Assessment of Change Request Artifacts Impact towards Fault Proneness

M.Rudra Kumar[1] Dr.A.Ananda Rao[2]

[1]Annamacharya Institute of Technology and Sciences, Rajampet

[2]Professor,Director of IR&P,SCDE
[2]JNTU Anantapur,

Abstract:

Exploring the impact of change requests applied on a software maintenance project helps to forecasts the fault-proneness of the change request to be handled further, which is either a bug fix or a new feature request. In practice the major development community stores change requests and related data using bug tracking systems such as bugzilla. These data, together with the data stored in a versioning system, such as Concurrent Versioning Systems, are a valuable source of information to create descriptions and also can perform useful analyses. In this paper we propose a novel knowledge based approach to assess the impact of the change request by the Change Request artifacts derived in our earlier work. The proposed model can be labeled as Assessment of Change Request Impact towards Fault Proneness (CRAI2FP). The method CRAI2FP exploits information retrieval algorithms to identify the influenced part of the code, architecture, modules and structure against the devised change request. And further evaluates the change impact value of the Change Request Artifacts towards fault proneness. The proposed method is evaluated by applying on known open source projects concurrent versioning and Change request logs.

Keywords: *Defect forecasting, product metrics, change request, artifacts, concurrent versioning system, fault proneness, SDLC, risk prediction*

1. Introduction:

The present open source projects are letting to access the version histories under free of cost, which is volume wise very high. This version history helps to extract the information regarding the progress of stages and strategies of that project development scenario, also provides information of the time and resource related to a change acquired. In recent literature related to software engineering and development, we can observe the extended role of this version history. Few of such developments are, using to access the change proliferation [1]; examining the impact of the bugs [2], accessing complexities of software [3], and also can use to access the reusability[4][5].

The said issues [1][2][3][4][5] issues usually raised due to analyzing the "outcome of the development" instead of "process of the development". In related to this, the research work devised in [2] concluded that fault proneness is proportional to the count of code changes applied. The research article [1] devised a strategy that extracts patterns from changes registered in version history and the same used to recognize the tuples of the code need to be modified in related to a modification required. In this regard in our earlier effort we defined chain of change request artifacts [24]. Further in this paper we propose a novel statistical approach to assess the impact of change request towards fault proneness. In this regard a change request artifact impact analysis model is devised. In preprocess stage we extract the effected dependencies, architecture, inheritance levels, sources and structure against change, which is by using information retrieval techniques. We use the development history log managed by the CVS [6], which is one of the popular product related to versioning system, and also we consider a bug tracking system called Bugzilla [7]. The main contributions of the proposed Change Request Assessment towards fault proneness are:

- Extracting modules, dependencies, architecture, inheritance levels, sources and structure at preprocessing level, which is using information retrieval technique.
- Assessing the impact of Change Request Artifacts such as
  - Dependency relation change impact
  - Structure Change Impact
  - Sources change impact
  - Inheritance change Impact
  - component or object Coupling change impact
- And further Change Request Impact towards fault proneness will be assessed

2. Related Work

The classification schemes with characteristics described in following listing are used in general to classify the impacts of the change requests, which in turn help to estimate the scope of a risk due to the requests related to software.

- o Concluding the hazards connected with change request and recognizing the possibility of considering change request.
- o Letting to categorize changes by depend on divergent decisive factors such as the change basis, change form, the influential area of the change, and the change influence.
- o Letting to devise common process to handle changes that categorized as analogous [8].

The work devised in [9] attempted to trace the occurrence ratio of the divergent maintenance activities usually practice by the software development communities. The research article [10] also related to the same idea. With assessment of these efforts [9][10] the changes categorically identified as a change related to the request of correcting an issue that went wrong, adapting a service or resource that missed, perfecting the service and preventing the possible pitfalls. The changes considered during the life cycle of the software development are categorized as changes related to Perfection. In general these changes are centric towards attaining perfection in requirements devised. The change requests related to noticed bugs attain the demand of correction. The change requests related to environmental and other functional issues such as version compatibility, component compatibility categorically concluded as adaptive change requests [10]. Change requests related to rectifying the instable states noticed in given software categorized as preventive [11].

The change request process flow listed below is devised in research article [12]:

1. Need of the change that requested should be formalized
2. Assess viability and consequences of the requested change
3. Trace and allocate the desired resources to assess and implement the said change request
4. Devise a strategy to handle the requested change
5. Devise a methodology to apply the strategy explored in previous step
6. Commence to handle the requested change

The consequences of changes applied on inheritance structure were analyzed in [13]. The changes that lead to consequences related to process and structure of the system were categorically identified in [12].

