Identification of Software Rot Using Range Control Limits

Harinder Kaur[1], Raveen Bajwa[2]

[1]M. Tech., Computer Science & Engineering, Baba Banda Singh Bahadur Engineering College, Fatehgarh Sahib, India

[2]Assistant Professor, Department of CSE/IT, Baba Banda Singh Bahadur Engineering College, Fatehgarh Sahib, India

Abstract: ***In this research work we have developed a threshold algorithm that helps to differentiate the metrics values into four categories for identification of degree for software rot ,that might occur in life cycle of software project been build, development and release based on agile development model .Our algorithm make use of dynamic range values for each metric rather than using simply mean for calculating multiple deviations for degree of software rot .As a result from this algorithm we can infer how much prone is software project leading to software rot or near amendment of software project.***

Keywords: Software rot, Agile development model, Conflicts, Software Erosion

1. Introduction

Agile Model does not involve long term planning as it works by breaking tasks into small increments with minimal planning. At the end of the iteration a working product is demonstrated to stakeholders [3]. This minimizes overall risk and allows the project to adapt to changes quickly. Multiple iterations might be required to release a product or new features hence Change is intrinsic to agile model [2]. However, Agile development also has its other side, due to its inherent nature of dealing with change from the stakeholders of the project, it becomes constant source of change leading to conflict, abandonment and non conformance of standards and increased friction between the parties, then the incremental nature of its releases, gives unfound opportunities to keep on rethinking, refactoring each aspect of the project from different Identifying Degree of Software rot using Range control limits levels of management etc. There is no final agreement and disagreement, everything is left for next meeting, for next iteration, this may lead to software rot and finally gives the way to software project dissolvent although Agile model fulfill the customer need from beginning to end and continuous improvement to add into valuable software. Agile allow change in requirements in the late Development [2]. Agile works on delivering software on regularly interval However in case of large software deliverables, it is difficult to assess the effort required at the beginning of the software development life cycle. There is lack of emphasis on required designing and documentation [2]. The project can easily get taken off track if the customer is not clear what final outcome that they want, only senior programmers are capable of taking the kind of decisions required during the development process. Hence it has no place for new/novice programmers, unless combined with experienced resources. Daily cooperation between business people and developers throughout the project include face-to-face conversation [3] leading to wastage of time with no desired results. It does not provide detail documentation at end which may lead to problem on later stages of maintenance [2], which may change to software rot or design erosion in future.

Software designs tend to erode over time to the point that redesigning from scratch becomes a feasible substitute compared to extending the life of the existing design [4]. Software leads to erosion when inappropriate software model testing tools are used there is increase in complexity (due to high dependency between modules) is one of the main cause of software rot .When the actual developer are no longer part of the project and their undocumented documentation may also lead to architectural drift for the new inexperienced developer [4]. Inadequate requirement [7] and time pressure are two more factors which cause software rot [6]. Unused code (interfaces etc.) which normally remain unexecuted start containing bug, with change in user requirements and other environmental factors, this code may executed later thereby introducing bugs and making the software less functional and redundant [6]. We have surveyed various related work on interpersonal conflicts and software rot as mentioned in next section.

2. Related Work

Early phases of software development are more prone to overall project risks [8]. Five types of interpersonal conflicts which may lead to project failure as shown in case study of Enterprise Resource Planning system are presented by Avinder [9]. Software design erosion is inevitable. There are number of non technical factors except technical factors that lead to Design erosion. Software organizations should not be judged by how effectively they prevent erosion but how effectively they identify and resolve eroded components [1]. Design decision earlier taken in evolution of system may conflict with requirements that need to incorporate later in evolution. In this paper evidence of architecture drift, vaporized design decision, design erosion has been given the optimal design strategy does not deliver an optimal design because of change in requirements in later evolving system and has also been discussed extension to object oriented paradigm is required as a solution for design erosion [4]. The decision making system has been proposed using fuzzy logics to deal with erosion symptoms by firstly recognize suitable metrics and results the maintenance actions for system based on resultant symptoms [5]. After conducting systematic review and interacting with multiple programmers we have developed a new model to solve these issues as discussed in next section.

