July 2014

FAULT-PRONE COMPONENTS IDENTIFICATION FOR REAL TIME COMPLEX SYSTEMS BASED ON CRITICALITY ANALYSIS

D. JEYAMALA

Department of Computer Applications, Thiagarajar College of Engineering, Tamilnadu – 625 015,
djmcse@tce.edu

S. BALAMURUGAN

Department of Computer Applications, Thiagarajar College of Engineering, Tamilnadu – 625 015,
balamsg4u@gmail.com

A. JALILA

Department of Computer Applications, Thiagarajar College of Engineering, Tamilnadu – 625 015,
mejalila@gmail.com

K. SABARI NATHAN

Department of Computer Applications, Thiagarajar College of Engineering, Tamilnadu – 625 015,
sabarinathan4you@gmail.com

Follow this and additional works at: <https://www.interscience.in/ijcsi>

Part of the [Computer Engineering Commons](#), [Information Security Commons](#), and the [Systems and Communications Commons](#)

Recommended Citation

JEYAMALA, D.; BALAMURUGAN, S.; JALILA, A.; and NATHAN, K. SABARI (2014) "FAULT-PRONE COMPONENTS IDENTIFICATION FOR REAL TIME COMPLEX SYSTEMS BASED ON CRITICALITY ANALYSIS," *International Journal of Computer Science and Informatics*: Vol. 4 : Iss. 1 , Article 10.

DOI: 10.47893/IJCSI.2014.1173

Available at: <https://www.interscience.in/ijcsi/vol4/iss1/10>

This Article is brought to you for free and open access by the Interscience Journals at Interscience Research Network. It has been accepted for inclusion in International Journal of Computer Science and Informatics by an authorized editor of Interscience Research Network. For more information, please contact sritampatnaik@gmail.com.

FAULT-PRONE COMPONENTS IDENTIFICATION FOR REAL TIME COMPLEX SYSTEMS BASED ON CRITICALITY ANALYSIS

D. JEYAMALA¹, S. BALAMURUGAN², A. JALILA³, K. SABARI NATHAN⁴

^{1,2,3,4}Department of Computer Applications, Thiagarajar College of Engineering, Tamilnadu – 625 015
E-mail: djmcse@tce.edu¹, balams4u@gmail.com², mejalila@gmail.com³, sabarinathan4you@gmail.com

Abstract- Generally, complexity of Software affects the development and maintenance Cost. The Complexity of the software increases, when the number of Components increase, among these components, some are more critical than others which will lead to catastrophic effects on field use. Hence, it is needed to identify such critical components after coding to test them rigorously. In this paper, we presented a novel approach that helps to identify the critical components in the software based on Criticality Analysis. Criticality Analysis analyzes the critical value of each component based on their Sensitivity and Severity metrics.

Keywords- Software Testing, Critical Components, Software Metrics, Criticality Analysis, Software under Test (SUT), Sensitivity and Severity analysis

I. INTRODUCTION

A Component is said to be critical, if the failure of which may have serious consequences, such as leading to software collapse, loss of data, and loss of money. Identification of such critical components is the most important task during software testing. Any complex software will have 20% of Critical Components [1]. This makes serious effects if some of these components missed during testing. For the proposed approach, a component's criticality is measured in terms of their core Sensitivity and Severity with other components.

In our approach, critical components are identified by analysing their level of criticality. The proposed criticality analysis is used to identify the probability of failure modes of the components using Sensitivity and Severity Analysis. Sensitivity analysis, which is an approach used to determine how a component will impact the other dependent components. This can be done with the help of six metrics such as, Fan-In, Fan-Out, Information Flow, Ratio of Pure Inherited Methods, Weightage of Methods in a class, and Weakness of Methods of a Component. We have used Fan-In, Fan-Out [2], and Information Flow [3] as directed metrics. The metrics Ratio of Pure Inherited Methods, Weightage of Methods in a class, and Weakness of Methods of a Component, are the novel metrics that are not proposed by any other approaches. The Severity analysis is dealt with assessing the impact of failure of a component. Each component of the SUT is analysed to identify what type of failure that might have and how it will impact the functionalities of that component. The failure types are categorized into Catastrophic, Critical, Marginal and Minor as per the approach proposed by Garousi [9]. In addition to the above Sensitivity and Severity analysis, the Criticality analysis considers the execution count and time taken

by a component for its execution, to classify a component as a critical component.

