Effect of Residual Defect Density on Software Release Management

Dr. Rachna Soni¹, Dr. Ashish Jolly², Arti Rana³

¹Department of Computer Science, DAV Girls College, Yamuna Nagar (Haryana), India.
²Department of Computer Science & Engineering, Chitkara University, Rajpura (Punjab) India.
³Amity Institute of Information Technology, Amity University, Noida (UP) India.

rsoni63@gmail.com, ashish_jolly_79@yahoo.co.in, artirana@yahoo.com

Abstract

In Global markets the competition has increased dramatically. This has resulted on the need for software development firms to produce at a lower cost, with higher quality and within shorter time frames. The focus must clearly be put on the customer and the objective must not simply be to satisfy, but to delight. This can only be accomplished by providing the right system and executing the pertinent project(s) in the right way. In industry, information on defect density of a product tends to become available too late in the software development process to affordably guide corrective actions. An important step towards remediation of the problem associated with this late information lies in the ability to provide an early estimation of defect density. The residual defect density of a software product can often only be estimated, based on the number of user complaints. The number of complaints does not just depend on the residual defect density, it also depends on the number of users, and the amount and duration of actual usage. The identification and removal of software defects constitutes the basis of the software testing process, a fact that inevitably places increased emphasis on defect related software measurements. The stochastic parameters of the proposed system with specific system boundaries under a given environment have been estimated using simulation.

Keywords: Defect Density, Software Release Management, Residual Defects.

1. Introduction

Residual defects are one of the most important factors that allow one to decide if a piece of software is ready to be released. In theory, one can find all the defects and count them, however it is impossible to find all the defects within a reasonable amount of time. Estimating defect density can become difficult for high reliability software, since remaining defects can be extremely hard to test for [5][6][7].

In industry, post-release defect density of a software system cannot be measured until the system has been put into production and has been used extensively by end users. The defect density of a software system is calculated by measuring the number of failures divided by the size of the system, using a size measure such as lines of code. Actual post-release defect density information becomes available too late in the software lifecycle to affordably guide corrective actions to software quality [1]. Correcting software defects is significantly more
expensive when the defects are discovered by an end user compared with earlier in the development process [2]. The latest Special Analysis Report on Software Defect Density from the ISBSG reveals useful information about defects in software, both in development and in the initial period after a system has gone into operation [4]:

- The split of where defects are found, i.e. in development or in operation, seems to follow the 80:20 rule. Roughly 80% of defects are found during development, leaving 20% to be found in the first weeks of systems’ operation.
- Fortunately, in the case of extreme defects, less than 2.5% were found in the first weeks of systems operation.
- Extreme defects make up only 2% of the defects found in the Build, Test and Implement tasks of software development.
- The industry hasn’t improved over time. Software defect densities show no changing trend over the last 15 years.

The report also provides useful ratios of when and where defects are discovered and the severity of those defects.

2. Defect Potentials, Removal Efficiencies and Defect Densities

- In software, the narrowest sense of product quality is commonly recognized as a lack of defects or ‘bugs’ in the product. Using this viewpoint, or scope, three important measures of software quality are [3]:
  - Defect potential, defined as the number of injected defects in software systems, per size attribute.
  - Defect removal efficiency, defined as the percentage of injected defects found and removed before releasing the software to intended users.
  - Defect density, defined as the number of released defects in the software, per size attribute.
- As software grows, defect potential increases and defect removal efficiencies decrease. The defect density at release time increases and more defects are released to the end-user(s) of the software product. Larger software size increases the complexity of software and thereby the likelihood that more defects will be injected. For testing, a larger software size has two consequences:
  - The number of tests required achieving a given level of test coverage increases exponentially with software size.
  - The time to find and remove a defect first increases linearly and then grows exponentially with software size.
- As software size grows, to just maintain existing levels of released defect densities, software manufacturers would have to exponentially improve their defect potentials and removal efficiencies.
3. The Price of Defects

