

**Abstract**

The Cr$_3^+$ substituted Ni-Co alloys with the chemical formula Co$_0.5$Ni$_0.5$Cr$_x$Fe$_{2-x}$O$_4$ were synthesized by sol-gel auto combustion method. The synthesized samples were annealed at 600°C for 4 h. X-ray diffraction data were used to evaluate the structure of the prepared samples. Lattice constant (a), X-ray density (dx), bulk density (db), hopping length (L), allied parameters were calculated using X-ray diffraction data.

The prepared samples show the formation of single phase cubic spinel structure without any impurity peaks. Lattice constant and X-ray density found to decrease with Cr$_3^+$ substitution. Hopping lengths and allied parameters also decreases with Cr$_3^+$ substitution.

Keywords: Ni-Co alloys; Sol-gel method; XRD

1. **Introduction**

As important magnetic materials, Ni-Co alloys have potential application in high-density magnetic recording media, ferrofluids technology, magnetic resonance imaging, high-temperature catalysis, microwave absorber and electromagnetic shielding material. Recently, the preparation methods and applications of one-dimensional (1D) Ni-Co alloy materials have been paid to more attention due to their unique properties of low Dimensionality [1]. Nanocrystalline magnetic alloys such as Ni-Cu, Co-Ni, Fe-Co and Fe-Co-Cu are promising candidates for hyperthermia and microwave applications because their magnetic transition temperatures and microwave absorbance characteristics vary with composition and grain size [2-6].

In our previous study we have observed some significant changes in the ferrite materials brought by Cr$_3^+$ ions [7,8]. Therefore, in the present study we have focused to investigate hopping length and allied parameter of Cr$_3^+$ substituted Ni-Co alloys with the chemical formula Co$_{0.5}$Ni$_{0.5}$Cr$_x$Fe$_{2-x}$O$_4$.

2. **Experimental**

The powders were synthesized by sol-gel auto-combustion. Analytical grade citric acid (C$_6$H$_{12}$O$_7$·2H$_2$O), nickel nitrate (Ni(NO$_3$)$_2$·6H$_2$O), cobalt nitrate (Co(NO$_3$)$_2$·3H$_2$O) and iron nitrate (Fe(NO$_3$)$_3$·9H$_2$O) were used as starting materials. Reaction procedure was carried out in air atmosphere without protection or inert gases. The molar ratio of metal nitrates to citric acid was taken as 1:3. The metal nitrates were dissolved together in a minimum amount of double distilled water to get a clear solution. An aqueous solution of citric acid was mixed with metal nitrates solution, then ammonia
solution was slowly added to adjust the pH at 7. The mixed solution was kept on a hot plate with continuous stirring at 90 °C. During evaporation, the solution became viscous and finally formed a very viscous brown gel. When finally all water molecules were removed from the mixture, the viscous gel began frothing. After few minutes, the gel automatically ignited and burnt with glowing flames. The decomposition reaction would not stop before the whole citrate complex was consumed. The auto-combustion was completed within a minute, yielding the brown-colored ashes termed as a precursor. The as-prepared powder then annealed at 600 °C for 4 h. The samples were powdered for X-ray investigations. Part of the powder was X-ray examined by Phillips X-ray diffractometer (Model 3710) using Cu-Kα radiation (λ=1.5405Å).

3. Results and discussion

Figure 1 shows the X-ray diffraction pattern of the typical sample x = 0.5. The XRD patterns exhibit peaks corresponding to Cr3+ substituted Ni-Co ferrites and the absence of any other impurity phases.

![Fig. 1: Typical XRD pattern (x = 0.5) of Co0.5Ni0.5Cr0.5Fe2-xO4](image)

The lattice parameter 'a' was calculated using the following equation [9],

\[ a = d \sqrt{h^2 + k^2 + l^2} \]

where, \( d \) is the inter-planer spacing and \((hkl)\) is the index of the XRD reflection peak. It is observed that lattice constant 'a' decreases from 8.352-8.331 Å with increase in Cr3+ substitution in Ni-Co alloys. The decrease in lattice constant is related to the ionic radii of the respective ions. In the present ferrite system smaller Cr3+ of 0.63 Å ionic radii replaces larger Fe3+ ions with 0.67 Å ionic radii.

![Fig. 2: Variation of X-ray density with Cr content x.](image)
The X-ray density \( (dx) \) of all the samples of the series was obtained by the following relation.

\[
dx = \frac{ZM}{NV} \tag{2}
\]

where \( Z \) is the number of molecules per unit cell, \( M \) is molecular weight of sample, \( N \) is the Avogadro's number and \( V \) is volume of unit cell. Figure 2 shows the variation of X-ray density with \( \text{Cr}^{3+} \) substitution. It can be noted from Fig. 2 that the X-ray density decreases with decrease in \( \text{Cr}^{3+} \) substitution. It is clear from eq. 2 that the X-ray density is directly proportional to the molecular weight \( (M) \). Molecular weight in the present ferrite system decreases with every substitution of \( \text{Cr}^{3+} \) ions result in the decrease in X-ray density.

The average crystallite diameter \( DXRD \) of powder estimated from the most intense (311) peak of XRD and using the Scherrer method [10],

\[
DXRD = \frac{C}{\beta/2 \cos \theta} \tag{3}
\]

where \( \beta/2 \) is the full width of half maximum in (2), \( \theta \) is the corresponding Bragg angle and \( C = 0.9 \). Figure 3 shows that the crystallite diameter decreases with increase in \( \text{Cr}^{3+} \) substitution.

![Fig. 3: Variation of Crystallite size with \( \text{Cr} \) content \( x \).](image)

The bulk density \( (dB) \) measured using the formula

\[
d_B = \frac{m}{\pi r^2 h} \tag{4}
\]

where \( m \) is mass, \( r \) is the radius and \( h \) is the height of the pellet. Figure 4 shows that bulk density decreases with increase in \( \text{Cr}^{3+} \) ions. The decrease in bulk density can be co-related with the decrease in crystallite diameter.