The process of fault prediction through the analysis of dependencies was devised in [15]. The model explored in [15] is able to recognize the proportionality between dependencies and faults. In this regard dependency structure devised in [14] is taken into consideration. As plotted in [14], the syntactic dependencies is category of product metrics that are related to direct dependencies, and product metrics related to transitive coupling are comes under rational dependencies. By considering these categories of process metrics, the model devised in [15] able to explore the proportionality between these process metrics and bugs. The empirical study found in [15] concluding that transitive dependencies are more fault prone compared to direct dependencies. In turn the same empirical study confirming that alone product metrics are not significant to bug forecasting and influences of the changes related to bugs fix and enhancement.

A review of all empirical studies from 1995 to 2010 to predict software fault-proneness with a specific focus on techniques used is explored in [23]. A machine learning model devised in [22] to discover the association among OO metrics such as CK-Metrics and fault-proneness and its severity. In this regard the model devised in [22] is using the logistic regression to define the relation between OO-metrics and fault-proneness. The results were analyzed using open source software. Further, the performance of the predicted models was evaluated by ROC analysis. Researchers have successfully applied fuzzy logic in software engineering disciplines such as effort estimation, project management. In this regard Handa et al [20] devised Fuzzy Logic for software metrics to predict the fault proneness. Chandra et al [19] devised an empirical study to evaluate the proportionality among MOOD metrics and quality of the product.

However, tracing the influenced sections due to requested change consequences is intricate. The model referred as Static analysis [18] is said to be extracts false positives and demands countable additional computation time. The other analytical model called dynamic analysis [19] able to confines bound areas in adaptive manner, but it often fails to recognize infrequently used but affected areas. In practical, the dynamic analysis is not adaptable more often. A model devised in [20] able to trace the effected sections related to the given change request, but it performs only by prior information of the module to which the requested change is related. The prior knowledge of the module that related to given change leads to raise the complexity to determine the affected sections with minimal false positives.

Henceforth here in this paper we propose a novel statistical approach to assess the impact of change request towards fault proneness, which is measuring based on the impacts of effected change request artifacts devised in our earlier work [24].

3. Assessing Change Request Impact Towards Fault Proneness

Here in this section we describe the proposed Assessment approach of Change Request Impact towards fault Proneness. Initially preprocessing will be done to identify the change request artifacts influenced by the given change request. In this regard an information retrieval technique will be used, which is described in following sub section.

3.1 The Task

The maintenance phase of software life cycle is critical as it deals with potential change requests. The updates applied to the software against to these change requests may leads to fault proneness. In particular, after considerable number potential changes made the fault proneness of further change requests increases. Henceforth, a practice of forecasting the possible fault proneness of a change request is worthy.

3.2 The approach

A learning strategy is adapted in our proposed model that attempts to forecast the fault proneness of a change request made. In this process the devised model calculates the weights of the change request artifacts (see figure1 for listing of those artifacts), which is based on the change impacts observed against to earlier change requests.

3.3 The input source of the process

A software tool such as bugzilla [2] is used to handle change requests. Any authorized individual involved with that software can request a change. In common, the structure of a change request in any of such tools contains long and short descriptions. These descriptions are taken as primary input to identify the requested change. The versioning systems such as concurrent versioning systems (CVS) [6] are used to log the every event (such as the details of lines of code modified, modified by whom and when) occurred during the software development and updates. The descriptions available in these versioning systems are taken as input to estimate the impacts of earlier change requests considered and applied.

3.4 Extracting change request type and related change request artifacts:

Extract short and long descriptors of the change request then apply text processing steps such as

- Tokenizing: split the short and long descriptors in to words
- Stop word removal: remove stop words such as the, and, of, a….
- Stemming: remove tense and ing-forms from the words
- And eliminating the duplicates: remove duplicate words and explore the final attributes labeled as descriptive tokens.

Then extract the work descriptors from versioning system and rank these descriptors in descending order according to frequency of the descriptive tokens. The descriptor with highest frequency of descriptive tokens will be ranked high.

Depend on the highly compatible descriptors, explore the change request artifacts figured in fig 1

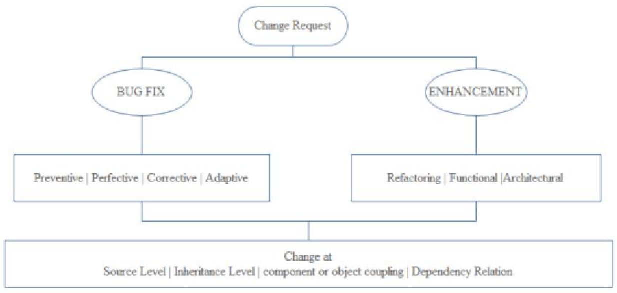
Change Request
BUG FIX
ENHANCEMENT
Preventive | Perfective | Corrective | Adaptive
Refactoring | Functional |Architectural
Change at
Source Level | Inheritance Level | component or object coupling | Dependency Relation

Figure 1: The Change request artifacts and their connected flow [24]

Top Level Artifact is “motivation of a change” and it represents either “new feature” or “bug fix”.

