Scientific Journal Impact Factor (SJIF) : 3.518
IJMTER
International Journal of Modern Trends in Engineering and Research
ISSN (ONLINE) : 2349 - 9745
ISSN (PRINT) : 2393 - 8161

MEASURING SOFTWARE LIFE EXPECTANCY

Saurabh Dhupkar

Abstract—This Any software generally goes roughly through following phases in its life

1. Birth
2. Learning
3. Earning
4. Retiring

From the business perspective, the software that stays in earning phase for longer time, can roughly be considered as a successful software (well, there can be some exceptions). Keeping this in mind, everyone who works with the software at any point of time in the life of software (irrespective of the role and responsibilities) should focus on working in such a way that software will stay in earning phase as long as possible. Even though we accept the fact that no software can stay in earning phase forever, it can certainly be tried to manage to provide optimum result. Everyone who works with the software at any point of time in any role leaves a mark on the software and contributes to increase or decrease in the overall life expectancy of the software. Therefore, this paper tries to study the software life expectancy, factors affecting it, its relation with different software quality attributes and processes to improve the software life.

Keywords—Software Quality; Software Quality Attributes; Measuring Software Quality Attributes; Software Life Expectancy;

I. INTRODUCTION

The entire life of software can be mainly categorized in four phases

1. Birth –
   - This is the part of life of Software, where software is taking shape conceptually.
   - It is still an abstract entity in this phase and no coding is actually started.
2. Learning –
   - This is the part where actual software development is done.
   - In layman language, the actual software can be 'seen' in this phase.
   - All the development and testing are done in this phase till releasing the software for end users.
3. Earning –
   - In this phase, the software actually works for end users.
4. Retirement –
   - In this phase, another software(s) starts taking over the responsibilities from this software.
   - At the end of this phase, the software is completely decommissioned.

Generally, in first and second phase we invest time and efforts in the software and expect the returns in third phase

Thus, the third phase plays most significant role from business perspective. The returns estimated from third phase are motivating stakeholders and techies to invest time, efforts and money in the software in its first and second phase.

Thus, it can be roughly said that overall success of the software depends upon it's overall performance in third phase as well as the duration of the third phase. In general, it can be said that the duration of third phase is directly proportional to the quality of software. That means, the overall quality of software plays vital role in deciding the life expectancy of the software.

II. SOFTWARE LIFE EXPECTANCY

In layman language, software life expectancy can be stated as overall duration the software stays usable.

Alternatively, the duration till critical software quality attributes don't fall below a threshold point of no return.

However, in this author's opinion Software life expectancy can be defined as Estimated duration the software maintains it's required quality attributes in acceptable range assuming it won't undergo any more changes, where required quality attributes include common critical quality attributes and special critical quality attributes.

2.1. Common Critical Quality Attributes

These are the basic quality attributes that each software must maintain to exist -
There are four common quality attributes

1. Reliability
2. Usability
3. Availability
4. Performance

2.1.1. Reliability

As per ‘IEEE Standard Glossary of Software Engineering Terminology’ approved on September 28, 1990 Software Reliability is defined as – The ability of the system or component to perform its required functions under stated conditions for a specified period.

If software is not reliable, i.e. it is not able to perform required functions, then user will avoid using it and eventually software will cease to exist. Therefore, in this author’s opinion Reliability can be considered as a Critical Software Quality Attribute.

2.1.2. Usability

As per definition by ISO, Software Usability is the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use.

In short, Software Usability deals with ease-of-use of the software. Even though the software is completely reliable, if the end user is not able to understand or use the software, users will be forced to look for alternatives. Therefore, Software Usability can be considered as a Critical Software Quality Attribute.

2.1.3. Performance

For a layman, Software performance is average amount of work completed by software per resource per unit time. In short, better performance indicates more amount of work in less time using least possible resources.

Thus, Software Performance is –

- inversely proportional to average time taken to complete a task
- inversely proportional to average resources utilized to complete a task

2.1.4. Availability

Availability is the ratio of time a system or component is functional to the total time it is required or expected to function.