![Fig. 4: Variation of bulk density \( (dB) \) with \( \text{Cr} \) content \( x \).](image)
The percentage porosity is calculated using the following relation:

$$P = \left( \frac{d_{\text{X}} - d_{\text{B}}} {d_{\text{B}}} \right) \times 100$$

where $d_{\text{X}}$ and $d_{\text{B}}$ are the X-ray density and bulk density, respectively.

Figure 5 shows that the porosity increases with increase in Cr+3 substitution. The increase in porosity is related to decrease in crystallite diameter and bulk density.

The hopping length for A-site (LA) and B-sites (LB) are calculated using the values of lattice constant. The variation of hopping lengths with Cr content $x$ is shown in Fig. 6. It is observed from Fig. 6 that the distance between the magnetic ions (hopping length) decreases as Cr substitution increases. This behavior of hopping length with $x$ is analogous with the behavior of lattice constant with Cr+3 substitution. This variation may be attributed to the difference in the ionic radii of the constituent ions, which makes the magnetic ions become smaller to each other and the hopping length decreases. Using the experimental values of lattice constant $a$, oxygen positional parameter $u$ and substituting using the following equations, the allied parameters such as tetrahedral and octahedral bond length ($d_{\text{AX}}$, $d_{\text{BX}}$), tetrahedral edge, shared and unshared octahedral edge ($d_{\text{AXE}}$, $d_{\text{BXE}}$ and $d_{\text{BXEU}}$) were calculated.

$$d_{\text{AX}} = a \sqrt[1/2]{(a^2 - (1/4) u^2 - (1/4) u^4)}$$

$$d_{\text{AXE}} = a \sqrt{2(2u^2)}$$

$$d_{\text{BXE}} = a \sqrt[1/2]{(a^2 - 3u^2 - (11/16))}$$

$$d_{\text{BXEU}} = a \sqrt[1/2]{(a^2 - 3a - (11/16))}$$

Please cite this article as: Anil Shripadkar and Ram Kadam, Cr+3 Ion-dependent hopping length and allied parameters of Ni-Co alloys: Indian Streams Research Journal (May 2012)
Cr³⁺ ion dependent hopping length and allied parameters of Ni-Co alloys

Table 1 indicates that all the allied parameter decreases after initial decrease with Cr³⁺ substitution increases. This could be related to the smaller ionic radius of Cr³⁺ ions as compared with Fe³⁺ ions and the site occupancy of the constituent ions in the present ferrite system.

Table 1

<table>
<thead>
<tr>
<th>Comp. X</th>
<th>dAX (Å)</th>
<th>dAX (Å)</th>
<th>Totta edge (Å)</th>
<th>Octa edge dAX (Å)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>1.895</td>
<td>2.041</td>
<td>3.094</td>
<td>2.811</td>
</tr>
<tr>
<td>0.25</td>
<td>1.893</td>
<td>2.040</td>
<td>3.092</td>
<td>2.808</td>
</tr>
<tr>
<td>0.5</td>
<td>1.892</td>
<td>2.038</td>
<td>3.090</td>
<td>2.807</td>
</tr>
<tr>
<td>0.75</td>
<td>1.891</td>
<td>2.037</td>
<td>3.088</td>
<td>2.805</td>
</tr>
<tr>
<td>1.0</td>
<td>1.890</td>
<td>2.036</td>
<td>3.086</td>
<td>2.804</td>
</tr>
</tbody>
</table>

Conclusions

The Co₀.5Ni₀.5CrₓFe₂-xO₄ ferrite system was successfully synthesized by sol-gel auto combustion method. The prepared samples show the formation of single phase cubic spinel structure without any impurity peaks. Lattice constant and X-ray density found to decrease due to the decrease in ionic radii and molecular with every substitution of Cr³⁺. Crystalline diameter decreases with decrease in Cr³⁺ substitution result in decrease in bulk density which eventually increases the porosity of the samples. Hopping lengths and allied parameters also decreases with Cr³⁺ substitution due to smaller ionic radii of Cr³⁺ ions.

References

Dear Professors, Researchers, and Scientists, research articles, review articles and short communications are invited for the July-August issue of International Journal of Computer Technology and Applications (IJCTA).
## IndexCopernicus Search Journals

Search results

Found: 36 records

<table>
<thead>
<tr>
<th>Title</th>
<th>ISSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advances in Computational Research</td>
<td>0975-9085</td>
</tr>
<tr>
<td>Advances in Computer Science and Engineering</td>
<td>0973-6999</td>
</tr>
<tr>
<td>IETE Journal of Research</td>
<td>0377-2063</td>
</tr>
<tr>
<td>IETE Technical Review</td>
<td>0250-4602</td>
</tr>
<tr>
<td>IJEST - INTERNATIONAL JOURNAL OF ADVANCED ENGINEERING SCIENCES AND TECHNOLOGIES</td>
<td>2230-7616</td>
</tr>
<tr>
<td>Indian Journal of Library and Information Science</td>
<td>0973-9548</td>
</tr>
<tr>
<td>Information Science and Technology</td>
<td>1676-0188</td>
</tr>
<tr>
<td>International Journal of Advanced Computer and Mathematical Sciences</td>
<td>2230-9624</td>
</tr>
<tr>
<td>International Journal of Advanced Research in Computer Science</td>
<td>0976-5697</td>
</tr>
<tr>
<td>International Journal of Advancements in Technology</td>
<td>0976-0660</td>
</tr>
<tr>
<td>International Journal of Applied Engineering Research</td>
<td>0976-4295</td>
</tr>
<tr>
<td>International Journal of BioSciences and Technology</td>
<td>0974-3997</td>
</tr>
<tr>
<td>International Journal of Computational Intelligence Techniques</td>
<td>0975-0474</td>
</tr>
<tr>
<td>International Journal of Computer Science and Information Technologies</td>
<td>0975-9646</td>
</tr>
<tr>
<td>International Journal of Computer Technology and Applications</td>
<td>2230-6093</td>
</tr>
<tr>
<td>International Journal of Engineering Science and Technology</td>
<td>0975-5402</td>
</tr>
</tbody>
</table>
Manuscript Acceptance letter

Dear Latika kharb,

Your Manuscript entitled "Reliable Software Development with Proposed Quality Oriented Software Testing Metrics" has been accepted for publication in the July-August 2011 (Volume 2 Issue 4) of IJCTA. Kindly pay the processing fee of Rs.3200. Kindly send the processing charge and duly signed copyrights form for further proceedings.