The second level artifacts of “feature request” are explored here

1. Refactor
2. Functional
3. Architectural

The second level Artifacts of “Bug Fix” are explored here:

1. Preventive
2. Perfective
3. Corrective
4. Adaptive

The third Level Artifacts that are common for all second level artifacts are listed here:

1. Changes in Source code
2. Changes in Inheritance levels
3. Changes in component or Object coupling
4. Changes in dependency relations
5. Decomposition of Structure

3.5 Estimating the artifacts fault prone weights:

3.5.1 First Level

The first level artifact weights have been measured as number of revisions occurred due to the faults emerged against for each change request.

Let crb_i is the i^{th} bug fix request and $rc(crb_i)$ is the revision count of crb_i, which is ≥ 1 represents the number of times software update occurred while fixing the bug represented by crb_i. The set of bug fix requests found in the change request log can be referred as

$$CR_b = \{crb_1, crb_2, ...crb_i, crb_{i+1}, ...crb_n\}$$

And set of software update count related to CR_b can be referred as

$$RCCR_b = \{rc(crb_1), rc(crb_2), ...rc(crb_i), rc(crb_{i+1}), ...rc(crb_n)\}.$$

In similar manner, let crf_i is the i^{th} change request related to new feature and $rc(crf_i)$ is the number revision occurred due to defects emerged while adding new feature request related to crf_i. The set of feature enhancement related change requests found in change-request-log can be referred as

$$CR_f = \{crf_1, crf_2, ..., crf_i, crf_{i+1}, ...crf_n\}$$

And set of software update count related to CR_f can be referred as

$$RCCR_f = \{rc(crf_1), rc(crf_2), ...rc(crf_i), rc(crf_{i+1}), ...rc(crf_n)\}.$$

The fault prone weight of each change request (first level artifacts) will be defined as follows:

Since the increased number of software update revisions indicates the increased fault prone weight, therefore the fault prone weight of crb_i and crf_i can be measured respectively as

$fpw(crb_i) = 1 - \dfrac{1}{rc(crb_i)}$ (Fault prone weight of the bug fix)

$fpw(crf_i) = 1 - \dfrac{1}{rc(crf_i)}$ (Fault prone weight of the feature enhancement)

And the average fault prone weight of the bug fix change request $fpw(CR_b)$ and feature enhancement change request $fpw(CR_f)$ can be measured respectively as follows

$$fpw(CR_b) = \frac{\sum_{i=1}^{n} fpw(crb_i)}{|CR_b|}$$

$$fpw(CR_f) = \frac{\sum_{i=1}^{n} fpw(crf_i)}{|CR_f|}$$

3.5.2 Second Level Artifact fault proneness Weights

The fault prone weight of bug fix second level artifacts will be defined as follows:

In a similar approach has been applied to find the fault prone weights of the second level artifacts, description is follows

Let $CR_b(pr) = \{crb_1, crb_2, ..., crb_i, crb_{i+1}, ..., crb_n\}$ is the set bug fix requests comes under the preventive artifact and $RCCR_b(pr) = \{rc(crb_1), rc(crb_1), ..., rc(crb_i), rc(crb_{i+1})....rc(crb_n)\}$ is the set of revision counts occurred for respective change request of preventive artifact

The $fpw_{(pr)}(crb_i)$ can be found as follows:

$$fpw_{(pr)}(crb_i) = 1 - \frac{1}{\{rc(crb_i) \forall rc(crb_i) \in RCCR_b(pr)\}}$$

The average fault prone weight of the bug fix requests under second level preventive artifact can be measured as follows:

$$fpw(CR_b(pr)) = \frac{\sum_{i=1}^{n} \{fpw_{(pr)}(crb_i) \mid crb_i \in CR_b(pr)\}}{|CR_b(pr)|}$$

And the similar way can be considered to measure the fault prone weight of the bug fix requests comes under other second level artifacts, the description follows

The fault prone weight of bug fix request crb_i that comes under perfective artifacts can be referred as $fpw_{(pe)}(crb_i)$, and the it can be measured as

$$fpw_{(pe)}(crb_i) = 1 - \frac{1}{rc(crb_i)}$$ such that $crb_i \in CR_b(pe)$ and $rc(crb_i) \in RCCR_b(pe)$

And the average fault prone weight of the bug fix requests under second level perfective artifact

$$fpw(CR_b(pe)) = \frac{\sum_{i=1}^{n}\{fpw_{(pe)}(crb_i) \mid crb_i \in CR_b(pe)\}}{|CR_b(pe)|}$$

