Global Journal of Computer Science and Technology
Software & Data Engineering
Volume 12 Issue 13 Version 1.0 Year 2012
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

# Requirement Implementation and Defect Removal across Component Versions: A Simulation Based Approach

By P K Suri & Sandeep Kumar
*Galaxy Global Group of Institutions, Dinarpur, Ambala, Haryana, India*

*Abstract -* Competition in component market and short time to market the software components forces the organization to develop and launch the components in an iterative manner. Components are launched in various versions. All gathered requirements cannot be implemented in initial version. So requirements need to be prioritized and implemented in subsequent versions. Similarly defects in one version are taken care of in subsequent versions. In the present work we have proposed a simulation model that can be used to study the operational characteristics of the requirements implementation process and defect removal process in a Component Based Software.

*Keywords : Component Based Software, COTS, Simulation, Requirements Implementation, Defects Removal, Exponential Distribution, Normal Distribution.*

*GJCST-C Classification: D.2.1*

REQUIREMENT IMPLEMENTATION AND DEFECT REMOVAL ACROSS COMPONENT VERSIONS A SIMULATION BASED APPROACH

# Requirement Implementation and Defect Removal across Component Versions: A Simulation Based Approach

P K Suri [α] & Sandeep Kumar [σ]

*Abstract -* Competition in component market and short time to market the software components forces the organization to develop and launch the components in an iterative manner. Components are launched in various versions. All gathered requirements cannot be implemented in initial version. So requirements need to be prioritized and implemented in subsequent versions. Similarly defects in one version are taken care of in subsequent versions. In the present work we have proposed a simulation model that can be used to study the operational characteristics of the requirements implementation process and defect removal process in a Component Based Software.

*Keywords : Component Based Software, COTS, Simulation, Requirements Implementation, Defects Removal, Exponential Distribution, Normal Distribution.*

## I. Introduction

Traditionally, development of software would focus on developing for a particular kind of application only. But Component Based Development is a market driven technology. Here Components are not developed for a specific application. Rather they are developed to be reused in many different kinds of applications. Different organizations in the common marketplace offer different components for the same functionality. Though, competition in the market is not that much high at present, but looking at the growth trends of the component based technology, the day doesn't seem to be far away when there will be a cut throat competition in the COTS market. In such type of scenario, it becomes very important for various market players and stakeholders to develop quality software components in least available time and release them in market. In this market driven environment, Requirement Engineering is getting more and more attention [1],[2],[3]

It's not only that quality components are to be released in the market as soon as possible; equally important is that the quality of the components is improved gradually. Various organizations throughout geographically dispersed locations improve the components continuously and release their independent versions in the market after the improvements [4], [5], [6]. If we follow Evolutionary development paradigm [14] then an organization must deliver first operational version of the software or component as fast as the system architecture is defined. This first version should incorporate a minimum set of requirements in such a way so that the end user can start working with it. That's why this initial version should be called an operational model. One of the reasons for the early release of the first operational version and subsequent versions of a software component is short time to market for components. Once first version of the component has been released in the market and used by the users, remaining system requirements (that were not incorporated in the earlier version) and some new requirements can be added to the component in its future version releases. Not only this, users of the component will come up with certain defects in the earlier version of the component. These defects can also be removed in such a way that they are not present in all future versions. Though it is possible that some new defects may creep-up in the current version release, and they can always be taken care of in the next version.

Although it is never possible to freeze the requirements in any software development paradigm, still efforts should be made to gather as much requirements as possible, before the release of first version of that component. Out of these most important features can be implemented in the first version of the component and rest of the features can be implemented in subsequent component versions, along with newly generated requirements between release of any two versions, and defects identified in previous component version removed.

For the efficient development of software component, requirements should be properly elicited, analysed and documented at the beginning of a project. Also important is the correct implementation and management of these requirements in the later stages of the component development and integration. This becomes all the more important because all other component development activities are based on how efficiently requirements have been managed. In an ideal software component development environment it is just sufficient to elaborate the requirements into working

*Author α : Dean, Research and Development; Chairman CSE/IT/MCA, HCTM Technical Campus, Kaithal, Haryana, India.-136034.*
*Author σ : Assistant Professor and Head, Faculty of Computer Applications, Galaxy Global Group of Institutions, Dinarpur, Ambala, Haryana-133207, India. E-mail : sandeepnain77@gmail.com*

software component designs, code, and tests. But software development in general and component development in particular is not that much a straight forward thing. In practical software development process (component development processes in that sense), the requirements keep changing, new requirements keep coming and sometimes old requirements need to be removed also. So the process of management and implementation of requirements is a complex task. Need to develop and release initial component version as quickly as possible makes the process more complex. Due to this sometimes problems are encountered in maintaining consistency among the various releases of the component versions. These problems generally come into picture when components are integrated in a component based system.