Not every defect is equally damaging, and it may be a bit too simplistic to consider only the average cost associated with bugs, there is no real hard data to fall back on here for an assessment of how the severity of bugs correlates with bug density, but we can formulate a hypothesis. West [13] classifies defects into levels based on the number of independent factors that are jointly required to cause their occurrence. In this classification, a defect triggered by a single cause is called a defect of level one. A defect of level two has two independent causes that must occur in a particular combination. The first cause could be the failure of a standard routine (“cannot write – disk full”), and the second cause could be the failure of the exception handling routine that is invoked to recover from the first failure. Similarly, a defect of level ten would require ten independent failures to occur in a specific combination. Clearly, the higher the level of a defect, the less likely its occurrence will be [2,4,6,13].

We can now formulate the hypothesis that the defect level of potentially catastrophic failures, say the ones that can cause a complete system failure, is relatively high, requiring multiple things to fail in combination. If true, high impact failures will tend to have a lower than average probability of occurrence, and are more likely to survive traditional testing.

4. Simulation Model

The simulation accepts a set of input parameters which specify the simulated situation. These input parameters include the number of defects entering the process each day, the number of available servers (employees) for each phase and average time spent on defect on each phase.

**Model with Poisson Input Constant Service Time**

The defects are modeled to have an exponentially distributed intensity of arrival, i.e. they arrive according to Poisson process.

We assume

(a) S servers (employees)

(b) Each server provides service at the same constant average rate $\text{avg}_r$.

(c) The average arrival rate $\text{arriv}_r$ is constant; for all $n$

\[
\text{arriv}_r < S.\text{avg}_r
\]

Where $\text{arriv}_r$ = arrival rate

$D$ = average number of defects in the system

$Dq$ = average number of defects in the queue

$Dw$ = average number of defects in the nonempty queues

$W$ = average time a defect spends in the system

$Wq$ = average time a defect spends in the queue

$Ww$ = average time a defect spends in the queue if it must wait

\[
P(n > k) = \left(\frac{\text{arriv}_r}{\text{avg}_r}\right)^{k+1} = \text{probability of more than } k \text{ defects in the system.}
\]
\[ p(T > t) = e^{-\text{avg}_r (\text{arriv}_r / \text{avg}_r)} = \text{probability the time in the system is greater than t} \]

\[ = \sum_{n=0}^{s} p_n \quad n = 0, 1, \ldots, s - 1 \]

With these assumptions (a), (b), (c) and (d), we have

\[ p_n = \frac{1}{s! s^{s-r}} \left( \frac{\text{arriv}_r}{\text{avg}_r} \right)^n p_0 \quad n \geq s \]

\[ p(n \geq s) = \text{probability an arrival has to wait for service} \]
\[ = \text{probability of at least s defects in the system} \]

\[ D_q = \frac{1}{s! s^s} \left( \sum_{n=0}^{s} \frac{1}{s!} \left( \frac{\text{arriv}_r}{\text{avg}_r} \right)^n \right) \]

\[ D = D_q + \frac{\text{arriv}_r}{\text{avg}_r} \]

\[ W = \frac{D}{\text{arriv}_r} \]

\[ W_q = \frac{D_q}{\text{arriv}_r} \]
\[ P(T > t) = e^{-t \cdot \text{avg}_r} \left\{ \frac{(\text{arriv}_r / \text{avg}_r)^s}{s!} P_0 \left[ 1 - e^{-t \cdot \text{avg}_r (s - 1) \cdot \text{arriv}_r / \text{avg}_r} \right] \right\} \]

5. Results and Discussion

Results of the simulation model stated above, (where arrival rate is generated through Poisson distribution) are as given below:

Case 1: \( \text{avg}_r=6, S=3 \) and \( T=10 \) days

<table>
<thead>
<tr>
<th>Table 1. Random Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>arriv_r</td>
</tr>
<tr>
<td>D</td>
</tr>
<tr>
<td>DQ</td>
</tr>
<tr>
<td>W</td>
</tr>
<tr>
<td>WQ</td>
</tr>
<tr>
<td>PT</td>
</tr>
<tr>
<td>PN</td>
</tr>
<tr>
<td>PZERO</td>
</tr>
<tr>
<td>PS</td>
</tr>
</tbody>
</table>

For higher values of \( \text{arriv}_r \), queueing system not valid because \( \text{arriv}_r < s \cdot \text{avg}_r \).