Low availability indicates frequent failures and longer downtimes. If the software is not capable to work when expected by users, they are forced to look for alternatives. And later on, the alternatives replace the original software

2.2. Special Critical Quality Attributes

Special quality attributes are those quality attributes which a software must maintain to exist apart from common critical quality attributes depending upon the requirements.
e.g. – Maintainability, Modularity, Modifiability, Fault Tolerance, Atomicity etc.

The importance of these attributes depend upon the context and functionality of the software. The stakeholders are supposed to select the appropriate software quality attribute based on the type of software and domain.

III. SOFTWARE LIFE EXPECTANCY Vs QUALITY ATTRIBUTES

The changes in levels of software quality attributes mentioned above leads to changes in software life expectancy.
e.g. – Improvement in performance leads to increase in software life expectancy, while decrease in usability leads to decrease in software life expectancy.

Depending on importance of each critical quality attributes, the impact in life expectancy changes.
i.e. – Higher the weightage, higher the impact.
e.g. – Poor software modifiability, modularity or maintainability will have less impact on software life expectancy than poor performance or usability.

$$Software\ Life\ Expectancy \propto \int (w_i * Q_i)$$

Equation 1 : Relation between Software Life Expectancy Vs Software Quality Attribute weightage and value

Where
Wi – Weightage to the quality attribute
Qi – Value of the quality attribute

Therefore, to keep the software life expectancy higher, the level of software quality attributes needs to be maintained at high levels

3.1. Primary Reasons behind loss of Software Quality

There are various different reasons behind loss of quality attributes. Following are the most common reasons –

1. Unclear or Unrealistic Vision
    a. Vague , unrealistic requirements
    b. Volatility of requirements
    c. Conflicting views
    d. Unattainable timeframe
2. Hurdles in Communication
    a. Language barriers
    b. Geographical differences in language
3. Inadequate Processes
    a. Inadequate timeframe for testing
    b. Unclear or vague test cases
    c. Difference in requirements interpretation resulting in confusions
    d. Less capable version control
    e. Incomplete , vague documentation and logs
4. Human Factors
    a. Unavailability of necessary skill sets at necessary skill levels
    b. Human resources issues

3.2. Developer’s Debt

The developer’s debt is a primary reason, which has ever-increasing impact on software quality and software life expectancy.

Developer’s debt represent a quick fix applied to a problem to ‘save the day’ instead of implementing proper solution.

The quick fixes applied can have various impacts on the software based on where they are applied and how the quick fix is implemented. As the number of such 'quick fixes' live in the software code increase, it starts impacting software quality attributes. These quick fixes are called as 'Developer's Debt' because longer they stay, more efforts are needed to settle them.

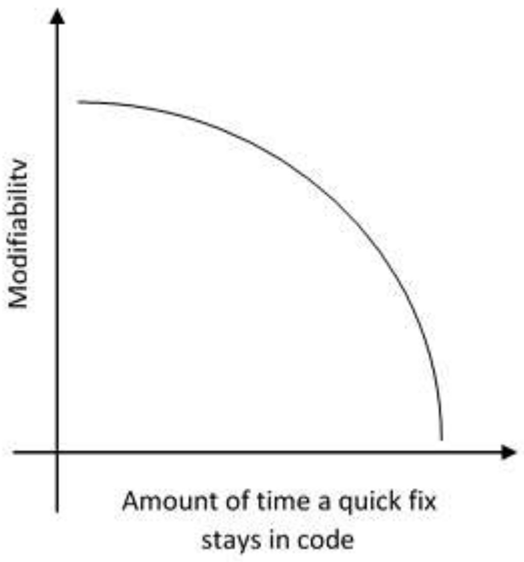

Figure 1. Impact on Software Modifiability by Developer's Debt

The above graph shows general impact of a 'Developer's Debt' on software modifiability.
i.e. Longer the 'quick fix' stays in code, it reduces Modifiability.