II. FAULT-PRONE COMPONENT IDENTIFICATION

1. Related Works

Ebert Christof [4] has evaluated classification techniques such as Pareto classification, Classification Trees, Factor-Based Discriminant Analysis, Fuzzy Classification and Neural Networks for identifying critical components to predict faults based on code complexity metrics. His study showed that among those classification techniques, fuzzy classification provides the best result for critical component identification. Also, they insisted that, Pareto analysis („80:20 rule“) showed good results for easy identification of the top 20 % of critical modules.

Malhotra and Jain [5] in their study, they have experimented a model to estimate fault proneness using Object Oriented CK metrics and QMOOD metrics using statistical and machine Learning approach. In their experiment they tested 19 object oriented metrics for predicting the faulty classes. Their study showed that, out of 19 tested metrics, a subset of metrics such as WMC - Weighted methods per class, DIT - Depth of Inheritance Tree, NOC - Number of Children, CBO - Coupling Between Object classes, RFC - Response for a Class, LCOM - Lack of Cohesion in Methods, Ca - Afferent couplings, Ce - Efferent couplings, NPM - Number of Public Methods, LCOM3 - Lack of cohesion in Methods, LOC - Lines of Code, DAM: Data Access Metric, MOA: Measure of Aggregation, MFA: Measure of Functional Abstraction, CAM:

Cohesion Among Methods of Class, IC: Inheritance Coupling, CBM: Coupling Between Methods AMC:

Average Method, ComplexityCC - McCabe's Cyclomatic Complexity are significant predictors of fault proneness.

Shatnawi et al. [6] has experimented the effectiveness of software metrics in identifying error-prone classes in post-release software evolution process. In their study they have tested software metrics such CBO, CTA (Coupling through Abstract Data Type), CTM (Through Message Passing), RFC, WMC, DIT, NOC etc., they proved that software metrics are used to identify error prone classes even after the software release evolution process.

Jacek Czerwinka et al. [7] proposed the approach that identifies the fault prone components based on the risk assessment of impact of such post-release change fixes.

The present their experiences with CRANE: a failure prediction, change risk analysis and test prioritization system at Microsoft Corporation that leverages existing research for the development and maintenance of Windows Vista. They identify and evaluate the impact and risk of a change is to understand the exact extent of changes.

Janes et al. [8] their experiment depicted that, the early lifecycle metrics can be used for identifying the most defect prone classes in the context of real-time system such as telecommunication software developed using C++. In their approach, they have applied models such as Poisson Regression, Negative Binomial Regression, and Zero-Inflated Negative Binomial Regression to compare CK metrics with LOC metric.

Their study shows that metrics such as RFC, CBO are based on the communication between classes and better predictor of fault prone classes than LOC metric. Also their study described that, Zero-Inflated Negative Binomial Regression Model counted the variability of the number of defects in classes effectively than other applied statistical models.

2. Working Principle of the Proposed Approach

The Software under Test (SUT) is given as input, and the Sensitivity Metrics and Severity Metrics are calculated for each Component in the SUT, as given below:

2.1 Sensitivity Analysis

It is an approach used to determine how a component will impact the other dependent components. Sensitivity Analysis can be done using the following metrics.

1. Fan In - Number of other Components calling a given Component in SUT.

2. **Fan Out** – Number of other Components being called by this Component.