- CHIEF EDITOR
Dear Latika kharb,

Your Manuscript entitled "Reliable Software Development with Proposed Quality Oriented Software Testing Metrics" has been accepted for publication in the July-August 2011 (Volume 2 Issue 4) of IJCTA. Kindly pay the processing fee of Rs. 3200. Kindly send the processing charge and duly signed copyrights form for further proceedings.

- CHIEF EDITOR
Reliable Software Development with Proposed Quality Oriented Software Testing Metrics

Ms. Latika Kharb*, Dr. Vijay Singh Rathore**
Research Scholar*, Professor**
Department of Computer Science, NIMS University, Jaipur*
Department of Computer Science, S.K. College, Jaipur**

ABSTRACT
For an effective test measurement, a software tester requires a testing metrics that could measure the quality and productivity of software development process along with increasing its reusability, correctness and maintainability. Until now, the understanding of measuring software quality is not yet sophisticated enough and is still far away from being standardized and in order to assess the software quality, an appropriate set of software metrics needs to be identified that could express these quality attributes. Our research objective in this paper is to construct and define a set of easy-to-measure software metrics for testing to be used as early indicators of external measures of quality. So, we've emphasized on the fact that reliable software development with respect to quality could be well achieved by using our set of testing metrics, and for that we've given the practical results of evaluation.

1. Introduction: Software Quality Measurement through Testing
In order to ensure that the software under development has been implemented correctly, software testing is deployed as an integral part of software development in every phase of the software development cycle specifically, to understand the requirements, to produce better-quality code and for the development of efficient testing measurement techniques that could assist in the creation of high-quality software with-in limited time and resources. In order to achieve this, entire user satisfaction is measured in terms of quality of compliant product and its delivery within scheduled time, cost and budget, i.e.

**Figure 1. The Purposes of Quality**

Software testing provides visibility into the product quality as well as process quality and as the goal of software testing is to discover errors in the software for building confidence in the proper operation of the software; therefore, solving the software-testing problems is not an easier task as one could never be sure that the specifications are correct. The key to effective measurement lies in the ability to clearly identify the goals to be accomplished and the issues to be tackled [2] because an effective software measurement technique for testing is the first step to make software development as well as software engineering a true engineering discipline as it helps in the evaluation of not only the quality of the requirements document for quality indicators, identifies volatility, but also tracks testing as a way for ensuring that all requirements have been satisfied. Measurement plays a critical role in effective software development [3]. Therefore, software-testing measurements must be planned carefully because it requires significant efforts to implement, and returns are realized only over a period of time. Software metrics for testing are important because of the benefits associated with
early detection and correction of problems while testing. Although many metrics have been proposed by researchers, but most of them are either ignored or are left in isolation. Testing metrics have been one of the most sophisticated processes for use in measurement and demonstration of the correctness and quality of a program. Software quality based testing metrics are developer-oriented and developers could use them to estimate quality at a very early stage in the software development process. Quality of a software product is directly linked with the faults present in the software module. Quality is an indicator of high performance, whether that performance is measured in terms of individuals, teams, products, or the entire organization. ISO 8402 defines quality as the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs [4]. Despite the range of definitions, the goals underlying the pursuit of quality are the same: achieving conformity, reducing variation, eliminating waste and rework, eliminating non-value-adding activity, preventing human error, increasing efficiency and effectiveness, improving productivity, and preventing defects. Metrics are important because of the benefits associated with early detection and correction of problems with requirements [5]. Actually, many factors determine the quality of the software like,

- Clear and documented functionality of the software.
- Status of the development of software.
- Test cases cover the complete functionality. Process to determine the severity and priority of the defect.
- Collecting and analyzing the testing metrics.

Using quality management techniques, as the basis for software development in testing is a promising avenue for improvements in a resource constrained environment where challenges are ever increasing. Software testing is one of the tools used to ascertain the quality of software as software quality is achieved by testing to verify its correctness, performance and scalability according to its specification. There are several test metrics identified as part of the overall testing activity in order to track and measure the entire testing process. These test metrics are collected at each phase of the testing life cycle (SDLC) and analyzed and appropriate process improvements are determined and implemented as a result of these test metrics that are constantly collected and evaluated as a parallel activity together with testing both for manual and automated testing irrespective of the type of application. The test metrics can be broadly classified into the following three categories such as: Project Related Metrics, Process Related Metrics, Customer related Metrics.

2. Metrics for Quality Testing

Test metrics are an important indicator of the effectiveness of a software testing process. In current years, there have been many discussions about the role of software metrics in helping software organizations to improve productivity and software quality. Some researchers have defined new metrics for specific measurement purposes and contexts software engineering management. Researchers have put much effort into learning how to use metrics for software process improvement (SPI) [6]. Our future work includes using real industry level data to evaluate these new metrics we recommended to measure performance of individual test phases, giving suggestions to test teams and support teams for necessary changes in the test process, and implementing the whole set of metrics in a production test environment. In this section, four new quality metrics for achieving operational excellence in software testing have been proposed namely, quality of code metric (QCM), quality of product metric (QPM), total test improvement metric (TTIM), and test effectiveness metric (TEM). As a single metric could only measure one aspect of software test quality; therefore, it must not be applied or used in isolation. So, we recommended metrics to measure test phase that may help to achieve the organizational goal of quality.