The process improvement proposals in an organization can be analysed by carrying out pilot studies or controlled experiments in that organization [7]. But this method is very time consuming and resource crunching. Alternatively, simulations can be used to study the behaviour of such a system [8], [9]. Simulation approach has been applied in many areas of engineering and is suitable for application in evaluation of software development processes also. After analysing the new processes using simulation, they can also be analysed in experiments and case studies to establish the fairness of the results obtained using simulation. In this way simulations can be a natural part of technology transfer [10] and evaluation. If simulation is applied for the evaluation of new technologies and processes then it becomes easy to identify the changes and evaluate them in experiments and pilot projects. Sometimes there are certain changes that do not result in process improvement. Application of simulation reduces the risks associated with such changes. Lot of human resources are often involved in experiments and pilot-studies in an organization, introduction and evaluation of wrong changes can lead to a lot of problems. This can potentially damage the continued process improvement work in the organisation for a long time. Hence simulation is needed in the evaluation of new software process technologies.

In the present work, we have proposed application of discrete event simulation [11] using queuing network model [12]. Objective of the study is to find ways for effective management of the human resources of an organization for requirement management and implementation and defect removal while releasing various versions of a software component one after the other. The motivation for the proposed model has come from REPEAT (Requirement Engineering Process At Telelogic)[15].

Basic idea is to have a database of all the requirements to be implemented in a software component. But it is not possible to implement all the requirements in first (or few subsequent versions for that matter) version of the component due to many factors. Most influential of these factors being the competition from other market players and very short time to market. Due to this reason requirements need to be prioritized on some basis and implemented according to their priorities. The steps of the REPEAT lifecycle model for requirement implementation are given as follow [13]:

### New

This state represents the initial state of a requirement, and every requirement is defined as new immediately after it has been issued and given an initial priority.

### Assigned

A requirement is elevated to the assigned state when an expert team has been assigned to investigate the requirement and determine the value of a number of attributes.

### Classified

When reaching this state, an expert team has assigned values to attributes representing a rough estimate of cost and architectural impact. Comments and implementation ideas may also be stated.

### Selected

All requirements in this state are selected for implementation for the coming release. They are sorted in priority order on two list: a must-list for mandatory requirements and a wish-list for “nice-to-have” requirements. They also have attributes assigned concerning detailed cost and impact estimations. There is also a more detailed textual specification of the requirement. A selected requirement may be deselected, due to changed circumstances, and then re-enters the classification state or gets rejected.

### Applied

This is an end-state indicating that the requirement has been implemented and verified. The requirement is now incorporated in a component release that can be marketed to customers.