3. **Information Flow (IF)** – It represents the flow of data in collective procedures in the processes of a concrete system. It can be calculated using the following:

$$\text{Information Flow} = [\text{Fan-in} * \text{Fan-out}]^2 \quad (1)$$

4. **Weightage of Methods in a class** – This is used to show the complexity of a given Component by counting the number of independent paths in each of the methods of the given Component, whereas the Cyclomatic complexity [10] refers to the number of independent paths in a component.

$$\text{Weightage of Methods} = \sum CC_i \quad (2)$$

Where,

CC_i – Cyclomatic complexity of a method in a component, $\forall i=1$ to m

m – Total number of methods in a component.

5. **Weakness of Methods of a Component (WM)** – The Weakness of methods of a component is the sum of Weakness of an each method. This can be represented as following:

$$WM(C_i) = \frac{\sum Wm_i}{m} \quad (3)$$

Where,

Wm_i - Weakness of a method in a component, $\forall i=1$ to m

m - Total number of methods in component

The Weakness of a particular method (Wm_i) can be analysed using the following formula:

$$Wm_i = [LV_i * V_i]^2 \quad (4)$$

Here,

$$LV_i = \frac{\text{Live variables count}}{\text{Number of Executable Lines}} \quad (5)$$

$$V_i = \frac{\text{Live variables count}}{\text{Number of Variables}} \quad (6)$$

Live variables means that the number of variable being used/live at a particular executable line. The following code snippet can be used to explain how live variables are counted.

TABLE - 1 EXAMPLE CODE SNIPPET

```

1  main()
2  void calculate()
3  {
4  int a, b, c;
5  read (a, b);
6  a=b + 5;
7  b=a - 5;
8  c=a + b;
9  write (a, b, c);
10 }
11 end;
```

In the above, the lines 5, 6, 7, 8 and 9 are executable lines. So the executable lines count is 5. The calculation of live variables count is shown in the following table:

TABLE – 2 COUNTING LIVE VARIABLES

Line No.	Live Variables	Count
5	a, b	2
6	a, b	2
7	a, b	2
8	a, b, c	3
9	a, b, c	3
Total		12

From the above calculation, we have live variables count as 12 and total number of variables as 3. So, the weakness of method for the method void calculate () is calculated.

$$LV_i = 12/5 = 2.40$$

$$V_i = 12/3 = 4$$

$$Wm_i = 9.60$$

$$WM(C_i) = 9.60/1 = 9.60$$

The code contains only one method. So, the weakness of method for the sample code we taken is 9.60.

- 6. Ratio of Pure Inherited Methods (RPIM)** – It is the ratio of the number of pure inherited methods from a component by the dependent components.

$$DRIM(C_i) = \frac{p}{m} \quad (7)$$

Where,

p - Number of pure inherited methods
m - Total number of methods in an inherited component.

Sensitivity analysis helps to find out what kind of impact it will be, when the results deviate from the expectations and which component is responsible for that deviation. Based on this analysis, the proposed approach is able to identify the components that highly impact the other components.

2.2 Severity Analysis

The Severity analysis is that the tactic of estimating the implications of failure and prioritizing the components per the severity level of implications. The components with higher severity value could cause the functionalities of the system. To mitigate those failures, the high severity components will be tested fastidiously.

$$SV(C_i) = \sum_{k=1}^p SV(M_k(C_i)) \quad (8)$$

Where,

SV ($M_k(C_i)$) – Severity Value of method k in the component C_i .

p – Number of methods in component C_i

SV ($M_k(C_i)$) is assigned with the values as **0.95, 0.75, 0.50, 0.25** based on the four characteristic described by Garousi [9].

Category 1: Criticality/ Importance of a message to the client. (The Control Flow of a method is purely depending on the outcome of a message from the server object called as Control Coupling.)

Category 2: Messages that call large amount of data/return values than other messages.

TABLE – 3 SEVERITY ANALYSIS

S. No	Message Category	Severity Type	Severity Value
1.	Category 1	Catastrophic	0.95
2.	Category 2	Critical	0.75
3.	Category 3	Marginal	0.50
4.	Category 4	Minor	0.25

Category 3: The return values from methods might be used frequently or for critical decision/computation in the client than other messages.