2.1. Quality of code Metric (QCM)

The following metric is used to measure the quality of software code and is measured by taking the average of the sum of the number of weighted defects found in a product before and after release & thousand source lines of code with the total lines of code of developed product. It is given by the equation:

\[
QCM = \frac{(W_{BR} + W_{AR})}{KSLOC}
\]

Where \(W_{BR}\) is the number of weighted defects found in a product under test (before release), \(W_{AR}\) is the number of weighted defects found in a product under test (after release). The weight for each defect is defined on the basis of defect
severity and removal cost. A severity is assigned to each defect by experienced software testers based on how important or serious the defect is. The more important or serious the defect, it's more expensive to remove it, and the higher the weight which is assigned to the defect. KSLOC denotes the thousand source lines of code and LOCpr is the number of total lines of code of developed product. This metric (QCM) captures the relation between the number of weighted defects and the size of the product release. The lower this number, indicating fewer defects or less serious defects found, the higher is the quality of the code delivered by the development teams.

2.2. Quality of Product Metric (QPM) (measured after official delivery to customer)

The following metric is used to measure the quality of software product and is measured by taking the average of the sum of the number of weighted defects found in a product before and after release with the total lines of code of developed product. The weight for defects is again defined based on defect severity and removal cost. It is given by the equation:

\[ QPM = \frac{W_{BR} + W_{AR}}{LOC_{pr}} \]

This metric (QPM) shows the relation between the number of weighted defects shipped to customers and the size of the product release. A low number here indicates fewer defects, or less serious defects, implying a higher quality of the code delivered by the test teams.

2.3. Total Test Improvement Metric (TTIM)

The following metric is used to measure the total test improvement and is measured by taking the average of the sum of number of weighted defects found in a product under test through white-box testing, black box testing and grey-box testing respectively with the total lines of code of developed product. It is given by the equation:

\[ TTIM = \frac{W_{WBT} + W_{BBT} + W_{GBT}}{LOC_{dp}} \]

Where \( W_{WBT} \) is the number of weighted defects found in a product under test through white-box testing, \( W_{BBT} \) is the number of weighted defects found in a product under test through black-box testing, and \( W_{GBT} \) is the number of weighted defects found in a product under test through grey-box testing. This metric (TTIM) shows the relation between the number of weighted defects detected by the test team during testing and the size of the product release. The higher this number, indicating more defects or more important defects were detected, the higher the improvement to the quality of the product that can be attributed to the test teams.

2.4. Test Effectiveness Metric (TEM) (to drive out defects after official delivery to customer)

The following metric is used to measure the total test effectiveness and is measured by calculating the ratio of the average of the Total Test Improvement Metric with the sum of the number of weighted defects found in a product before and after release. It is given by the equation:

\[ TEM = \frac{TTIM}{(W_{BBT} + W_{GBT})} \times 100\% \]

Where TTIM is the test Improvement Metric. This metric (TE) shows the relation between the number of weighted defects detected during testing and the total number of weighted defects in the product. The higher the number, indicating a higher ratio of defects or important defects were detected before release, the higher is the effectiveness of the test organization to drive out defects.

3. Practical Work: The Project Process

The basic idea behind these quality metrics is that it is possible to set a number of basic requirements that shall be fulfilled so that a software system becomes well designed. By formalizing the requirements and gathering metrics for them it is possible to analyze code, and becomes possible to grade developed systems in terms of the design quality. In next section, the practical usage of software metrics will be discussed by presenting practical strategies. In this research work, there were three major phases in our research work:

- Data collection phase,
- Metrics calculation phase, and
- Result analysis phase.
3.1. Data collection Phase

Data for calculating test process metrics have to be collected from the test process. From our experience, a small project is useful and necessary before adopting any metric into practice. In doing theoretical work, such as defining the metric, we may not truly understand the availability of data. Even when we have tried to make data collection relatively easy, some minor changes may still be required in the test process to get correct and accurate data. In this research work, we used two of our working projects of Hotel Management and Payroll System in languages of Visual Basic and Oracle. Based on the definition of the metrics, we derived a list of variables for which we needed to collect data in the data collection phase. These variables were organized into four categories as follows.

- **Product size data:** $KSLOC, LOC_{DP}$
  
  \[ KSLOC \text{ denotes the thousand source lines of code and } LOC_{DP} \text{ is the number of total lines of code of developed product.} \]

- **Weighted defects data:** $W_{HR}, W_{AR}, W_{WBT}, W_{CBT}$

3.2. Metrics calculation Phase

In the data collection phase, we've collected information for all the variables for whom we need to calculate the metrics. The metrics calculation phase is straightforward: by applying the value of each variable to the metrics, we can calculate the results easily. After all the metrics have been calculated, table similar to the ones given in next section could be created.

We’ve taken values of $W_{HR}, W_{AR}, KSLOC$ and $LOC_{DP}$ and placed the respective values in the equation of $QCM, QPM, TTIM$ and $TEM$ metrics. Placing values in metrics equations, we've the results:

1. $QCM$ for Project 1
   \[
   QCM = (220+4) / 8000 \times 30000 = 9.333
   \]

2. $QCM$ for Project 2
   \[
   QCM = (150+5) / 10000 \times 30000 = 5.166
   \]

3. $QPM$ for Project 2
   \[
   QPM = (4) / (220+4) = 0.0074
   \]

4. $TEM$ for Project 1
   \[
   TEM = (0.0074/ (220+4)) \times 100\% = 0.0032
   \]

3.3. Result analysis Phase

In the result analysis phase, we used data to assess whether the software test process meets the organizational goals. Also, we can compare metrics for two different product releases to predict characteristics of future output or performance, and also to suggest future improvements. We’ve summarized the results of calculated metrics in Table 1 below.