3. Methodology

This section discusses the various steps involved in identification of software rot:
3.1. Development of rubic for identification of issues, metrics that lead to software rot.
3.2. Selection of metrics that influence software rot as shown in Table 1.

Table 1: Metrics Relationship with Software rot

S.No	*Metric name*	*Relation with Software rot*
1	WMC	Higher methods per class means more complexity**.**
2	DIT	Increase in density of bugs lead to software rot.
3	NOC	High NOC indicates improper abstraction and high reuse of base class is not good.
4	CBO	High CBO it makes the design monolithic and dependent
5	RFC	High RFC susceptibility of changes, Density of bugs.
6	LCOM	High LCOM value susceptible to ambiguity.
7	Ca	High Ca means High susceptibility for change.
8	NPM	High NPM means class is highly accessible to complexity and understandability.

3.3. Development of selection criteria for selection of projects: In this step we have selected those five Open Source projects written in Java which were introducing maximum number of insertions and deletions on continuous basis in the coding by developers in very short span of time, building a fact that there was conflict between the developers/analysts or stakeholders as there are continuous changes in coding affecting the overall project by adding more tightly coupled classes or methods to base class or others increasing complexity, ambiguity and improper abstraction of classes which may ultimately lead to software rot as shown in Table 2.

Table 2: Projects Description

Project	*CVS*	*Releases*	*Contributors*	*Total Commits*
Orientdb	GitHub	8	33	6458
Spout	GitHub	0	69	5910
Okhttp	GitHub	7	20	541
Jogl	GitHub	39	28	5126
RxJava	GitHub	48	44	1640

3.4. Algorithm for identification of limits that can help us to find degree of software rot.

Step-1: Let 'n' be the number of Metrics
a. *HE* is High Prone to Software rot degree,
b. *ME* is Moderate Prone to Software rot degree,
c. *LE* is Low Software rot degree.
d. *UTH,MTH,LTH* representing Upper Threshold , Middle Threshold, Lower Threshold.
e. *STD* represents Standard Deviation.
f. Variance is represented by *Vi*.
g. Let Metric values represented by *IN*. *M* represent the mean of your metric values and n is total no. of metric values.
Let where *M1*-array be the numeric values of metric1
Let *M2*-array be the numeric values of metric2
Let *M3*-array be the numeric values of metric3
Let *M4*-array be the numeric values of metric4
Let *MN*-array be the numeric values of metricN

Step-2: For each metric in metrics
1. Let x1, x2……xn are the metric values (*IN*). (1)
2. M=x1+x2+xn/n giving the *V*. (2)
3. *Vi*=(x1-*V*)^2+(x2-*V*)^2+(xn-*V*)^2 (3)
4. *Vi*=(x1-*V*)^2+(x2-*V*)^2+(xn-*V*)^2/n. (4)
5. *STD*= Sqrt (*Vi*). (5)
6. MULTIPLY *STD* by 3 giving *R*. (6)
7. *V+R* result it *UTH*. (7)
8. *V-R LTH*. (8)

IF (*IN*> *UTH*) THEN
{
"add in the group of the high prone to software rot object"
}
ELSE IF (*IN*<=*UTH* &&*IN*>= *MTH*) THEN
{
"add in the group of the medium prone to software rot object"
}
ELSE IF ((*IN* <= *MTH*) &&(*IN*>*LTH*) THEN
{
"add in the group of the less prone to software rot object"
}
ELSE
{
"add in the group of the no prone to software rot object"
}

As shown above we have used different equations for calculations. We have used Equation (4) to calculate variance and standard deviation has been calculated using Equation (5). The ranges are calculated from Equations (6) to (8) and IF-THEN-ELSE control structure is used for selection procedure as mentioned above.