The fault prone weight of bug fix request crb_i that comes under corrective artifacts can be referred as $fpw_{(co)}(crb_i)$, and the it can be measured as

$$fpw_{(co)}(crb_i) = 1 - \frac{1}{\{rc(crb_i) \forall rc(crb_i) \in RCCR_b(co)\}}$$

And the average fault prone weight of the bug fix requests under second level corrective artifact

$$fpw(CR_b(co)) = \frac{\sum_{i=1}^{n}\{fpw(crb_i) \mid crb_i \in CR_b(co)\}}{|CR_b(co)|}$$

The fault prone weight of bug fix request crb_i that comes under adaptive artifacts can be referred as $fpw_{(ad)}(crb_i)$, and it can be measured as

$$fpw_{(ad)}(crb_i) = 1 - \frac{1}{\{rc(crb_i) \forall rc(crb_i) \in RCCR_b(ad)\}}$$

And the average fault prone weight of the bug fix requests under second level perfective artifact

$$fpw(CR_b(ad)) = \frac{\sum_{i=1}^{n}\{fpw(crb_i) \mid crb_i \in CR_b(ad)\}}{|CR_b(ad)|}$$

The fault prone weight of second level artifacts of "feature enhancement" will be defined as follows:

In a similar approach that described for bug fix second level artifacts has been applied to find the fault prone weights of the second level artifacts of feature enhancement, description is follows

Let $CR_f(rf) = \{crf_1, crf_2, ..., crf_i, crf_{i+1}, ..., crf_n\}$ is the set of feature requests comes under the refactor artifact and

$$RCCR_f(rf) = \{rc(crf_1), rc(crf_1), ...,$$
$$......, rc(crf_i), rc(crf_{i+1}) .... rc(crf_n)\}$$

is the set of revision counts occurred for respective change request of refactor artifact

The $fpw_{(rf)}(crf_i)$ can be found as follows:

$$fpw_{(rf)}(crf_i) = 1 - \frac{1}{\{rc(crf_i) \forall rc(crf_i) \in RCCR_f(rf)\}}$$

The average fault prone weight of the feature requests under second level refactor artifact can be measured as follows:

$$fpw(CR_f(rf)) = \frac{\sum_{i=1}^{n}\{fpw_{(rf)}(crf_i) \mid crf_i \in CR_f(rf)\}}{|CR_f(rf)|}$$

And the similar way can be considered to measure the fault prone weight of the feature requests comes under other second level artifacts, the description follows

The fault prone weight of feature request crf_i that comes under functional artifacts can be referred as $fpw_{(fu)}(crf_i)$, and it can be measured as

$$fpw_{(fu)}(crf_i) = 1 - \frac{1}{\{rc(crf_i) \forall rc(crf_i) \in RCCR_f(fu)\}}$$

And the average fault prone weight of the feature requests under second level functional artifact

$$fpw(CR_f(fu)) = \frac{\sum_{i=1}^{n}\{fpw_{(fu)}(crf_i) \mid crf_i \in CR_f(fu)\}}{|CR_f(fu)|}$$

The fault prone weight of feature request crf_i that comes under architectural artifact can be referred as $fpw_{(ar)}(crf_i)$, and the it can be measured as

$$fpw_{(ar)}(crf_i) = 1 - \frac{1}{\{rc(crf_i) \forall rc(crf_i) \in RCCR_f(ar)\}}$$

And the average fault prone weight of the feature requests under second level architectural artifact

$$fpw(CR_f(ar)) = \frac{\sum_{i=1}^{n}\{fpw(crf_i) \mid crf_i \in CR_f(ar)\}}{|CR_f(ar)|}$$

In a glance

- $fpw(CR_b(pr)), fpw(CR_b(pe)), fpw(CR_b(co))$ **and** $fpw(CR_b(ad)$ are fault prone weights of the second level bug fix artifacts called preventive, perfective, corrective and adaptive respectively,
- and $fpw(CR_f(rf)), fpw(CR_f(fu)),$ **and** $fpw(CR_f(ar))$ are fault prone weights of the feature request second level artifacts called refactor, functional, architectural respectively.

3.5.3 Third level artifact fault proneness weights

The model devised to measure the fault proneness weights of the first and second level artifacts can be considered to find fault prone weights for third level artifacts. Not like in second level artifacts, the third level artifacts are same for all change requests called bug fix and feature enhancement. Hence forth the fault proneness weights measures under the context of all types of change requests.

Let CR be the set of change requests representing $\{cr_1, cr_2, ..., cr_i, cr_{i+1}, ...cr_n\}$. And $RCCR_{sc}, RCCR_{iht}, RCCR_{oc}, RCCR_{dr}, RCCR_{str}$ are representing set of revisions of source code updates ($RCCR_{sc}$), inheritance updates ($RCCR_{iht}$), object coupling updates ($RCCR_{oc}$), dependency relation updates and structure updates ($RCCR_{str}$) respectively.