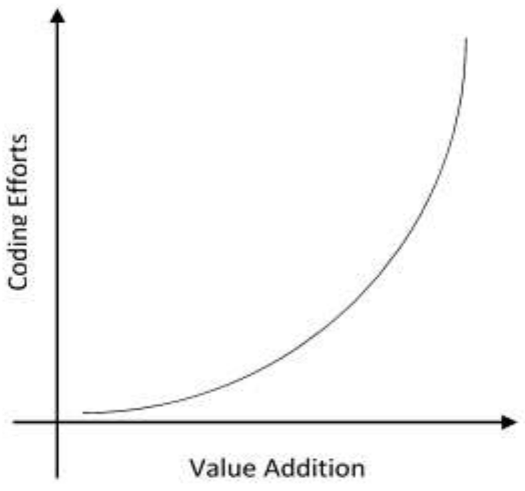

Figure 2. Impact of Developer's Debt on Value Addition

That means, for every new value addition to the code, the amount of efforts required goes on increasing. As a particular point, value addition to the software becomes too time consuming and expensive, that it becomes practically impossible.

With the decrease in Modifiability, Maintainability decreases, which leads to decrease in Performance and Availability and finally it impacts Reliability and Usability.

IV. CALCULATE SOFTWARE LIFE EXPECTANCY

4.1. Steps to quantify Software Life Expectancy

The For quantification of Software Life Expectancy, following are the steps –

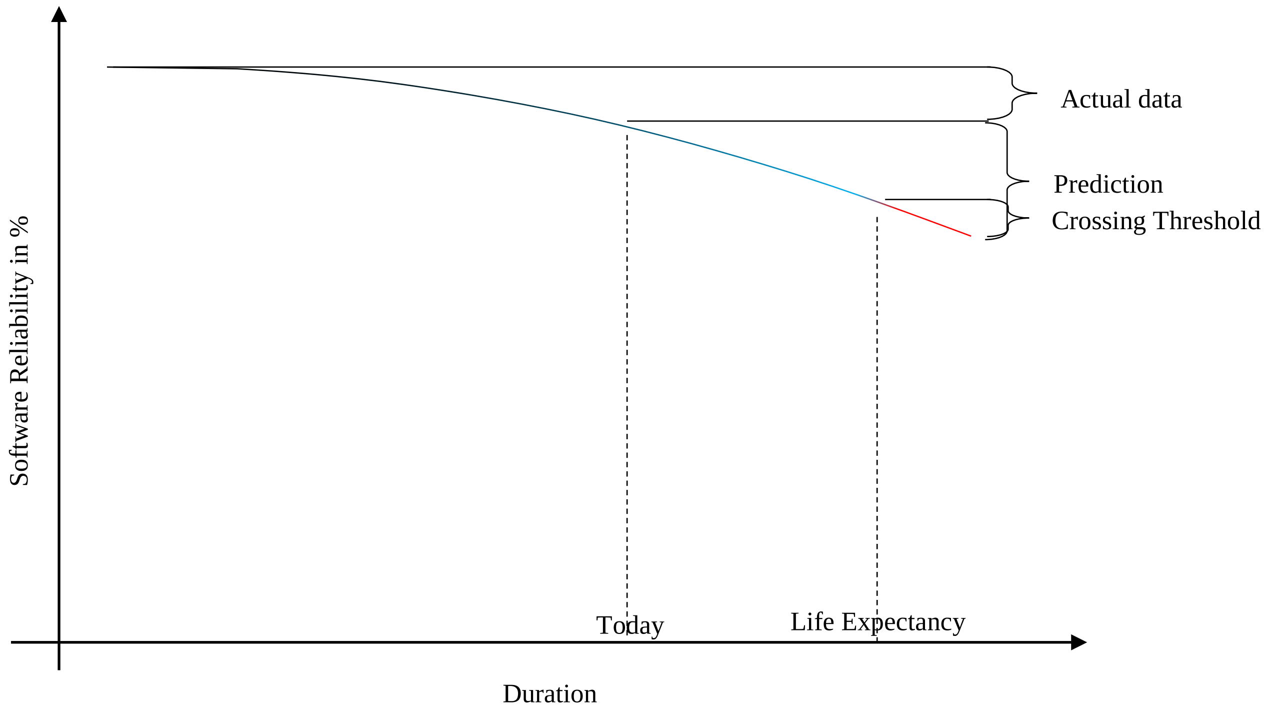

Figure 3. Example trend analysis of Software Reliability for Software Life Expectancy

5. Define all the ‘Critical Quality Attributes’
6. Define the threshold value for all the ‘Critical Quality Attributes’ in suitable units, which indicate that the value of the quality attribute must not cross over the threshold.
7. Define a unit time for calculating quality attributes.
8. Monitor the value of all defined ‘Critical Quality Attributes’.
9. Calculate the average rate of change per unit time for all the defined ‘Critical Quality Attributes’.
10. Perform the trend analysis and forecast the time when all the Critical Quality Attributes reach the pre-defined threshold.
11. Compare and identify the first Critical Quality Attribute to cross the respective threshold.
12. Compare the actual values obtained from monitoring with the one forecasted for this iteration instance
13. Repeat steps 4 to 8
14. The pre-defined threshold represent the business requirements. Therefore, the threshold values can change over the period.