4. Results and Interpretation

4.1. Weighted Methods per Class (WMC) limits

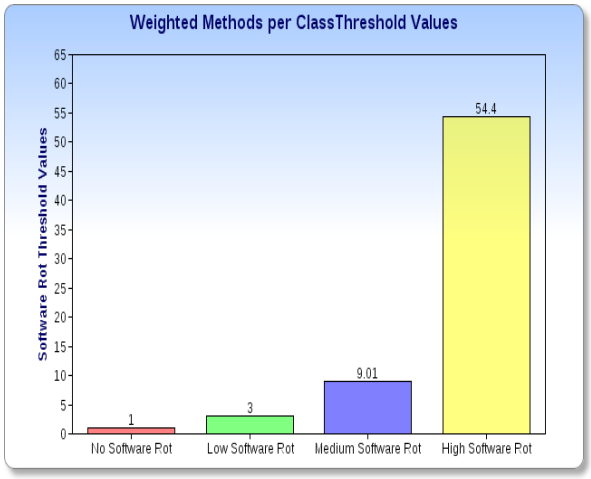

Figure 1: Weighted Methods per Class

The above bar graph (Figure 1) shows the different threshold values found by the threshold algorithm for WMC limits. It is visible from the graph that if the class experiences the analysis by using tool we have build, we will get value of WMC metric. If the value of WMC is between 1 and 3 then there is no probability of Software rot. If value is between 3 and 9.01 there is chances of Software rot which can be

resolved. If it lies between 9.01 and 54.4 then it is medium software rot which need to be resolved. If it is above 54.4 then the project will be leading to software rot. High WMC value in a class means it is highly application specific [10].

4.2. Depth of Inheritance (DIT) limits

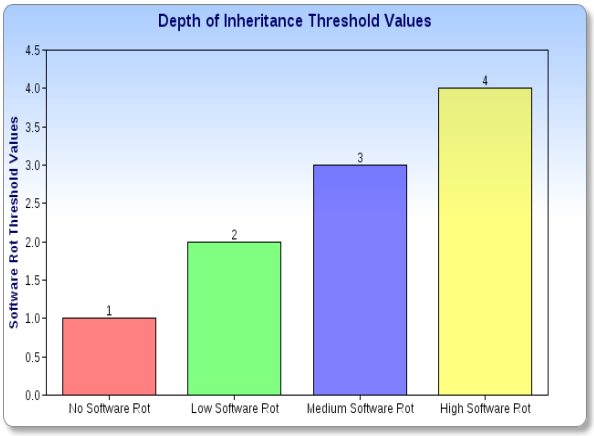

Figure 2: Depth of Inheritance

The above bar graph (Figure 2) shows the different threshold ranges found by the threshold algorithm for DIT. It is apparent from the graph if the class undergoes the analysis by using tool we have build, we will get value of DIT metric. If the value of DIT is close to 1 then there is no probability of Software rot, if value is between 1 and 2 there is chances of Software rot which can be resolved or avoided as it is not risky. If it lies between 2 and 3 its medium software rot which need to be resolved. If above 4 then the project will be highly affected and leading to software rot. The deeper the class hierarchy inherits large methods which make it more complex for the prediction of its behavior [10], then will move to conflicts and may cause base for software rot.

4.3. Number of Children (NOC) limits

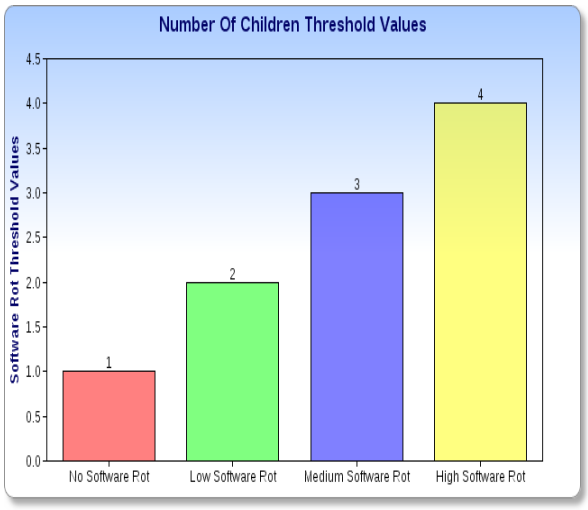

Figure 3: Number of Children

The above bar graph shows the different threshold limits found by the threshold algorithm for NOC as shown in Figure 3. It is noticeable from the graph if the class undergoes the analysis by using tool we have build, we will get value of NOC metric. If the value of NOC is close between 1 and 2 then there is no probability of Software rot, if value is between 2 and 3 there is chances of Software Rot which can be resolved or avoided as it is not risky. If it ranges between 3 and 4 its medium software rot which need to be resolved. If above 4 then the project will be highly endure and leading to software rot. High NOC indicates improper abstraction, high reuse of base class which is not a good practice [10].