<table>
<thead>
<tr>
<th>List of variables</th>
<th>Project 1</th>
<th>Project 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_{HR}$</td>
<td>220</td>
<td>150</td>
</tr>
<tr>
<td>$W_{AR}$</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>$KSLOC$</td>
<td>8000</td>
<td>10000</td>
</tr>
<tr>
<td>$LOC_{DP}$</td>
<td>30000</td>
<td>30000</td>
</tr>
<tr>
<td>$W_{WBT}$</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>$W_{CBT}$</td>
<td>120</td>
<td>50</td>
</tr>
<tr>
<td>$W_{GRT}$</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>$QCM$</td>
<td>9.333</td>
<td>5.166</td>
</tr>
<tr>
<td>$QPM$</td>
<td>0.0074</td>
<td>0.0051</td>
</tr>
<tr>
<td>$TTIM$</td>
<td>0.0074</td>
<td>0.0051</td>
</tr>
<tr>
<td>$TEM$</td>
<td>0.0033</td>
<td>0.0032</td>
</tr>
</tbody>
</table>

4. Effectiveness of the Metrics

Researchers have put much effort into learning how to use metrics for software process improvement (SPI) and there have been many discussions in current years about the role of
software metrics in helping software organizations to improve productivity and software quality. The research spans a wide variety of topics. Some researchers have defined new metrics for specific measurement purposes and contexts software engineering management and metrics are not purely technical disciplines. Educating project managers, test managers, and development managers as to what we are measuring, as well as what those numbers mean is very important. This should be done for two reasons. The first is to ensure that managers support and understand the value of the metrics. It is vital that they are interested in these metrics as much as we are in providing them. The second reason is to educate them on what they can do to affect each metric positively. This last reason is the most important, yet is also the most difficult to explain. Defects themselves pose an interesting problem when it comes to classification. An effective measurement activity should be able to evaluate the current process and provide suggestion to the manager for future improvement. The metrics we used in our project could be able to provide information that is helpful for justifying the current test process. The metric results clearly show the improvement that the test teams had made in the test process in terms of quality. In section 4.3, we discussed some of the results found in our two projects and our work includes real data for the evaluation of proposed metrics and implementing the whole set of metrics in a project environment.

5. Conclusion
The goal of the testing activity is to find as many errors as possible before the user of the software finds them. We can use testing to determine whether a program component meets its requirements. To accomplish its primary goal (finding errors) or any of its secondary purposes (meeting requirements), software testing must be applied in a systematic fashion. Testing involves operation of a system or application under controlled conditions and evaluating the results. By using our software testing metrics in a consistent manner, software developers will see improvement in the software and on the use of the metrics. However, no single metric works during all of the development phases; therefore, using several metrics for one system helps to have a handy solution that can be used during different aspects of the process of software development. The metrics covered in this paper are the following: quality of code metric (QCM), quality of product metric (QPM), total test improvement metric (TTIM), and test effectiveness metric (TEM). This paper presents a study and implementation of different software metrics. We apply these metrics to our sample projects, and evaluated the results. We find that there are specific metrics for different software quality assessments. When used properly, i.e., when a company uses the best software testing metric during each development phase, the quality of the software will dramatically increase. Therefore, we highly recommend using software-testing metrics for the software quality assessment.

6. References
Abstract
For an effective test measurement, a software tester requires a testing metrics that could measure the quality and productivity of software development process along with increasing its reusability, correctness and maintainability. Until now, the understanding of measuring software quality is not yet sophisticated enough and is still far away from being standardized and in order to assess the software quality, an appropriate set of software metrics needs to be identified that could express these quality attributes. Our research objective in this paper is to construct and define a set of easy-to-measure software metrics for testing to be used as early indicators of external measures of quality. So, we've emphasized on the fact that reliable software development with respect to quality could be well achieved by using our set of testing metrics, and for that we've given the practical results of evaluation.

1. Introduction: Software Quality Measurement through Testing
In order to ensure that the software under development has been implemented correctly, software testing is deployed as an integral part of software development in every phase of the software development cycle specifically, to understand the requirements, to produce better-quality code and for the development of efficient testing measurement techniques that could assist in the creation of high-quality software with-in limited time and resources. In order to achieve this, entire user satisfaction is measured in terms of quality of compliant product and its delivery within scheduled time, cost and budget, i.e.

User Satisfaction = Compliant Product + Good Quality + Delivery within Schedule
With fundamental research that addresses the challenging problems like development of methods, tools and empirical studies, we could not expect significant improvement in the way we test software [1].

Figure 1. The Purposes of Quality
Software testing provides visibility into the product quality as well as process quality and as the goal of software testing is to discover errors in the software for building confidence in the proper operation of the software; therefore, solving the software-testing problems is not an easier task as one could never be sure that the specifications are correct. The key to effective measurement lies in the ability to clearly identify the goals to be accomplished and the issues to be tackled [2] because an effective software measurement technique for testing is the first step to make software development as well as software engineering a true engineering discipline as it helps in the evaluation of not only the quality of the requirements document for quality indicators, identifies volatility, but also tracks testing as a way for ensuring that all requirements have been satisfied. Measurement plays a critical role in effective software development [3]. Therefore, software-testing measurements must be planned carefully because it requires significant efforts to implement, and returns are realized only over a period of time. Software metrics for testing are...
important because of the benefits associated with early detection and correction of problems while testing. Although many metrics have been proposed by researchers, but most of them are either ignored or are left in isolation. Testing metrics have been one of the most sophisticated processes for use in measurement and demonstration of the correctness and quality of a program. Software quality based testing metrics are developer-oriented and developers could use them to estimate quality at a very early stage in the software development process. Quality of a software product is directly linked with the faults present in the software module. Quality is an indicator of high performance, whether that performance is measured in terms of individuals, teams, products, or the entire organization. ISO 8402 defines quality as the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs [4]. Despite the range of definitions, the goals underlying the pursuit of quality are the same: achieving conformity, reducing variation, eliminating waste and rework, eliminating non-value-adding activity, preventing human error, increasing efficiency and effectiveness, improving productivity, and preventing defects. Metrics are important because of the benefits associated with early detection and correction of problems with requirements [5].

Actually, many factors determine the quality of the software like,

- Clear and documented functionality of the software.
- Status of the development of software.
- Test cases cover the complete functionality. Process to determine the severity and priority of the defect.
- Collecting and analyzing the testing metrics.