The target for the trend analysis is to forecast the values of the critical quality attribute under analysis to estimate the duration for it to reach the threshold value. Statistical analysis should be done based on one or more change factors and their average values from past.

Accuracy of forecasting depends upon thee statistical data and analysis. Thus, better the knowledge management better will be the forecasting.

This process comes under ‘Proactive Maintenance’

The graph below shows an example trend analysis of ‘Software Reliability’ and its relation with Software Life Expectancy.

If the software doesn’t undergo any refactoring or renovation, an estimated duration to reach threshold is Software Life Expectancy.

In case of multiple Critical Quality Attributes, the one that reaches the respective threshold first, plots the life expectancy.

The Quality Attributes with higher weightage should have more stringent threshold. Every quality attribute to be calculated using its own units.

4.2. Example

For explanatory purpose, 'Performance' is the only 'Critical Software Quality Attribute' is defined for a dummy web application.
The 'Response Time' represents 'Performance' in this case with unit time is specified as a 'quarter'.

Table 1. Example average response time of dummy web application and its trend analysis

Measurement Instance	Actual Response Time (millisecond)	Predicted Response Time (millisecond)
2014-01	244	
2014-04	252	
2014-07	263	
2014-10	295	
2015-01	305	
2015-04	326	
2015-07	334	
2015-10	346	
2016-01	359	
2016-04	385	
2016-07	398	400.7
2016-10		413.4
2017-01		428.8
2017-04		444.2
2017-07		459.6
2017-10		475.0