4.4 Coupling between Object Classes (CBO) limits

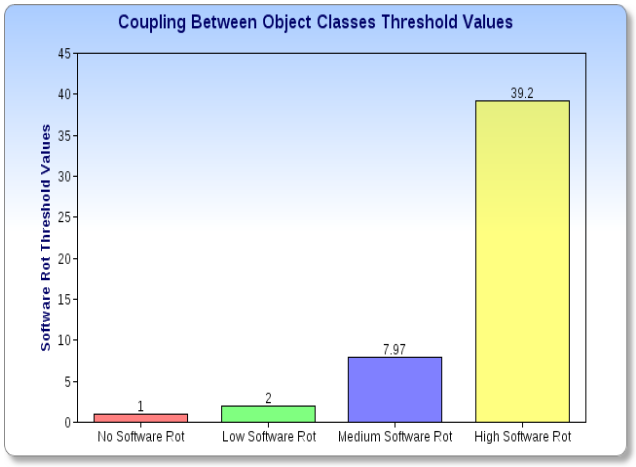

Figure 4: Coupling Between Object Classes

As shown in Figure above different threshold ranges found by the threshold algorithm for Coupling Between Object Classes Threshold Values .It is given from the graph if the class undergo the analysis by using our customized tool, we will get value of this metric. If the value lies between 1 and 2 then there is no probability of Software rot, if value is between 2 and 7.97 there is chances of Software Rot which can be resolved or avoided as it is not risky. If it ranges between 7.97 and 39.2 its medium software rot which need to be resolved. If above 39.2 then the project will be highly prone to software rot. High CBO is not desirable consequently it makes the design monolithic and dependent and hence much more susceptible to changes since either of the coupled classes face changes others will require change thus it will lead to software rot in future[10].

4.5 Response for a Class (RFC) limits

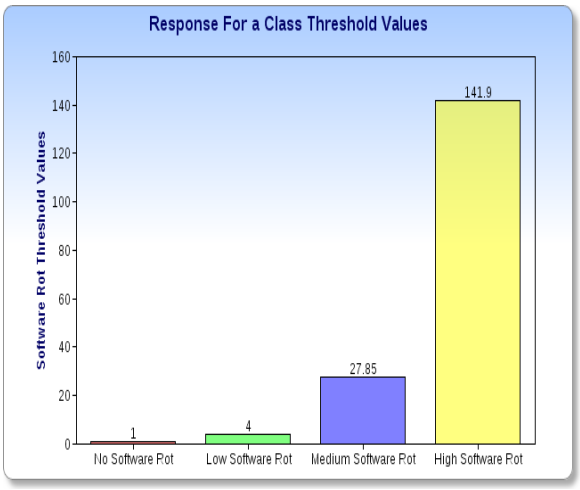

Figure 5: Response For a Class

As shown in Bar graph (Figure 5) the different threshold ranges found by the threshold algorithm for Response for Class. It is traceable from the graph if the class undergoes the analysis by using customized tool we will get value of this metric. If the value is close in range from 1 and 4 then there is no probability of Software rot, if value is between 4 and 27.85 there are chances of Software Rot which can be resolved or avoided as it is not risky. If it ranges between 27.85 and 141.9 its medium software rot which need to be resolved. If above 141.9 then the project will be prone to high software rot. High RFC means High susceptibility of changes; include Density of bugs [10].