The $RCCR_{sc}$ represents $\{rc_{sc}(cr_1), rc_{sc}(cr_2), ..., rc_{sc}(cr_i), rc_{sc}(cr_{i+1}), ..., rc_{sc}(cr_n)\}$, here $rc_{sc}(cr_i)$ indicates the set of source code update revisions required for change request cr_i .

The $RCCR_{iht}$ represents $\{rc_{iht}(cr_1), rc_{iht}(cr_2), ..., rc_{iht}(cr_i), rc_{iht}(cr_{i+1}), ..., rc_{iht}(cr_n)\}$, here $rc_{iht}(cr_i)$ indicates the set of inheritance update revisions required for change request cr_i .

The $RCCR_{oc}$ represents $\{rc_{oc}(cr_1), rc_{oc}(cr_2), ..., rc_{oc}(cr_i), rc_{oc}(cr_{i+1}), ..., rc_{oc}(cr_n)\}$, here $rc_{oc}(cr_i)$ indicates the set of object coupling update revisions required for change request cr_i .

The $RCCR_{dr}$ represents $\{rc_{dr}(cr_1), rc_{dr}(cr_2), ..., rc_{dr}(cr_i), rc_{dr}(cr_{i+1}), ..., rc_{dr}(cr_n)\}$, here $rc_{dr}(cr_i)$ indicates the set of dependency relation updates revisions required for change request cr_i .

The $RCCR_{str}$ represents $\{rc_{str}(cr_1), rc_{str}(cr_2), ..., rc_{str}(cr_i), rc_{str}(cr_{i+1}), ..., rc_{str}(cr_n)\}$, here $rc_{str}(cr_i)$ indicates the set of object coupling update revisions required for change request cr_i .

And then the fault proneness weight of each change request under third level artifacts (source code updates, inheritance update, object coupling update, dependency relation update and structural update) measures as follow:

Fault proneness weight of source code update artifact for change request cr_i is

$$fpw_{sc}(cr_i) = 1 - \frac{1}{\{rc_{sc}(cr_i) \forall rc_{sc}(cr_i) \in RCCR_{sc} \,\&\, rc_{sc}(cr_i) \geq 1\}}$$

Average fault proneness weight of source code update artifact for change request set CR is

$$fpw_{sc}(CR) = \frac{\sum_{i=1}^{n} \{fpw_{sc}(cr_i) \forall cr_i \in CR \,\&\, rc_{sc}(cr_i) \geq 1\}}{|CR|}$$

Fault proneness weight of inheritance update artifact for change request cr_i is

$$fpw_{iht}(cr_i) = 1 - \frac{1}{\{rc_{iht}(cr_i) \forall rc_{iht}(cr_i) \in RCCR_{iht} \,\&\, rc_{iht}(cr_i) \geq 1\}}$$

Average fault proneness weight of inheritance update artifact for change request set CR is

$$fpw_{iht}(CR) = \frac{\sum_{i=1}^{n} \{fpw_{iht}(cr_i) \forall cr_i \in CR \,\&\, rc_{iht}(cr_i) \geq 1\}}{|CR|}$$

Fault proneness weight of object coupling update artifact for change request cr_i is

$$fpw_{oc}(cr_i) = 1 - \frac{1}{\{rc_{oc}(cr_i) \forall rc_{oc}(cr_i) \in RCCR_{oc} \,\&\, rc_{oc}(cr_i) \geq 1\}}$$

Average fault proneness weight of object coupling update artifact for change request set CR is

$$fpw_{oc}(CR) = \frac{\sum_{i=1}^{n} \{fpw_{oc}(cr_i) \forall cr_i \in CR \,\&\, rc_{oc}(cr_i) \geq 1\}}{|CR|}$$

Fault proneness weight of dependency relation update artifact for change request cr_i is

$$fpw_{dr}(cr_i) = 1 - \frac{1}{\{rc_{dr}(cr_i) \forall rc_{dr}(cr_i) \in RCCR_{dr} \,\&\, rc_{dr}(cr_i) \geq 1\}}$$

Average fault proneness weight of dependency relation update artifact for change request set CR is

$$fpw_{dr}(CR) = \frac{\sum_{i=1}^{n} \{fpw_{dr}(cr_i) \forall cr_i \in CR \,\&\, rc_{dr}(cr_i) \geq 1\}}{|CR|}$$

Fault proneness weight of structural update artifact for change request cr_i is

$$fpw_{str}(cr_i) = 1 - \frac{1}{\{rc_{str}(cr_i) \forall rc_{str}(cr_i) \in RCCR_{str} \,\&\, rc_{str}(cr_i) \geq 1\}}$$

Average fault proneness weight of structural update artifact for change request set CR is

$$fpw_{str}(CR) = \frac{\sum_{i=1}^{n} \{fpw_{str}(cr_i) \forall cr_i \in CR \,\&\, rc_{str}(cr_i) \geq 1\}}{|CR|}$$