Measured and forecasted values can be plotted to graph as -

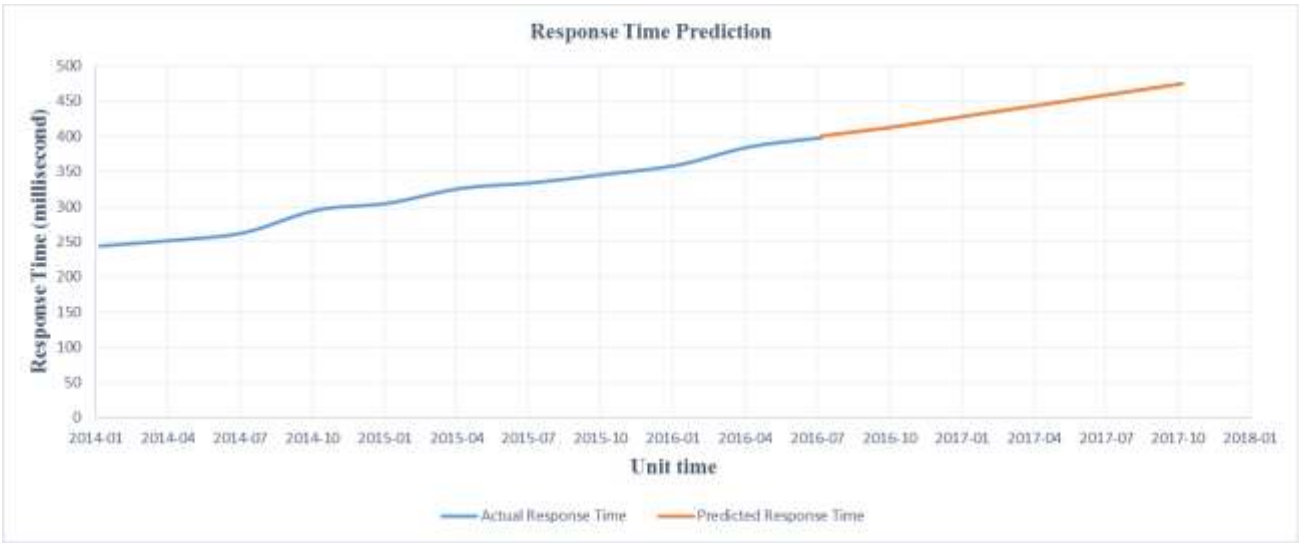

Figure 4. Trend analysis of actual 'Performance' and forecast

If the threshold is set to be 500 millisecond, it can be predicted that application will cross the threshold in second quarter of 2018. That gives application stakeholders 7 quarters to prepare and act well in advance.

This also shows the comparison between value forecasted for 2016-07 and actual value measured for that period.

4.3. Advantages

1. Teams can start application refactoring or renovation activities well in advance.
2. As the calculation shows the quality factor trends, teams can focus on right place.
3. Quantification and trend analysis can be used to set up SMART goals and KPIs and can be tracked and verified.
4. Software retirement can be planned well in advance. This can minimize or reduce the business impact

4.4. Disadvantages

1. Not all Software Quality Attributes can be measured. E.g. – Usability
2. Based on software context, some of the quality attributes can not be measured in standard way. Each software may need to calculate value of some of the quality attributes their way.
3. Software Quality Attributes are independent of business priorities. Therefore, areas in node pointed out by these software quality attributes may not be aligned with business priorities.

V. CONCLUSION

Software quality attributes play extremely vital role in success or failure of the software. Quantification and trend analysis of software quality attributes based on relevant factors can provide the areas to focus on. This paper discusses about relation between the software quality attributes and success of software

VI. REFERENCES

[1] M. Rouse, "Reliability, Availability and Serviceability (RAS)," March 2011. [Online]. Available: http://whatis.techtarget.com/definition/Reliability-Availability-and-Serviceability-RAS.

[2] "IEEE Standard Glossary of Software Engineering Terminology," 28 September 1990. [Online]. Available: http://www.mit.jyu.fi/ope/kurssit/TIES462/Materiaalit/IEEE_SoftwareEngGlossary.pdf.

[3] "Usability," [Online]. Available: https://en.wikipedia.org/wiki/Usability.

[4] S. Dhupkar, "Measuring Software Reliability," *International Journal of Current Trends in Engineering & Research (IJCTER)*, vol. 2, no. 7, pp. 70-81, 2016.

[5] S. Dhupkar, "Biography of Software," April 2016. [Online]. Available: http://www.academia.edu/26242659/Biography_of_Software.

[6] S. Grenier, "Developer Debt," 30 09 2007. [Online]. Available: http://www.followsteph.com/2007/09/30/developer-debt/.

[7] "Technical debt," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Technical_debt.

[8] "List of system quality attributes," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/List_of_system_quality_attributes.

[9] "FURPS," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/FURPS.

[10] "Reliability engineering," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Reliability_engineering.