Using quality management techniques, as the basis for software development in testing is a promising avenue for improvements in a resource constrained environment where challenges are ever increasing. Software testing is one of the tools used to ascertain the quality of software as software quality is achieved by testing to verify its correctness, performance and scalability according to its specification. There are several test metrics identified as part of the overall testing activity in order to track and measure the entire testing process. These test metrics are collected at each phase of the testing life cycle /SDLC and analyzed and appropriate process improvements are determined and implemented as a result of these test metrics that are constantly collected and evaluated as a parallel activity together with testing both for manual and automated testing irrespective of the type of application. The test metrics can be broadly classified into the following three categories such as: Project Related Metrics, Process Related Metrics, Customer related Metrics.

2. Metrics for Quality Testing

Test metrics are an important indicator of the effectiveness of a software testing process. In current years, there have been many discussions about the role of software metrics in helping software organizations to improve productivity and software quality. Some researchers have defined new metrics for specific measurement purposes and contexts software engineering management. Researchers have put much effort into learning how to use metrics for software process improvement (SPI) [6]. Our future work includes using real industry level data to evaluate these new metrics we recommended to measure performance of individual test phases, giving suggestions to test teams and support teams for necessary changes in the test process, and implementing the whole set of metrics in a production test environment. In this section, four new quality metrics for achieving operational excellence in software testing have been proposed namely, quality of code metric (QCM), quality of product metric (QPM), total test improvement metric (TTIM), and test effectiveness metric (TEM). As a single metric could only measure one aspect of software test quality; therefore, it must not be applied or used in isolation. So, we recommended metrics to measure test phase that may help to achieve the organizational goal of quality.

2.1. Quality of code Metric (QCM)

The following metric is used to measure the quality of software code and is measured by taking the average of the sum of the number of weighted defects found in a product before and after release & thousand source lines of code with the total lines of code of developed product. It is given by the equation:

\[
QCM = \frac{(W_{BR} + W_{AR})}{KSLOC}
\]

Where \(W_{BR}\) is the number of weighted defects found in a product under test (before release), \(W_{AR}\) is the number of weighted defects found in a product under test (after release). The weight
for each defect is defined on the basis of defect severity and removal cost. A severity is assigned to each defect by experienced software testers based on how important or serious the defect is. The more important or serious the defect, it's more expensive to remove it, and the higher the weight which is assigned to the defect. KSLOC denotes the thousand source lines of code and LOC_{DP} is the number of total lines of code of developed product. This metric (QCM) captures the relation between the number of weighted defects and the size of the product release. The lower this number, indicating fewer defects or less serious defects found, the higher is the quality of the code delivered by the development teams.

2.2. Quality of Product Metric (QPM) (measured after official delivery to customer)

The following metric is used to measure the quality of software product and is measured by taking the average of the sum of the number of weighted defects found in a product before and after release with the total lines of code of developed product. The weight for defects is again defined based on defect severity and removal cost. It is given by the equation:

\[ \text{QPM} = \frac{W_{BR} + W_{AR}}{\text{LOC}_{DP}} \]

This metric (QPM) shows the relation between the number of weighted defects shipped to customers and the size of the product release. A low number here indicates fewer defects, or less serious defects, implying a higher quality of the code delivered by the test teams.

2.3. Total Test Improvement Metric (TTIM)

The following metric is used to measure the total test improvement and is measured by taking the average of the sum of number of weighted defects found in a product under test through white-box testing, black box testing and grey-box testing respectively with the total lines of code of developed product. It is given by the equation:

\[ \text{TTIM} = \frac{W_{WBT} + W_{BBT} + W_{GBT}}{\text{LOC}_{DP}} \]

Where \( W_{WBT} \) is the number of weighted defects found in a product under test through white-box testing, \( W_{BBT} \) is the number of weighted defects found in a product under test through black box testing, and \( W_{GBT} \) is the number of weighted defects found in a product under test through grey-box testing. This metric (TTIM) shows the relation between the number of weighted defects detected by the test team during testing and the size of the product release. The higher this number, indicating more defects or more important defects were detected, the higher the improvement to the quality of the product that can be attributed to the test teams.

2.4. Test Effectiveness Metric (TEM) (to drive out defects after official delivery to customer)

The following metric is used to measure the total test effectiveness and is measured by calculating the ratio of the average of the Total Test Improvement Metric with the sum of the number of weighted defects found in a product before and after release. It is given by the equation:

\[ \text{TEM} = \frac{\text{TTIM}}{W_{BR} + W_{AR}} \times 100\% \]

This metric (TE) shows the relation between the number of weighted defects detected during testing and the total number of weighted defects in the product. The higher the number, indicating a higher ratio of defects or important defects were detected before release, the higher is the effectiveness of the test organization to drive out defects.

3. Practical Work: The Project Process

The basic idea behind these quality metrics is that it is possible to set a number of basic requirements that shall be fulfilled so that a software system becomes well designed. By formalizing the requirements and gathering metrics for them it is possible to analyze code, and becomes possible to grade developed systems in terms of the design quality. In next section, the practical usage of software metrics will be discussed by presenting practical strategies. In this research work, there were three major phases in our research work:

- Data collection phase,
- Metrics calculation phase, and
- Result analysis phase.
3.1. Data collection Phase

Data for calculating test process metrics have to be collected from the test process. From our experience, a small project is useful and necessary before adopting any metric into practice. In doing theoretical work, such as defining the metric, we may not truly understand the availability of data. Even when we have tried to make data collection relatively easy, some minor changes may still be required in the test process to get correct and accurate data. In this research work, we used two of our working projects of **Hotel Management** and **Payroll System** in languages of **Visual Basic** and **Oracle**. Based on the definition of the metrics, we derived a list of variables for which we needed to collect data in the data collection phase. These variables were organized into four categories as follows.

- **Product size data**: KSLOC, LOC$_{DP}$
  
  Where KSLOC denotes the thousand source lines of code and LOC$_{DP}$ is the number of total lines of code of developed product.

- **Weighted defects data**: WBR, WAR, WWBT, WGBT, WHT

3.2. Metrics calculation Phase

In the data collection phase, we’ve collected information for all the variables for whom we need to calculate the metrics. The metrics calculation phase is straightforward: by applying the value of each variable to the metrics, we can calculate the results easily. After all the metrics have been calculated, table similar to the ones given in next section could be created.