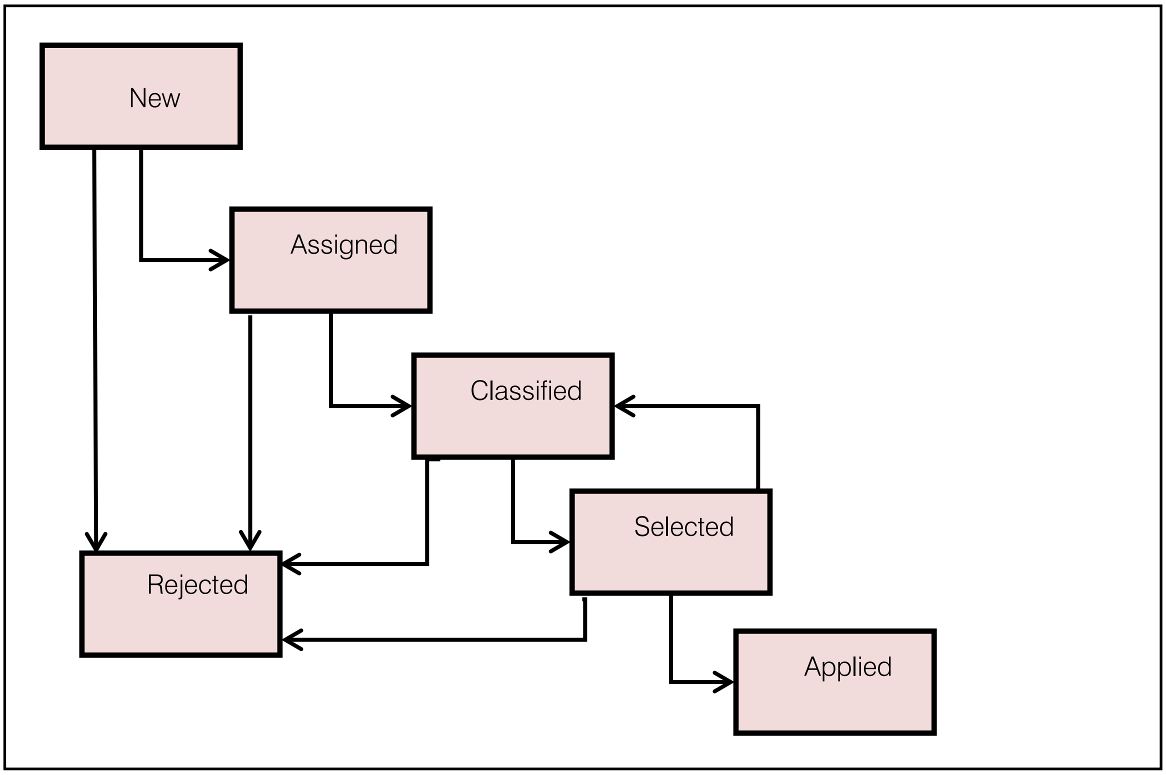

*Fig. 1 :* Repeat Requirement Lifecycle Model [CAR 2000]

**Rejected**

This is an end-state indicating that the requirement has been rejected, e.g. because it is a duplicate, already implemented, or it does not comply with the long-term product strategy.

## II. Proposed Model

For the proposed model, we assume that first operational version of the component has been released in the market. Model can be implemented from second version onwards. Once, the first component version (with bare minimum requirements) becomes operational, process for the release of second and subsequent versions start. More requirements may be added to the requirement database between the releases of any two versions. So this set of requirements to be implemented in a version of software component form a queue. These requirements are to be implemented by a team of developers. Once a component has been released in the market, it is used by various end users in their applications and feedback from the users is received. Certain defects may also be reported by the users. These defects may have crept in due to implementation errors or discrepancies. These defects need to be removed so that they are not part of any future version of the component. So these defects form another set of inputs to the system. We assume that requirements to be implemented in the future component version and defects reported from the previous component version form a common queue. System is modelled as "two parallel servers" queuing system. It is the job of Software Component project manager, modelled as team T, to decide which of the inputs are new requirements, and which of the inputs are defects. Depending upon the nature of input, it is assigned to a different team. Requirement implementation is performed by team TR and defect removal process is performed by team TD.

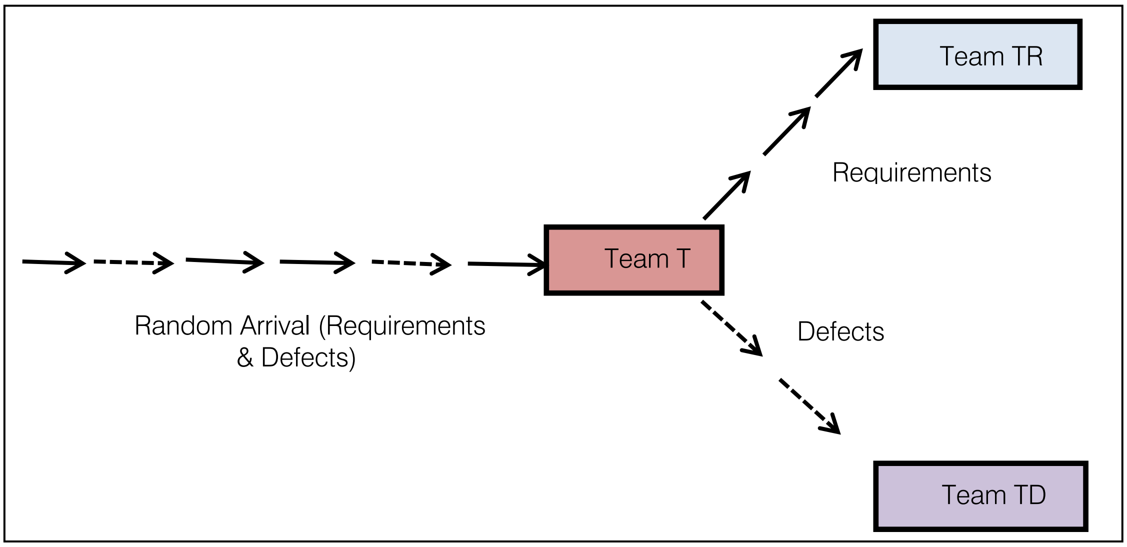

*Fig. 2 :* Requirements and Defects Arrival and Service

## III. Terms and Notations

Following terms and notations have been used for the proposed queuing model of the system as far as arrival patterns and service patterns are concerned:

| | | |
|---|---|---|
| λ | : | Average requirement/defect inter arrival time at team T. |
| $\mu$ | : | Requirements/Defects arrival rate at team T. |
| mR | : | Mean service time of team TR. |
| sdR | : | Standard Deviation of service times at team TR. |
| mD | : | Mean service time at team TD. |
| sdD | : | Standard Deviation of service times at team TD. |
| N | : | Total no. of arrivals at team T (requirements+ defects). |
| R | : | No. of requirements implemented by team TR. |
| D | : | No. of defects removed by team TD. |
| qR | : | Queue length of requirements at team TR. |
| qD | : | Queue length of defects at team TD. |
| sR | : | Service terminating at team TR (Requirements). |
| sD | : | Service terminating at team TD (Defects). |
| IAT | : | Inter arrival time between any two consecutive requirements/ defects. |
| NAT | : | Next arrival time of requirement/ defect. |
| wtR | : | Time a requirement waits in queue before it is implemented. |
| wtD | : | Time a defect waits in the queue before it is removed. |
| itR | : | Idle time of team TR. |
| itD | : | Idle time of team TD. |
| btR | : | Busy time of team TR |
| btD | : | Busy time of team TD |
| stR | : | Team TR service time. |
| stD | : | Team TD service time. |
| SRUNS | : | No. of simulation Runs. |
| $r_i$ | : | Random number. |
| maxqR | : | Maximum requirements in queue at team TR at any time. |
| maxqD | : | Maximum defects in queue at team TD at any time. |

## IV. Algorithm

Formally, algorithm for the model is described as follows:

```
1. Read Input Data.
2. Initialize SRUNS. Set clock:=0, N:=0, R:=0, D:=0, qR:=0, qD:=0, sR:=0, sD:=0, wtR:=0, wtD:=0, itR:=0, itD:=0.
3. Generate random numbers $r_i$'s.
4. Compute inter arrival times of requirements/defects (IAT's) at team T using exponential distribution with arrival rate $\mu$.
5. (At team T, categorise arrival as a requirement or defect.)
If ($r_i$< .8),
    Designate the arrival as a requirement, increment qR.
Else
    Designate the arrival as a defect, increment qD.
6. (Check present status of team TR)
  a. If (clock >=sR), then do
Update wtR.
If qR is positive, then do
    i. Decrement qR by 1.
    ii. Generate stR's using normal distribution with mean mR and standard deviation sdR.
    iii. sR:= clock+stR.
    iv. Increment R by 1.
Else do
Update itR (idle time of team TR).
  b. If (clock <sR) then do
Update waiting time, wtR of requirement at team TR.
```

7. (Check present status of team TD)

a. If (clock >=sD), then do
Update wtD.
If qR is positive, then do

i. Decrement qD by 1.

ii. Generate stD's using normal distribution with mean mD and standard deviation sdD.

iii. sD:= clock+stD.

iv. Increment D by 1.
Else do
Update itD (idle time of team TD).

b. If (clock <sD), then do
Update waiting time, wtD of requirement at team TD.

8. Compute total Busy and Idle times of team TR and TD
9. Compute average waiting times of requirements and defects.
10. Print Required Data.
11. Stop.

## V. Results and Discussion

Simulator was executed for various values of SRUNS. If we assume that on an average 1 requirement or defect arrives at team T every 7 time units, with exponential distribution, requirements are implemented by team TR at a service rate that is normally distributed with value of mR=6.0 and sdR=2.0; and defects are removed by team TD at a service rate that is again normally distributed with value of mD=12.0 and sdD=6.0, then results for various values of SRUNS are shown in table 1.