3.6 The Algorithmic approach of the CR assessment

- Extract the CR state from Change Request submitted
- Find CR-State is Enhancement or Bug
- If CR-State is Bug then find bug-state
  - If bug-state is not repetitive then
    - Extract short and long descriptions of the bug explored
    - Preprocess the descriptions to extract the descriptive tokens
    - Extract versioning system work descriptions by the descriptive token compatibility and rank them based on the descriptive token frequency
    - Find that bug state is preventive, corrective or adaptive by the ranked descriptions.
    - Find the influenced dependents impact
    - Find the influenced structure impact
    - Find the Influenced Sources impact
    - Find the influence Inheritance structure impact
    - Find the influence component or object coupling impact
  - Else if CR-State is Enhancement
    - Extract short and long descriptions of the Enhancement explored
    - Preprocess the descriptions to extract the attributes
    - Extract work sheet descriptions by the attribute compatibility and rank them based on their comparison weights
    - Find that enhancement state is functional, architectural or refactor by the ranked descriptions.
    - Find the influenced dependents impact
    - Find the influenced structure impact
    - Find the Influenced Source impact
    - Find the influence Inheritance structure impact
    - Find the influence component or object coupling impact

Further the dependents relation change impact (DRCI) can be found as follows:

$$DRCI = \frac{|cdr|}{|dr|} \text{ …. Eq 1.}$$

Here in this Eq 1, $|cdr|$ is number of dependency relations changed, and $|dr|$ is total number of dependency relations

The Inheritance change impact (HCI)can be measured as follows:

$$HCI = \frac{|chl| + |nhl|}{|hl|} \text{ …Eq 2.}$$

Here in Eq 2, The $|chl|$ is set changed inheritance levels, $|nhl|$ is set of new inheritance levels and $|hl|$ is set of all inheritance levels

$$OCHI = \frac{|cco| + |nco|}{|co|} \text{ … Eq 3.}$$

The Eq 3 is indicating the process of measuring Coupling Change Impact (CCI), here in this equation $|cco|$ indicates the set of changed couplings, $|nco|$ indicates the new couplings and $|co|$ indicates the all available couplings count.

Structure Update Impact (SUI) can be measured as follows:

$$SUI = 1 - \frac{1}{DRCI + HCI + OCHI} \text{ … Eq 4.}$$

$$SCCI = \frac{|csl| + |nsl|}{|sl|} \text{ … Eq 5.}$$

The Eq 5 indicates the process of measuring Source Code Change Impact (SCCI), here in eq 5, $|csl|$ is set of changed source lines, $|nsl|$ is set of new source lines and $|sl|$ is total number of source lines.

$$\begin{aligned} CRI'' &= SUI * fpw_{str}(CR) + DRCI * fpw_{dr}(CR) \\ &+ OCHI * fpw_{oc}(CR) + HCI * fpw_{iht}(CR) \quad \text{… Eq 6} \\ &+ SCCI * fpw_{sc}(CR) \end{aligned}$$

Here in Eq 6, CRI'' indicates the third level change request impact, which is the sum of impacts of various third level artifacts are multiplied by the respective fault proneness weights.

The feature enhancement change request impact CRI_f can measured as follows:

$$CRI'_f = 1 - \frac{1}{\{CRI''*(fpw_{rf}(CR_f) + fpw_{fu}(CR_f) + pw_{ar}(CR_f))\}}$$
… Eq 7

Here in Eq7 CRI'_f is the change request impact of second level artifacts

$$CRI_f = 1 - \frac{1}{CRI'_f * fpw(CR_f)} \text{ …. Eq 8}$$

Here in Eq 8 CRI_f indicates the feature enhancement change request impact.

The impact of the bug fix change request can be measured as
follows

$$CRI'_b = 1 - \frac{1}{\{CRI''*(fpw_{(pr)}(CR_b) + fpw_{(pe)}(CR_b) + fpw_{(co)}(CR_b) + fpw_{(ad)}(CR_b))\}}$$
… Eq 9

In Eq9 CRI'_b indicates the impact of second level bug fix artifacts

$$CRI_b = 1 - \frac{1}{\{CRI'_b * fpw(CR_b)\}} \text{ ….. Eq 10}$$

And finally equation determines the impact of a bug fix change request CRI_b

The Eq 7 is measuring the second level feature change request impact, the Eq 8 follows that measures first level feature change request impact. Here in Eq 7 CRI'' is multiplied by the sum fault proneness weights of all second level artifacts of the feature change request. In Eq 8 CRI' is multiplied by the fault proneness weight of the feature change request.