We’ve taken values of WBR, WAR, KSLOC and LOC$_{DP}$ and placed the respective values in the equation of QCM, QPM, TTIM and TEM metrics. Placing values in metrics equations, we’ve the results:

1. **QCM for Project 1**
   \[ = \frac{(220+4)}{8000}/30000 = 9.333 \]

2. **QCM for Project 2**
   \[ = \frac{(150+5)}{10000}/30000 = 0.0051 \]

3. **QPM for Project 2**
   \[ QPM_{Project 2} = \frac{(150+5)}{30000} = 0.0051 \]

4. **TEM for Project 1**
   \[ TEM_{Project 1} = \frac{(0.0074}{(220+4)) \times 100 \% = 0.0033 \]

3.3. Result analysis Phase

In the result analysis phase, we used data to assess whether the software test process meets the organizational goals. Also, we can compare metrics for two different product releases to predict characteristics of future output or performance, and also to suggest future improvements. We’ve summarized the results of calculated metrics in Table 1 below.

<table>
<thead>
<tr>
<th>List of variables</th>
<th>Project 1</th>
<th>Project 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WBR</td>
<td>220</td>
<td>150</td>
</tr>
<tr>
<td>WAR</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>KSLOC</td>
<td>8000</td>
<td>10000</td>
</tr>
<tr>
<td>LOC$_{DP}$</td>
<td>30000</td>
<td>30000</td>
</tr>
<tr>
<td>W$_{WBT}$</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>W$_{GBT}$</td>
<td>120</td>
<td>50</td>
</tr>
<tr>
<td>W$_{WBT}$</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>QCM</td>
<td>9.333</td>
<td>5.166</td>
</tr>
<tr>
<td>QPM</td>
<td>0.0074</td>
<td>0.0051</td>
</tr>
<tr>
<td>TTIM</td>
<td>0.0074</td>
<td>0.0051</td>
</tr>
<tr>
<td>TEM</td>
<td>0.0033</td>
<td>0.0032</td>
</tr>
</tbody>
</table>

4. Effectiveness of the Metrics

Researchers have put much effort into learning how to use metrics for software process improvement (SPI) and there have been many discussions in current years about the role of
software metrics in helping software organizations to improve productivity and software quality. The research spans a wide variety of topics. Some researchers have defined new metrics for specific measurement purposes and contexts; software engineering management and metrics are not purely technical disciplines. Educating project managers, test managers, and development managers as to what we are measuring, as well as what those numbers mean is very important. This should be done for two reasons. The first is to ensure that managers support and understand the value of the metrics. It is vital that they are interested in these metrics as much as we are in providing them. The second reason is to educate them on what they can do to affect each metric positively. This last reason is the most important, yet is also the most difficult to explain. Defects themselves pose an interesting problem when it comes to classification. An effective measurement activity should be able to evaluate the current process and provide suggestion to the manager for future improvement. The metrics we used in our project could be able to provide information that is helpful for justifying the current test process. The metric results clearly show the improvement that the test teams had made in the test process in terms of quality. In section 4.3, we discussed some of the results found in our two projects and our work includes real data for the evaluation of proposed metrics and implementing the whole set of metrics in a project environment.

5. Conclusion
The goal of the testing activity is to find as many errors as possible before the user of the software finds them. We can use testing to determine whether a program component meets its requirements. To accomplish its primary goal (finding errors) or any of its secondary purposes (meeting requirements), software testing must be applied in a systematic fashion. Testing involves operation of a system or application under controlled conditions and evaluating the results. By using our software testing metrics in a consistent manner, software developers will see improvement in the software and on the use of the metrics. However, no single metric works during all of the development phases; therefore, using several metrics for one system helps to have a handy solution that can be used during different aspects of the process of software development. The metrics covered in this paper are the following: quality of code metric (QCM), quality of product metric (QPM), total test improvement metric (TTIM), and test effectiveness metric (TEM). This paper presents a study and implementation of different software metrics. We apply these metrics to our sample projects, and evaluated the results. We find that there are specific metrics for different software quality assessments. When used properly, i.e., when a company uses the best software testing metric during each development phase, the quality of the software will dramatically increase. Therefore, we highly recommend using software-testing metrics for the software quality assessment.

6. References
SOFTWARE COMPONENT COMPLEXITY MEASUREMENT THROUGH PROPOSED INTEGRATION METRICS

Ms. Latika¹, Dr. Vijay Singh Rathore²

¹Research Scholar,
NIMS University, Jaipur, India
latika.kharb@gmail.com
²Director
S.K. College, Jaipur, India

Abstract: Component based software paradigm has become one of the preferred streams for developing large and complex systems by integrating prefabricated software components which not only facilitates the process of software development but also solves many adaptation and maintenance problems, so it's changing the ways for software professionals to develop software applications. Till today, numerous attempts have been made by several organizations, software development teams, developers as well as researchers to improve software component systems through improved measurement tools and techniques i.e. through an effective metrics. Our paper is a simple attempt to work for the demand of an appropriate and relevant metrics and in this paper, we've proposed some integration metrics for the measurement of complexity of a software component, which could be used as one of the approaches for further guidance in component complexity measurement and problem reduction.

Keywords: Component complexity measurement, COSS, integration metrics.

SOFTWARE COMPONENT BASED SYSTEMS: AN INTRODUCTION

Software Component based systems has become one of the preferred streams for developing large and complex systems by integrating prefabricated software components which not only facilitates the process of software development but also solves many adaptation and maintenance problems, so it's changing the ways for software professionals to develop software applications. However, a mismatch exists in the usability of metrics by academicians/researchers and industry men/developers. Even today, much of the metrics activity in industrial sector is based on metrics invented long ago in 70's. This mismatch exists between them because of the following reasons:

A) Researchers/academicians are mainly concerned with detailed and code-oriented metrics while the industrial sectors demand those metrics that could help them in their software process improvement. This difference between needs is the main cause behind the mismatch between usability criteria of various metrics.