Graph in figure 3 shows that values of various operational characteristics have larger variation if simulator is run less than 10000 times. Values of btR, itR, btD and itD tend to stabilize after 10000 simulation runs. Same is true for the results depicted in figure 4. Hence it can be said that 10000 simulation runs are sufficient to get the accurate results.
Relationships between number of simulation runs v/s N, R and D is shown in figure 5.

Table 2 shows the results obtained from 50000 simulation runs where value of λ varies from 5 to 10 in steps of 1. Table contains values of idle times of teams TR (itR) and TD (iTD), waiting times of teams TR (wtR) and team TD (wtD) and maximum queue lengths at TR and TD for various values of λ. It is clear from figure 4 that idle times of team TR (itR) and team TD (itD) increase with the increase in the value of λ. waiting times of the requirements and defects decrease with increase in the values of λ. There is variation in maximum queue lengths initially, but as the value of λ increases, maximum queue lengths for both the teams tend to get stabilize.

| SRUNS | btR (%) | btD (%) | itR (%) | itD (%) | wtR (avg.) | wtD (avg.) | Ma qR | Max qD | N | R | D |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1000 | 74.43 | 27.7 | 25.57 | 72.3 | 6.43 | 10.38 | 6 | 3 | 149 | 127 | 22 |
| 2000 | 76.79 | 35.04 | 23.21 | 64.96 | 8.53 | 5.38 | 7 | 3 | 308 | 257 | 51 |
| 10000 | 69.14 | 34.13 | 30.86 | 65.87 | 7.55 | 3.93 | 11 | 3 | 1429 | 1151 | 278 |
| 20000 | 67.44 | 34.72 | 32.56 | 65.28 | 6.89 | 3.52 | 11 | 3 | 2805 | 2232 | 572 |
| 30000 | 68.35 | 35.2 | 32.65 | 64.8 | 7.32 | 3.87 | 11 | 4 | 4257 | 3396 | 861 |
| 40000 | 68.63 | 35.47 | 31.37 | 64.53 | 7.02 | 3.62 | 11 | 4 | 5711 | 4555 | 1156 |
| 50000 | 68.73 | 35.64 | 31.27 | 64.36 | 6.85 | 3.72 | 11 | 4 | 4146 | 5698 | 1441 |

*Table 1 :*

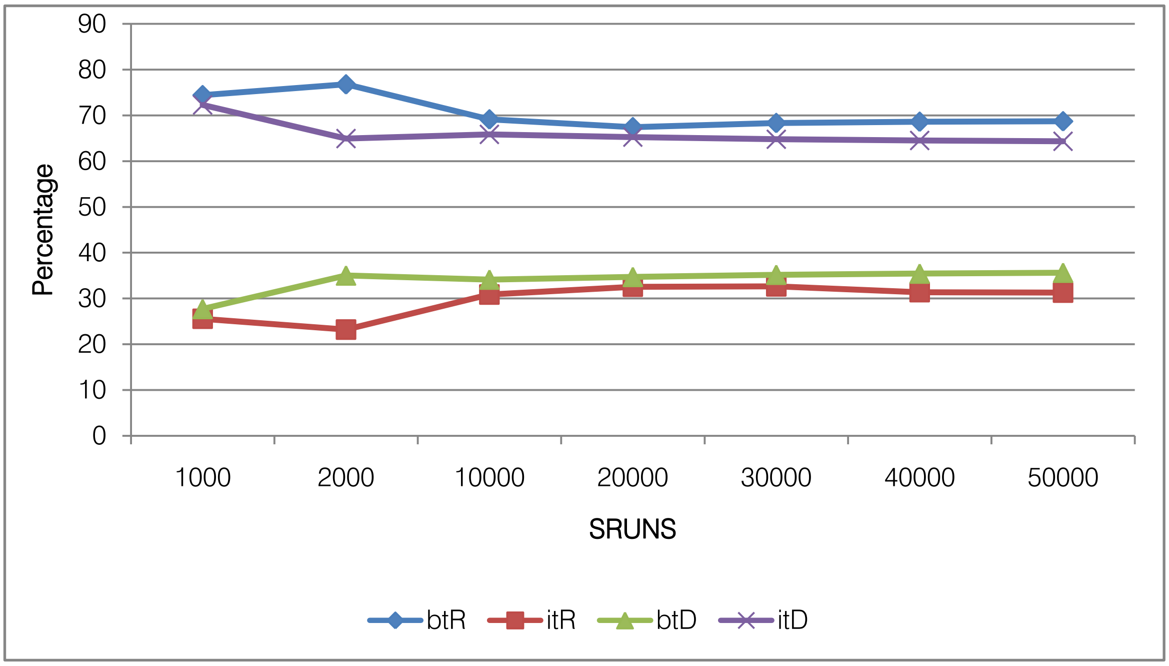
90
80
70
60
50
40
30
20
10
0
Percentage
1000
2000
10000
20000
30000
40000
50000
SRUNS
btR
itR
btD
itD