The Eq 9 is measuring the impact of second level change request artifacts of Bug Fix change request, The Eq 10 follows that measures change request impact of the bug fix change request. Here in Eq 9 CRI'' is multiplied by the sum fault proneness weights of all second level artifacts of the Bug Fix change request. In Eq 10 CRI' is multiplied by the fault proneness weight of the first level artifact called Bug Fix.

The CRI value can be used to access the impact of change request towards fault proneness, which is as follows

If $CRI < 0.33$ CR is accessed as not fault prone

If $CRI \geq 0.33 \&\& CRI < 0.66$ then CR is assessed as Moderate to fault prone

If CRI>0.66, CR is assessed as highly fault prone

4. Empirical Analysis

We conducted experiments on 4 years old change request log and concurrent versioning log of an in house maintenance project. We make sure the heterogeneity in the number of change requests and their impact in various SDLC stages considered for experiments. We measured accuracy of accessing the change request artifact impact towards the fault proneness as follows:

$$\mathrm{S(CRI)} = \frac{|\mathrm{CRI}|}{|\overline{CRI}|} \quad .... \text{ Eq 11}$$

Here in Eq 11 |CRI| indicates the change request impacts assessed by propose Statistical approach.

$|\overline{CRI}|$ Indicates the change request impacts that are actually observed in logs

Here in this empirical study we observe that proposed statistical model performs better in predicting the degree of change request impact towards fault proneness, which stands with an approximate value of above 70%.

4.1 Input Data characteristics

The data that we used to analyze the performance of the proposed Statistical Analysis of Change Request Impact towards Fault Proneness is the change requests made to selected in house maintenance project. The change requests maintained using BUGZILLA, and Work descriptors of the software updates against change requests maintained in concurrent versioning system. The work specifications are customized such that they reflect the second and third level artifacts used in this proposal. The first level artifact information is extracted from the Change Request descriptors. The performance analysis done on over 500 change requests, out of that majority of CRs related to first level change request artifact called Bug Fix.

4.2 Performance Analysis

We used fault forecasting accuracy (the percentage of conceptually valid faults by the proposed CRAI2FP) as the main performance measure. In addition to measuring fault forecasting accuracy, the precision, recall, and F-measure were used to measure the performance; these are defined using Eq12, Eq13 and Eq14.

$$pr = \frac{t_+}{t_+ + f_+} \quad ..... \text{ Eq12}$$

Here in Eq12 the pr indicates the precision, t_+ indicates the true positives and f_+ indicates the false positive

$$rc = \frac{t_+}{t_+ + f_-} \quad ..... \text{ Eq13}$$

Here in the Eq13, the 'rc' indicates the recall, 'f_-' indicates the false negative.

$$F = \frac{2 * pr * rc}{pr + rc} \quad ...... \text{ Eq14}$$

Here in the Eq14, 'F' indicates the F-measure.

Table 1: Precision, recall and F-measure values for faults actually rose in regard to change requests, faults proneness actually forecast by proposed CRAI2FP.

	Precision	Recall	F-Measure
Faults proneness actually rose against Change Requests	0.9976	0.9983	0.998
Fault Proneness forecast by CRAI2FP	0.9972	0.9979	0.99725

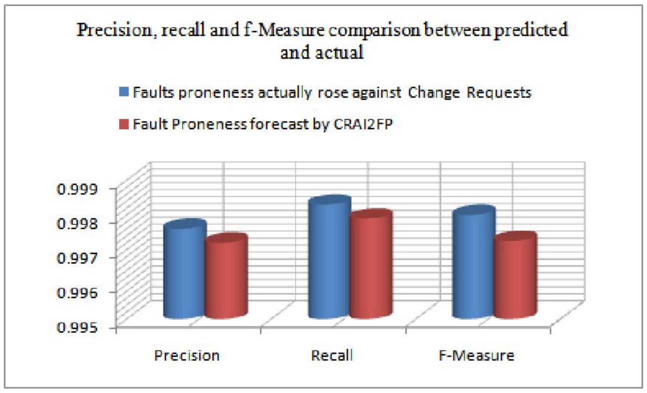

Fig 1: The comparison bar chart of the Precision, recall and F-measure values for faults actually rose in regard to change requests, faults proneness actually forecast by proposed CRAI2FP.

5. Conclusion:

The work described in this paper is a knowledge based approach to assess the impact of change request artifacts impact towards fault proneness, in short referred as

CRAI2FP. The model devised here is using the Change Request Artifacts devised in our earlier work [24]. The state of fault proneness of the change request is forecasted by the proposed model with around more than 70% accuracy, which is explored by accessing accuracy $S(CRI)$, precision pr, recall rc and f-measure F in experimental study. This work motivates us to further research towards developing mining approaches to access the change request impact towards the fault proneness.