4.6 Lack of Cohesion of Methods (LCOM) limits

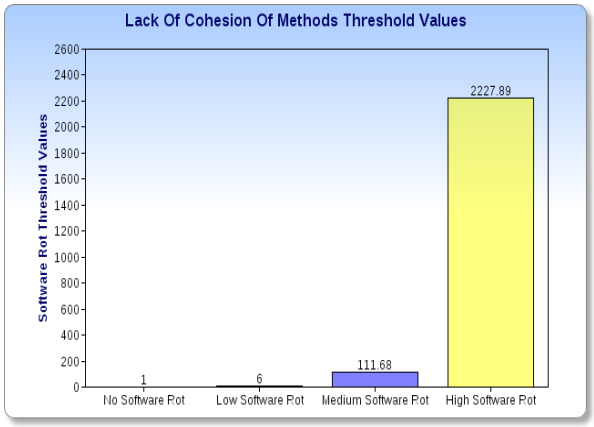

Figure 6: Lack of Cohesion of Methods

The above bar graph(Figure 6) shows the different threshold limits found by the threshold algorithm for Lack of Cohesion of Methods .It is apparent from the graph if the class undergo the analysis by using tool we have build ,We will get resultant value of metric. If the value is close between 1and 6 then there is no probability of Software rot, if value is between 6 and 111.68 there is chances of Software rot which can be resolved. If it ranges between 111.68 and 2227.89 its medium software rot which need to be resolved. If above 2227.89 then the project will be highly leading to software rot. High LCOM value implies class more susceptible to errors and might be disaggregated into two or more classes and increase complexity [10].

4.7. Afferent Coupling (Ca) Limits

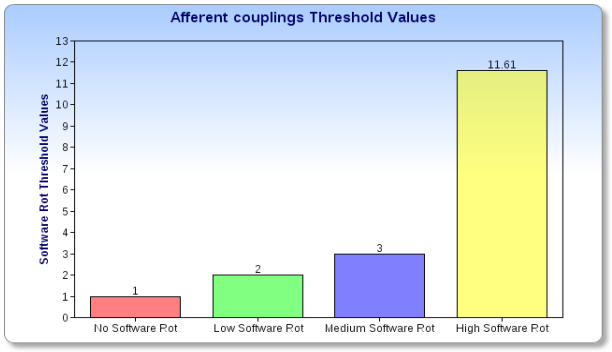

Figure 7: Afferent Coupling

The above bar graph shows the different threshold limits found by the threshold algorithm for Afferent Coupling as shown in Figure 7. It is noticeable from the graph if the class undergo the analysis by using tool we have build, We will get value of Afferent Coupling metric .If the value of metrics is close to range 1-2 then there is no probability of Software rot, if value is between 2 and 3 there is chances of Software rot which can be resolved or avoided as its not risky. If it ranges from 3 to 11.61 its medium software rot which need to be resolved. If above 11.61 then the project will be leading to high software rot. High Ca leads to High susceptibility for change.

4.8. Number of Public Methods (NPM) Limits

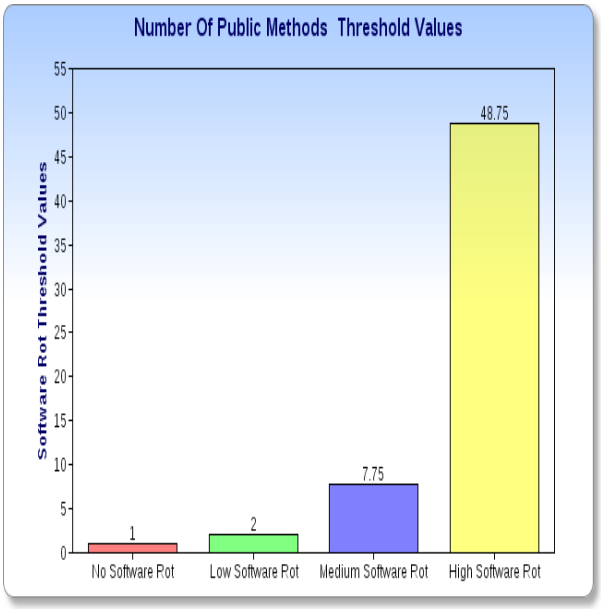

Figure 8: Number of Public Methods

The above bar graph (Figure 8) shows the different threshold ranges found by the threshold algorithm for Number of Public Methods. It is noticeable from the graph if the class undergoes the study by using tool we have build, we will get value of this metric. If the value NPM is close to 1- 2 then there is no probability of Software rot, if value is between 2 and 7.75 there is chances of Software Rot which can be resolved. If it lies between 7.75 and 48.75 then it leading to medium software rot which need to be resolved. If above 48.75 then the project will be leading to high software rot. High NPM means class is highly accessible to other parts of software thus High NPM leads to High susceptibility for changes and complexity.