*Fig. 3 :*

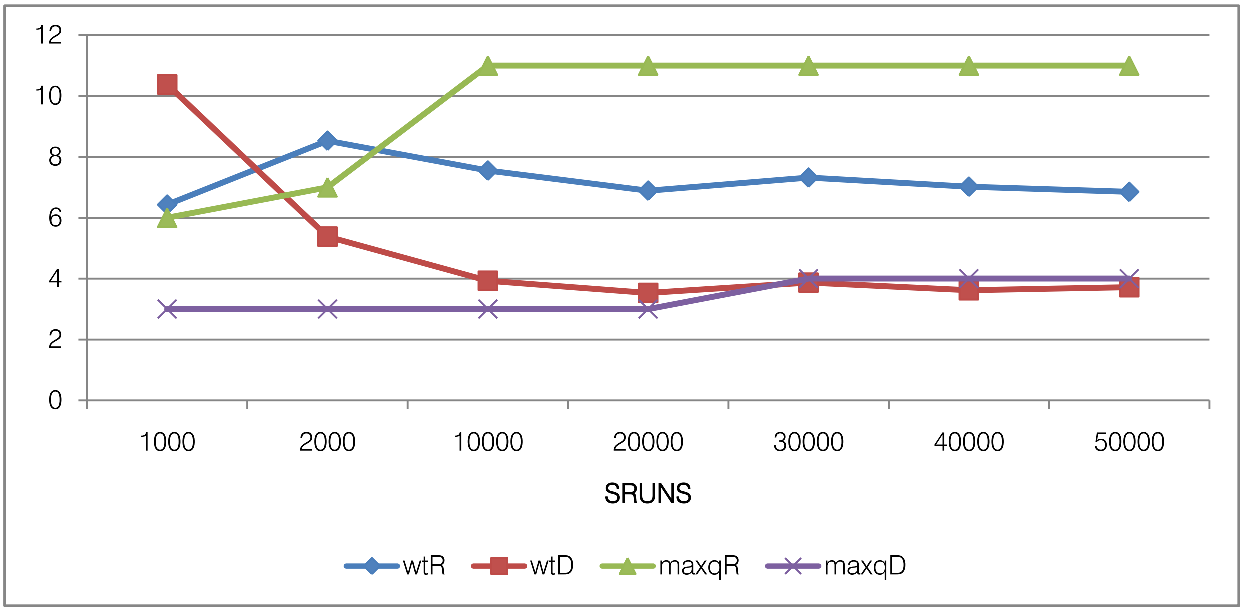
12
10
8
6
4
2
0
1000
2000
10000
20000
30000
40000
50000
SRUNS
wtR
wtD
maxqR
maxqD