![Fig 1. Waiting time of defects in the system](image1)

![Fig 2. Probability Graph](image2)
From the table it is observed that average no of defects waiting in the queue increases with increase in the arrival rate of the defects, so the average time a defect spends in the system (w) and queue (both) increasing. From fig 2 shows that the probability that no defect is in the queue is decreasing. The probability of at least s defects in the system is increasing. So, the probability that an arrival has to wait for removal is also increasing.

**Case 2:** $\text{arriv}_r = 15$, $T=10$ days and $\text{avg}_r = 6$

<table>
<thead>
<tr>
<th>Table 2. Random Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>arriv_r</strong></td>
</tr>
<tr>
<td><strong>avg_r</strong></td>
</tr>
<tr>
<td>N</td>
</tr>
<tr>
<td>S</td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<td>D</td>
</tr>
<tr>
<td>DQ</td>
</tr>
<tr>
<td>W</td>
</tr>
<tr>
<td>WQ</td>
</tr>
<tr>
<td>PT</td>
</tr>
<tr>
<td>PN</td>
</tr>
<tr>
<td>PZERO</td>
</tr>
<tr>
<td>PS</td>
</tr>
</tbody>
</table>

The above graph reveals that by adding two employees in the system the defect creep can be stabilized. It will reduce the average time a defect spends in the system and queue. The critical Probability chart in fig 4 also shows that by adding the employees the probability that no defect in the system started increasing and the stabilized.
6. Conclusion

In the practical software development software quality is generally evaluated by the no of residual defects. To keep the no of residual defects with permissible value too much effort is often assigned to software testing. In the above work a simulation technique has been used to know the effect of residual defects on the project. The present simulation process will be an asset in IT industry in order to stabilize the effect of residual errors in software development process and to release the software product in an estimated scheduled time. The future research lies in the holistic view and system vide solutions to the entire management process.

References

Authors

Dr Rachna Soni is Associate Professor and Head, Deptt of Computer Science & Applications, did her M.Phil from IIT, Roorkee and Ph.D from Kurukshetra University, Kurukshetra. She has more than eighteen years teaching experience in the institution of repute. She has published eight papers in International/national journal /conferences. Her area of interest includes Software Risk Management, Project Management, Requirement Engineering, Component based Software Engineering, Simulation etc.

Dr. Ashish Jolly received his MCA degree from University of Madras, Chennai in the year 1999. He is Ph.D in Computer Science from Department of Computer Science & Applications, Kurukshetra University, Kurukshetra, India. He is working as a Professor in the Department of Computer Science & Engineering, Chitkara University, Rajpura, Punjab, India. He has more than eleven years of teaching experience. He has published many research papers in referred National/International journals of repute. His research area includes Simulation, Software Engineering, Requirement Engineering and Software Project Management.

Arti Rana, a Ph.D. candidate at Dept. of Computer Science & Engineering, Birla Institute of Technology, Mesra (Ranchi). She received her M. Phil. in Computer Science from CDLU, Sirsa in 2008 & Masters in Computer Applications from Punjabi University, Patiala in 2002. She has been in to teaching computer applications for more than six years at both under-graduate and post-graduate levels. Presently, she is working in Amity Institute of Information and Technology, Amity University, Noida. She has published several research papers in various national conferences & journals in India. Her research area includes Software Engineering, Software Release Management & Simulation.