Category 4: Some of the messages may be triggered more frequently than other messages.

2.3 Criticality Calculation

The critical value associated with each component is calculated based on Sensitivity and Severity metrics associated with it, the execution count and time taken by a component for its execution.

It can be calculated as follows:

$$CI(C_i) = P(CV(C_i)) * P(SV(C_i)) * P(E(C_i)) * \text{Time-taken}(C_i) \quad (9)$$

Where,

- P(CV(C_i))** - based on the Sensitivity Analysis of the Components.

$$P(CV(C_i)) = \frac{CV(C_i)}{\sum CV} \quad (10)$$

Here,

CV (C_i) – Sensitivity value of Component C_i .

CV – Sensitivity value of all the components in SUT

- **P (SV (C_i))** – based on the Severity Analysis of the Components.

$$P(SV(C_i)) = \frac{SV(C_i)}{\sum SV} \quad (11)$$

Here,

SV (C_i) – Severity value of Component C_i.
SV – Severity value of all the components in SUT

- **P (E (C_i))** – based on execution count (E) of Component C_i. Fan-In count represents the execution count of the components.

$$P(E(C_i)) = \frac{FanIn(C_i)}{\sum FanIn} \quad (12)$$

Here,

Fan-In (C_i) – Fan-In count of Component C_i.
Fan-In – Fan-In count of all components in SUT

- **Time-taken (C_i)** - Total time taken by a component for its execution.

2.4 Criticality Analysis

This module analyses the critical level of each component with other components to generate the critical test paths to cover them. Then the selected components are assigned with a flag value to indicate their priority for testing.

For Critical Component selection in a path is calculated as follows:

$$Re(C_{i,j}) = \left| \frac{CI(C_i)}{\sum CI} \right| \quad (13)$$

Where,

CI (C_i) – Critical value of Component C_i
 $\forall i=1 \text{ to } n$
 n – Total number of Components

$$Re(P_j) = \sum Re(C_{i,j}) \quad \forall j=1 \text{ to } q \quad (14)$$

Where,

Re (C_{i, j}) – Risk Exposure of Component i connected to component j.
 q – Number of components in the SUT.

The identified Paths with highest Risk Exposure are selected first and test cases are generated based on message passing using JUnit and Ration Functional Tester.

3. Algorithm of the Proposed Approach The following algorithm explains how the proposed approach is implemented.

Step -1 Read the Project

Step -2 Extract each component from the project

Step -3 Calculate the sensitivity of component as in 2.1

Step -4 Identify the severity type of component as in 2.2

Step -5 Calculate the critical value as in 2.3

Step -6 Repeat the steps 2-6 until all the components are extracted

Step -7 Analyse critical value of components

Step -8 Prioritize the critical components based on critical value

Step -9 List out the critical components

4. Implementation of the Proposed Approach

The proposed approach has been implemented as a tool in Java. The Screen shots of the tool are shown in the Appendix

A. To illustrate the working principle of the proposed approach, a case study (Hospital Management System) is demonstrated here.

4.1 Case Study – Hospital Management System

i) Component Extraction

In this section, we extracted all the Components involved in the SUT, which we taken as Case Study. While extracting the Components, we found the connected Components of each Component.

The Components in the SUT are listed in the following table:

TABLE – 4 COMPONENTS IN SUT

S. No	Components
1	Appointment
2	Bills
3	Diagnosis
4	DietInfo
5	Doctor
6	Injection
7	Inpatient
8	Labtech
9	Medicine
10	Nurse
11	Operation
12	Outpatient
13	Patient
14	Preport
15	Register
16	Staff
17	Test
18	Treatments
19	Ward
20	Wardallot

In the case study we taken, there are 20 components available. The components details are keep in repository and later they'll be used for Criticality Analysis.

ii) Sensitivity Analysis

In this Section, we calculate the metric values that are related to Sensitivity Analysis for each of the Component in the SUT. The calculated metrics are used to find out the Sensitivity of each Component.