*Fig. 4 :*

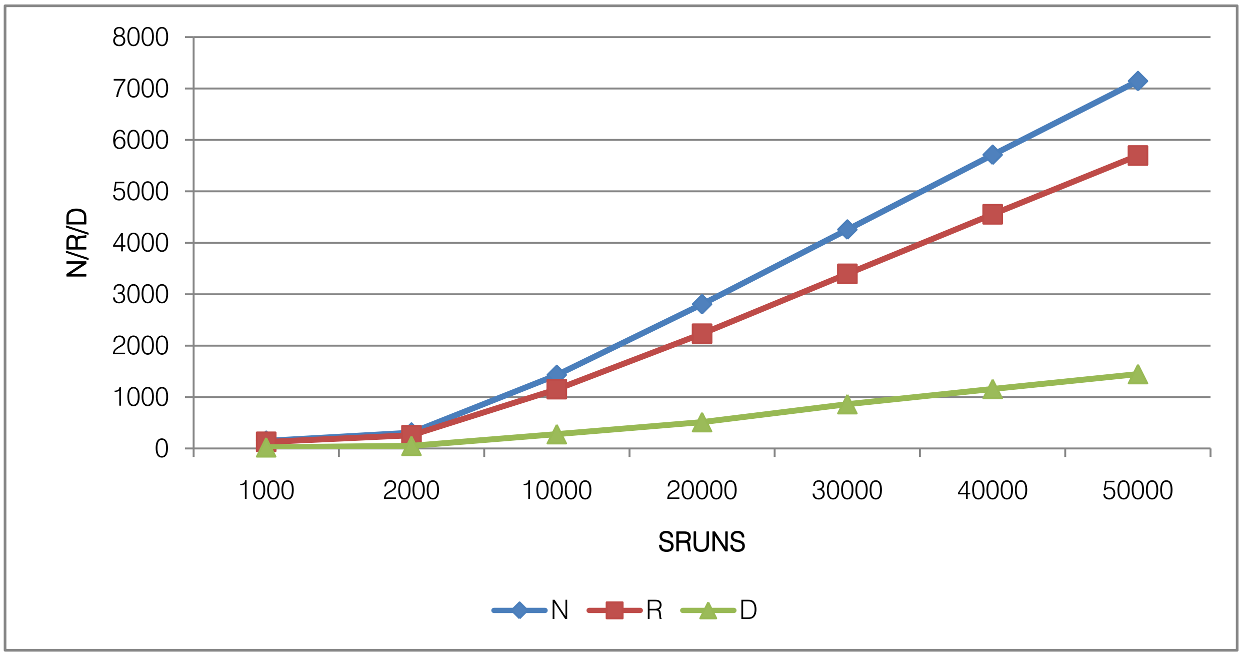

*Fig. 5 :*

| λ | μ | itR | itD | wtR | wtD | maxqR | maxqD |
|---|---|---|---|---|---|---|---|
| 5 | 0.2 | 4.35 | 52.65 | 49.13 | 7.07 | 30 | 6 |
| 6 | 0.1667 | 19.21 | 59.24 | 13.9 | 5.73 | 23 | 7 |
| 7 | 0.1429 | 31.16 | 68.8 | 7.42 | 4.21 | 7 | 3 |
| 8 | 0.125 | 38.76 | 70.04 | 5.06 | 3.26 | 8 | 4 |
| 9 | 0.1111 | 45.6 | 73.2 | 4.02 | 2.73 | 8 | 4 |
| 10 | 0.1 | 57.32 | 75.56 | 3.03 | 2.43 | 6 | 3 |

*Table 2 :*

Global Journal of Computer Science and Technology ( C ) Volume XII Issue XIII Version I Year 2012