References

[1] A. T. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll. Predicting source code changes by mining revision history. IEEE Transactions on Software Engineering, 30:574–586, Sept. 2004. [24] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. InICSE '04: Proceedings of the 26th International Conference on Software Engineering, pages 563–572. IEEE Computer So-ciety, 2004.

[2] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy. Predicting fault incidence using software change history. IEEE Trans. Softw. Eng., 26(7):653–661, 2000.

[3] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus. Does code decay? assessing the evidence from change management data.IEEE Trans. Softw. Eng., 27(1):1–12, 2001.

[4] A. Michail. Data mining library reuse patterns using gen-eralized association rules. InICSE '00: Proceedings of the 22nd international conference on Software engineering, pages 167–176. ACM Press, 2000.

[5] L. Lpez, J. Gonzlez-Barahona, and G. Robles. Applying so-cial network analysis to the information in cvs respositories. In IEEE 26th International Conference on Software Engi-neering - The International Workshop on Mining Software Repositories, 2004.

[6] Cvs. concurrent versions system. http://www.cvshome.org/.

[7] Bugzilla. bug tracking system. http://www.bugzilla.org/.

[8] N. Nurmuliani, D. Zowghi, and S. P. Williams. "Using Card Sorting Technique to Classify Requirements Change," in Proceedings of the 12th IEEE International Requirements Engineering Conference, 2004, pp. 240-248.

[9] B. Lientz and B. Swanson, Software Maintenance Management Addison-Wesley, 1980

[10] I. Sommerville, Software Engineering. 7th ed: Addison-Wesley, 2004

[11] P. Mohagheghi and R. Conradi. "An Empirical Study of Software Change: Origin, Acceptance Rate, and Functionality Vs. Quality Attributes," in Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE '04), 2004, pp. 7- 16.

[12] J. Nedstam, E. A. Karlsson, and M. Host. "The Architectural Change Process," in Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE '04), 2004, pp. 27-36.

[13] D. Kung, J. Gao, P. Hsia, F. Wen, Y. Toyoshima, and C. Chen. "Change Impact Identification in Object Oriented Software Maintenance," in Proceedings of the International Conference on Software Maintenance, Victoria, BC, 1994, pp. 202-211.

[14] Gall, H., Hajek, K., and Jazayeri, M., "Detection of rational coupling based on product release history," IEEE Int'l Conf. on Softw. Maint. ICSM, pp.190-198, 1998.

[15] Cataldo, M., Mockus, A., Roberts, J. A., and Herbsleb, J. D., "Software dependencies, work dependencies, and their impact on failures," IEEE Trans. Softw. Eng. 36, 2, pp.864-878, 2009.

[16] Zimmermann, T., and Nagappan, N., "Predicting defects using network analysis on dependency graphs," Int'l Conf. on Softw. Eng. ICSE, pp.531-540, 2008.

[17] Kobayashi, K.; Matsuo, A.; Inoue, K.; Hayase, Y.; Kamimura, M.; Yoshino, T.; , "ImpactScale: Quantifying change impact to predict faults in large software systems," Software Maintenance (ICSM), 2011 27th IEEE International Conference on , vol., no., pp.43-52, 25-30 Sept. 2011; doi: 10.1109/ICSM.2011.6080771

[18] Bohner, S. A., and Arnold, R. S. (Eds.), "Software change impact analysis," Bohner, S. A. and Arnold, R. S., "An introduction to software change impact analysis," IEEE Computer Society Press, pp.1-26, 1996.

[19] Chandra, E. and Linda, P.E. 2010. Assessment of software quality through object oriented metrics. CIIT Int. J. Software Engg. 2: 2.

[20] Handa, A. and Wayal, G. 2012. Software quality enhancement using Fuzzy logic with object oriented metrics in design. Int. J. Comp. Engg. Technol. (IJCET). 3(1): 169-179.

[21] Malhotra, R. 2012. A defect prediction model for open source software. Proc. of the World Congress on Engineering. Vol. II. July 4-6. London (UK).

[22] Malhotra, R., Kaur, A. and Singh, Y. 2010. Empirical validation of object-oriented metrics for predicting fault proneness at different severity levels using support vector machines. Int. J. Syst. Assurance Engg. Management. 1(3): 269-281.

[23] Saxena, P. and Saini, M. 2011. Empirical studies to predict fault proneness: A review. Int. J. Computer Appl . 22(8): 41-45.

[24] Rudra Kumar Madapudi, Ananda A Rao and Gopichand Merugu. Article: Change Requests Artifacts to Assess Impact on Structural Design of SDLC Phases. International Journal of Computer Applications 54(18):21-26, September 2012. Published by Foundation of Computer Science, New York, USA

[25] Mrinal Singh Rawat, Sanjay Kumar Dubey; Software Defect Prediction Models for Quality Improvement: A Literature Study;IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 5, No 2, September 2012; ISSN (Online): 1694-0814; www.IJCSI.org