5. Conclusion

In this research work we have been successfully able to model the interpersonal conflicts reflected in the coding process of building project by measuring the factor/metrics that influence the software rot and based on these values we are able to find the various possible intervals that can be represented in terms of degree of software rot of overall projects in agile development model. From all the graph representations, we can conclude that for these five projects, the range or threshold limits really varies from metric to metric and in each case the value/magnitude identified by the threshold algorithm is quite high. in each case ,For example WMC(54.4),DIT(4), LCOM(2227.89), Ca(11.61), CBO(39.2), RFC(141.9), NOC(4), NPM(48.75). It can also seen that lower bounds found by algorithm basically reflect the conformance levels which must be maintain for the

degree of code to remain relevant for current context of its development in agile development model.eg all this is apparent from values 1 in case of LCOM ,Ca etc and conforming to standards .finally we can safely say that this algorithm can work as base for similar projects studied here for identification of software rot. Suitable and corrective measures can be taken as soon as these metric values go beyond the limits found by this algorithm.

6. Future Scope

We can further extend our work by using machine algorithms for designing decision support system by calculating the value of different parameters.

References

[1] V.Gurp, Jilles,Jan Bosch, and Sjaak Brinkkemper. "Design Erosion in Evolving Software Products." In ELISA workshop, p. 134. 2003.
[2] L. R.Vijayasarathy, Dan Turk,"Agile Software Development: A Survey of early Adoption", Journal of Information Technology Management ,Volume X1X,Number-2,2008.(journal style)
[3] Software Engineering: A Practitioner's Approach,7/e (McgrewHill, 2009) by Roger S. Pressman, Slides copyright@1996,2001,2005,2009. (book style)
[4] Van Gurp , Jilles, and Jan Bosch. "Design erosion: problems and causes." Journal of systems and software 61, no. 2: 105-119, 2002.(journal style)
[5] R. Perez -Castillo,Ignacio and Mario Piattini, "Diagnosis of Software Erosion through Fuzzy logics", Software Maintenance and Reengineering, 2009. CSMR '09. 13th European Conference ,2009 . (conference style)
[6] H.P.S Dhami & Dr Anuj Kumar", Analysis of software design issues ", International journal of advance research in computer science and software engineering, july, 2013.(journal style)
[7] Damian, Daniela E., and Didar Zowghi. "An insight into the interplay between culture, conflict and distance in globally distributed requirements negotiations." In System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on, pp. 10-pp. IEEE, 2003.(conference style)
[8] A.Shawo, "Determination of risk during requirement engineering process", Journal of emerging trends in computing & information science 3(3), 358-364,2012.(journal style)
[9] walia, Khalid Shergil, "Impact of interpersonal conflict on requirements", A Research Review", University of westernontario ,London ONN6A5B7.
[10] S.R Chidamber and Chris F Kemerer, "A Metrics Suite for Object Oriented Design, IEEE Transactions on Software Engineering, Vol 20 no-6 , 476-493, June 1994.

Author Profile

Harinder Kaur received the B.Tech degree in Computer Science & Engineering from Beant College of Engineering & Technology in 2005. In 2005, she joined Institute of Engineering & Technology Bhaddal as Lecturer in the Department of Computer Science. She is actively involved both in academics and research projects in the field of Computer Engineering & Information Technology.
#

Raveen Bajwa received the B.Tech and M-Tech degrees in Computer Science & Engineering from Baba Banda Singh Bahadur Engineering College in 2005 and 2011 respectively and pursuing her P.H.D from Punjab Technical University Jalandhar. In 2005, she joined Baba Banda Singh Bahadur Engineering College as Assistant Professor in the Department of Computer Science. Her areas of interests include Computer Networks, Database Management System and Computer Graphics.##