%)) \]

\[ R(3) = 97.9\% + (0.021 \times (71.4\%)) \]

\[ R(3) = 97.9\% + (1.50\%) \]

\[ R(3) = 99.4\% \]

Thus, overall Reliability will be -

\[ R = 99.6\% \times 99.4\% \times 99.4\% \]

\[ R = 98.4\% \]

That means, in a unit time if 1000 requests come, there will be 984 successful requests. To crosscheck with the original values from table –
Final successful requests = showStandardPage + showAdminPage + show minimal page  
Final successful requests = 897 + 72 + 15 = 984

Thus, with these figures, overall reliability of application can be calculated and studied.

VIII. POSSIBLE TREND ANALYSIS

From the captured data, many types of matrices can be prepared and studied. Mentioning some examples here as –

8.1. Basic Trend
Basic trend represents simple graph of reliability over longer period.  
e.g. – if reliability is being measured per minute, then reliability of larger period will be average of reliabilities of all the minutes in the period. Therefore, basic trend of year will be graph of reliabilities of all minutes in the year or the average reliability per day in the year.

8.2. Reliability Vs Change
It is a simple bar graph showing average reliability in period between two consecutive changes. This can be used to measure impact of change(s) and can also be used to set expectations and vision for future changes.

8.3. Reliability Vs Load
It can be a simple line graph showing average reliability percent with respect to number of incoming requests. It can be used to check if number of incoming requests are having any impact on reliability. i.e. – if it is observed that as load on application increases, its reliability decreases, then it gives an improvement scope.

8.4. Incomplete Use Case
In this type of metric, the sales team can study the number of requests which were aborted by user even though there were no failures.  
e.g. – a user selected some items and pushed them in shopping kart, but didn’t checkout and simply left without completing the transaction.  
In this kind of cases, teams can study the areas where maximum users leave the flow without any failure experience. This metric can put light on other software quality attributes (like usability, performance etc.) and also business cases where user might feel that some competitor has better offers etc.

IX. LIMITATIONS

- It is not just some another matrix calculating successful requests. It demands complete attention and brainstorming.
- If not understood and followed religiously, just like all the other processes this too will become meaningless and will yield no output.
- Trend analysis is crucial activity. Just following SLA per cycle will not help.
- Setting proper checkpoints as the key of success / failure of this measurement.
- How to implement it technically differs with technology and architecture.
- It is trickier to implement for already running application than to those which include it in design phase itself.
- UI failures due to client side scripting bugs may go unnoticed if not handled properly.
For complete standalone client side applications, gathering all data from users all over the world can be complicated.

X. RELIABILITY MANAGEMENT

Thus Reliability management has following four phases –

10.1. Phases of Reliability Management

10.1.1. Monitor
Includes logging every request and its result, so that it can be later identified if the attempt was successful or failed.

10.1.2. Measure
Includes calculation of reliability based of internal relations formulas.

10.1.3. Analyze
Based on reliability value obtained per unit time, analyzing the problems and forwarding them to Incident Management if found something.

10.1.4. Forecast
Includes trend analysis and deriving different suitable metrics and forecasting reliability and proposing appropriate actions wherever required.

Based on these four phases, Reliability Management can be of two types –

10.2. Types of Reliability Management

10.2.1. Reactive Reliability Management
In the above proposed Reliability Management, measuring reliability by given formulas and taking actions based on is part of Reactive Reliability Management.

As in this part technical teams will be studying the past data and analyze the unexpected behaviors from technical as well as business point of view. This kind of study can reveal some unrecorded incidents, where user didn’t bother to raise a complaint when experienced some failure.

Users may not always answer to all the surveys that companies put forth. However, if teams can study actual behavioral patterns, they can discover some issues or come up with some improvement ideas for weak links.

10.2.2. Pro-Active Reliability Management

Studying reliability trends and forecasting issues / problems and forecasting life expectancy of the software is part of Proactive Reliability Management.

In this case, technical teams try to forecast based on past data and can prepare for future.

e.g. – When a threshold is set for reliability saying that reliability should never fall below this particular value, if trend analysis shows that based on impact of every change, may be after some particular number of changes deployed to production, reliability might cross the threshold and drop below accepted limit. The technical teams and study the bottleneck areas, dependencies and work on weak links. Instead of applying some last minute patches, if issues can be forecasted well in advance, they can be managed better.

XI. CONCLUSION

Measuring and studying reliability and its trends in application can reveal things which testing might not discover. For better end user satisfaction, instead of simply relying on incident and problem management, reliability management can discover more areas of improvement. Reliability management gives an opportunity to see the application from the end user’s perspective and improve the applications usefulness by continuous monitoring, understanding and forecasting.
REFERENCES