The Components and their related Sensitivity metrics are listed in the following table:

TABLE – 5 SENSITIVITY METRICS OF COMPONENTS IN SUT

N o	Class	F I	F O	CC	IF	WM	RPI M
1	Inpatient	5	2	3	100	1.410667	0
2	Treatments	3	3	6	81	1.265312	0
3	Operation	2	3	7	36	1.864696	0
4	Appointment	2	2	11	16	1.07495	0
5	Diagnosis	1	4	10	16	0.600511	0
6	Staff	2	2	8	16	1.032447	0
7	DietInfo	2	1	8	4	1.100059	0.33
8	Register	2	1	8	4	0.511687	0
9	Wardallot	2	1	5	4	2.415961	0
10	Injection	1	1	10	1	0.94327	0
11	Bills	1	2	5	4	1.115325	0
12	Doctor	1	1	7	1	1.180669	0
13	Patient	0	2	8	0	1.618415	0
14	Medicine	1	1	6	1	1.04762	0
15	Test	1	1	6	1	0.791302	0
16	Preport	1	1	5	1	1.091804	0
17	Outpatient	1	0	6	0	0.915448	0
18	Labtech	1	1	3	1	2.005556	0
19	Nurse	1	1	3	1	1.186275	0
20	Ward	1	1	3	1	1.313636	0

From this, we found 6 different kinds of Sensitivity metrics related to each component of the Case Study we taken. These metrics are analysed and its ratio is taken in account for calculating the Critical values.

iii) Severity Analysis

In this section, we find severity level of each Component using the methodology proposed by Vahid Garousi [9]. Based on this, each component is categorized to various severity types and corresponding severity values are assigned. The

components with higher severity value are identified to take the necessary action that avoids the consequences of failure.

The Components and their related Severity Values are listed in the following table:

TABLE – 6 SEVERITY VALUES OF COMPONENTS

S. No	Class	Severity Type	Severity Value
1	Treatments	Catastrophic	0.95
2	Appointment	Critical	0.75
3	Doctor	Critical	0.75
4	Injection	Critical	0.75
5	Patient	Critical	0.75
6	Staff	Critical	0.75
7	Wardallot	Critical	0.75
8	Register	Marginal	0.5
9	Test	Marginal	0.5
10	Bills	Minor	0.25
11	Diagnosis	Minor	0.25
12	DietInfo	Minor	0.25
13	Inpatient	Minor	0.25
14	Labtech	Minor	0.25
15	Medicine	Minor	0.25
16	Nurse	Minor	0.25
17	Operation	Minor	0.25
18	Outpatient	Minor	0.25
19	Preport	Minor	0.25
20	Ward	Minor	0.25

From this, we group the components into 4 different categories of Severity. Each category is assigned a value as 0.95, 0.75, 0.50 and 0.25. The ratio of each component Severity value is taken in account for calculating the Critical values.

iv) Criticality Calculation

Based on the Severity Analysis and Sensitivity Analysis, we calculate the Criticality of each Component using the following formula:

$$CI(C_i) = P(CV(C_i)) * P(SV(C_i)) * P(E(C_i)) * \text{Time-taken}(C_i)$$

After calculating the Criticality Index (CI) of each Component, we prioritize those Components as a list based on their Critical values. That prioritize list is shown in the following table:

TABLE – 7 CRITICALITY INDEX OF COMPONENTS

S. No	Class Name	Criticality Index (CI)	Normalized Criticality Index (CI * 100)
1	Appointment	0.118256	11.825
2	Treatments	0.084236	8.423
3	Inpatient	0.064874	6.487
4	Operation	0.016855	1.685
5	Diagnosis	0.009383	0.938
6	Staff	0.006927	0.692
7	Register	0.005622	0.562
8	Injection	0.004587	0.458
9	Wardallot	0.003336	0.333
10	DietInfo	0.002811	0.281
11	Test	0.002780	0.278
12	Doctor	0.002259	0.225
13	Bills	0.001869	0.186
14	Nurse	0.000764	0.076
15	Medicine	0.000469	0.046
16	Patient	0.000359	0.035
17	Outpatient	0.000324	0.032
18	Preport	0.000231	0.023
19	Ward	0.000173	0.017
20	Labtech	0.000094	0.009