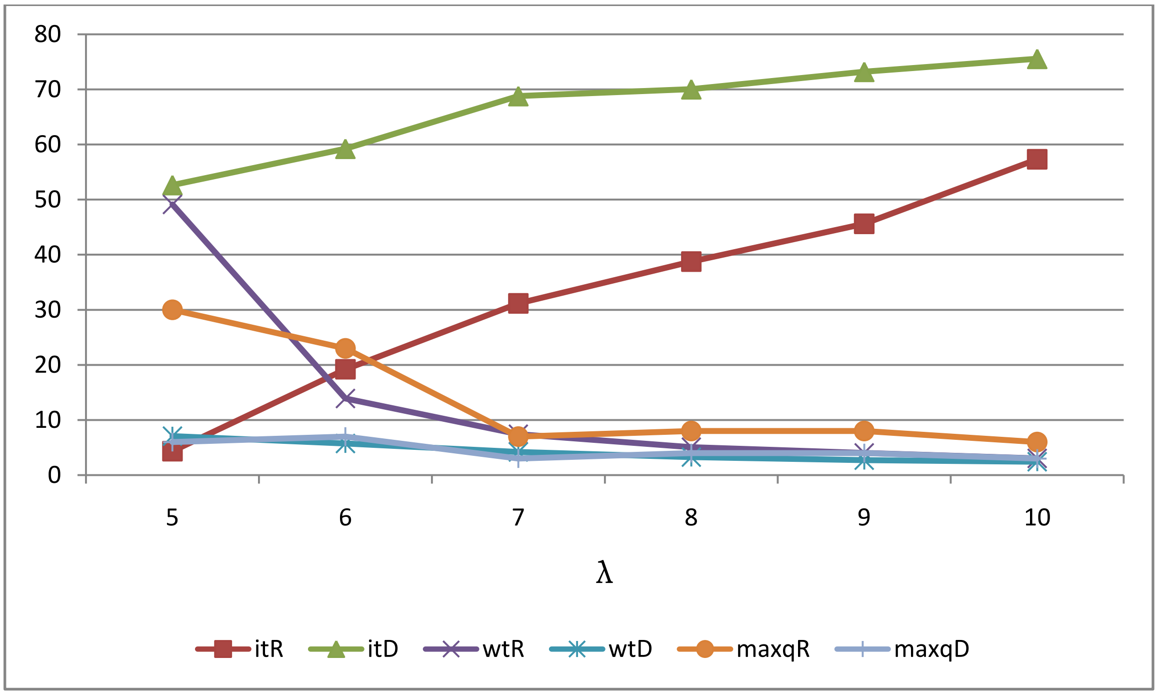

*Fig. 6 :*

## VI. Conclusion

In the presented work, a simulator has been proposed that can be helpful in implementation of user requirements and removal of defects across various versions of a software component in such a way so that size of the requirements implementation team and defects removal team can be optimized. Simulator has been modelled as a two parallel server queuing model, where requirements and defects initially form a common queue and then they are categorized as requirements or defects depending upon their characteristics. Requirements and defects are then handled by different teams. Busy and idle times of both the teams can be studied and depending upon that team size can be decided. Simulator can also be used to study other operational characteristics like the time a requirement or defect has to spend waiting before it is implemented/ removed and maximum length of the queues formed by requirements and defects at each team.

## References Références Referencias

1. Lubars M., Potts C. and Richter C. (1993), A Review of the State of the Practice in Requirements Modeling, Proceedings of First IEEE International Symposium on Requirements Engineering (RE'93), San Diego USA, IEEE Computer Society Press.
2. Potts C.(1995), Invented Requirements and Imagined Customers: Requirements Engineering for Off-the Shelf Software, Proceedings of Second IEEE International Symposium on Requirements Engineering (RE'95), York UK, IEEE Computer Society Press.
3. Yeh A.(1992), Requirements Engineering Support Technique (REQUEST) – A Market Driven Requirements Management Process, Proceedings of Second Symposium of Quality Software Development Tools, New Orleans USA, IEEE Computer Society Press, 211-223.
4. Szyperski C.(1998), Component Software—Beyond Object-Oriented Programming. Addison-Wesley/ACM Press: Boston, MA, 1998.
5. Heineman G.T. and Councill W.T. (2001.), Component-Based Software Engineering: Putting the Pieces Together. Addison-Wesley: Reading.
6. Crnkovic I. et al.(2001), Proceedings of the Fourth ICSE Workshop on Component-Based Software Engineering: Component Certification and System Predication. Software Engineering Institute: Pittsburgh.
7. Wohlin C., Runeson P., Höst M., Ohlsson M.C., Regnell B. and Wesslén A. (2000), Experimentation in Software Engineering -An Introduction , Kluwer Academic Publishers.
8. Kellner M.I., Madachy R.J. and Raffo, D.M.(1999), Software Process Simulation Modeling: Why? What? How? Journal of Systems and Software, Vol. 46, No. 2-3, 91-105.
9. Pfahl D. and Lebsanft K.(2000), Using Simulation to Analyse the Impact of Software Requirement Volatility on Project Performance, Proceedings of the combined 11th European Software Control and Metrics Conference and the 3rd SCOPE conference on Software Product Quality , Munich, Germany,267-275.
10. Linkman S. and Rombach, H.D.(1997), Experimentation as a Vehicle for Software

Technology Transfer - A Family of Software Reading Techniques, Information and Software Technology, Vol. 39, No. 11, pp. 777-780.

11. Banks J., Carson J. S. and Nelson, B. L.(1996), Discrete-Event System Simulation, 2nd Ed., Prentice Hall.
12. King P. J. B.(1990), Computer and Communication Systems Performance Modelling, Prentice Hall.
13. Carlshamre P. and Regnell B.(2000), "Requirements Lifecycle Management and Release Planning in Market-Driven Requirements Engineering Processes", Published by IEEE CS press in the Proceedings of International Workshop on the Requirements Engineering Process: Innovative Techniques, Models, and Tools to support the RE Process, Greenwich UK.
14. Sommersville I.(1996), Software Process Models", ACM Computing Surveys (CSUR) Volume 28, Issue 1, 269-271.
15. Regnel B., Beremark P. and Eklundh, O(1998). "A Market-Driven Requirements Engineering Process - Results from an Industrial Process Improvement Programme", Journal of Requirements Engineering, Vol. 3, No. 2, 21-29.

This page is intentionally left blank