From the above table, we found that the Component “Appointment” has the higher critical value than other Components. So, it is said to be one of the fault-prone Components of the SUT we taken. These results are matching with real-time usage of that Software under Test (SUT).

On taking 0.65 as the selection point for critical component, it has been identified that the following components are high critical in the case study we taken.

TABLE – 8 FAULT-PRONE COMPONENTS LIST

S. No	Class	Criticality Index (CI)	Normalized Complexity (CI * 100)
1	Appointment	0.118256	11.825
2	Treatments	0.084236	8.423
3	Inpatient	0.064874	6.487
4	Operation	0.016855	1.685
5	Diagnosis	0.009383	0.938
6	Staff	0.006927	0.692

III. CONCLUSION

Fault-prone components will have major impact on the Software. The proposed approach has been automated to extract the components from the software and analyse their criticality. Based on this analysis, our approach is able to identify the fault-prone components of the software. The other approaches related to this work use a variety of metrics to identify the fault-prone components of the software. But our approach depicts that using the known metrics such as Fan-In, Fan-Out, Weightage of Methods in a class, Weakness of Methods and Ratio of Pure Inherited Methods metrics is enough to identify the fault-prone components. In addition to the metrics, our approach analyses the severity of components, which leads to an efficient way of identifying the fault-prone components.

Though identification of fault-prone components is an important task, testing of those identified fault-prone components rigorously is most important. This work can be extended to test them based on component coverage. This will make this as an automated approach to identify and verify the fault-prone components of any real-time complex software in an efficient way.

IV. ACKNOWLEDGEMENT

The proposed approach is a part of the UGC Major Research Project entitled, “Critical Component Identification and Verification for Real Time Complex Systems using Artificial Bee Colony based Approach”

REFERENCES

- [1] Roger S. Pressman, Software Engineering, A Practitioner's Approach, 5th ed. McGraw Hill, 1997.
- [2] Ohlsson, Niclas, M. Helander, and C. Wohlin, "Quality improvement by identification of fault-prone modules using software design metrics", in Proc. International Conference on Software Quality., pp. 1-13, 1996.
- [3] Hendry, Sallie, Kafura Dennis, "Software Structure Metrics Based on Information Flow," IEEE Transactions on Software Engineering., Vol. 7, No. 5, pp. 510-518, 1981.
- [4] Christof Ebert, "Classification techniques for metric-based Software development," Software Quality Journal., Vol.5, No.4, pp. 255-272.
- [5] Malhotra Ruchika, Jain Ankita, "Fault Prediction Using Statistical and Machine Learning Methods for Improving Software Quality," Journal of Information Processing Systems., Vol.8, No.2, pp. 241- 262, 2012.
- [6] Shatnawi A Raed, Li Wei, "The Effectiveness of Software Metrics in Identifying Error-Prone Classes in Post-Release Software Evolution Process," Journal of Systems and Software., Vol. 81, pp.1868 – 1882, 2008.
- [7] Czerwonka, Jacek, Rajiv Das, Nachiappan Nagappan, Alex Tarvo, and Alex Teterev, "Crane: Failure prediction,

change analysis and test prioritization in practice-experiences from windows." In Software Testing,

Appendix – A

The Screenshots of the tool developed to automate the proposed approach are given below:

Fig.1. Component Extraction

Fig.2. Metrics Calculation

Fig.3. Critical Components

Fig.4. Critical Test Path

Fig.5. Connected Component Graph

Fig.6. Critical Components Graph