B) Industrial sectors have to abide by some rules and regulations/standards of their company while academicians/researchers are not bound by any such rigid standards and can select/change metrics whenever needed according to their needs and requirements.

C) Researchers go for relatively small field works with small data (consisting of small programs) that could get them quick outputs. But the industry men have to go for large projects (to develop huge software). In academics, metrics may or may not be evaluated for correctness, quality and timeliness in hard values. They have to just provide data/values in form of theoretical validations. But the industry men are the one who deal with practical implementation of data and so they have to check each and every metric very minutely as even 1% error rate could be critical if it belongs to real life software development viz. aeronautical systems. A software component should be adequately packaged/specification through its interfaces in order to facilitate proper usage. CBSD offers an effective approach to develop the components required to support various functions and processes for a particular area.

In short, in the last few years-most of the research has been inclined towards methods and approaches that work towards development of software systems and in comparison, a very little work has been made for the development of measures/metrics that can be used to evaluate the complexity of components being developed, using component integration. The main issues in component metrics for capturing integration complexity and complex interfaces tend to complicate the testing process of the system [1].

INTRODUCTION TO SOFTWARE COMPONENT TECHNOLOGY

As the software development managers are increasingly changing their focus on component technology, it seems that in recent future Software Component based systems will become the most preferred industry approach towards development of improved software systems. In Software Component based systems, software components are assembled so that they interact with each other and satisfy predefined functions, so each component has to provide a pre-specified service with other components and thus interface is an important concern to be discussed before proposing metrics for measurement of integration complexity. Software Component based systems is a branch of the software engineering discipline which lays emphasis on decomposition of the engineered systems into functional...
or logical components with well-defined interfaces used for communication across the components. A COSS is a software system that is modeled, designed and developed by integrating components through independent deployment. Software Component based systems are built by combining the well-defined, independently produced pieces with self-made components [2]. A software component is a unit of composition that can be deployed independently by third parties and contains only the contractually specified interfaces and the explicit context dependencies. A software component is made up of three essential parts: the interface, the implementation, and the deployment [3]. A component interface consists of a variable part and a fixed part. The variable part corresponds to possible variants in the component’s implementation and maps to the collection of possible implementations; the fixed part expresses invariant characteristics of the component. Combining components to form a software system implies combining their fixed and variable parts. Combining the variable parts may easily lead to a combinatorial explosion of possible configurations. As a component is typically developed in a system environment, which is different from the environment of the final system, so it is difficult to predict the component behavior in the new system.

COMPONENT COMPLEXITY

Complexity is a measure of the resources expanded by a system while integrating with a piece of software to perform a given task [4]. Software complexity is that aspect of software which is used to predict external properties of the program like reliability, understandability & maintainability. Complexity measures the number of components and their interconnections and is typically based on internal product attributes, such as cohesion and coupling [5]. Complexity mainly results from the components’ organization and interactions between these components. We can define overall system complexity as a function of the interactions between system components and individual complexities of components. Software Component based systems can be obtained as a result of the composition of some components with defined interfaces and then the component’s functionality is implemented in its methods and is provided for other components through its well-defined interfaces [6].

Component Composition and Component Decomposition

A productive component market would deliver a wide range of components, designed for different integration mechanisms, programmed in different programming languages, and located at a diverse number of places. True collaborative software development demands that such diverse components can easily be composed, retrieved, and configured. However, in practice achieving such compositionality turns out to be rather complicated [7]. Components must be integrated through some well-defined infrastructure. This infrastructure provides the binding that forms a system from the disparate components [8].

Coss Metrics

The component paradigm focuses on developing large software systems by integrating prefabricated software components [9]. Component metrics are used to evaluate properties of something being measured, such as quality, complexity or effort, in an objective manner. A component’s functionality is implemented in its methods, which is then provided to other components through its well-defined interfaces.

Component Integration

One of the main objectives of developing Software Component based systems is to enable efficient building of systems through the integration of components [10]. Integration can occur at different moments in time, each requiring a different integration mechanism. The component paradigm focuses on developing large software systems by integrating prefabricated software components [9] and also facilitates the process of software development to solve problems of adaptation and maintenance.

PROPOSED COMPONENT INTEGRATION METRICS

Software metrics can provide useful information to project managers and software developers by providing means of measuring the complexity of a software product. Software complexity means measurement of the resources expended in developing, testing, debugging, maintenance, user training, operation and correction of software products. Component complexity possesses two intrinsic complexities coming from mechanisms inside the component, and extrinsic complexities resulting from interactions with other components i.e. incoming and outgoing interactions. In other words, according to our metric approach, a component oriented complexity metric is valid if it accurately measures the aspects of component-oriented system that influence its internal and external interactions. Incoming-interactions are any received interface that is required in a component, and/or any received event that comes to a component. Outgoing-interactions are any provided interface used and/or possible source of events consumed.

% Of Component Interactions (C I %)

CI% is defined, as a ratio of the available number of incoming interaction used to the available number of outgoing interaction i.e. the component interaction metric CI% will provide a ratio of interactions in a system where, I0 is denoted for the number of outgoing interaction used, and I1 is denoted for the number of incoming interaction available. The equation is given by C I %.

\[ CI\% = \frac{I_0}{I_0 + I_1} \times 100\% \]  

Interaction %Age Metrics For Component Integration (I%MCI)

We’ve proposed the Interaction %age metrics for component integration (I%MCI) in order to measure the interaction density among components in a software system. To measure I%MCI, we define I%MCI which is the ratio between the actual number of interactions to the CI % which is the % of component Interactions metric (as in equation (i)).

\[ I\%MCI = \frac{I_1 + I_0}{CI\%} \]  

Actual Interactions (Ai)

We’ve proposed the metric actual interactions (Ai), which is the ratio between the actual numbers of interactions \((I_1 + I_0)\) to the maximum number of performed interactions \((I_{max})\).
A) Complexity of incoming interactions,
B) Complexity of outgoing interactions, and
C) Components of the whole system.

In our view of complexity, we concentrated on the complexity that is mainly dependent on structure. The metrics proposed in this paper could explain additional variance in measurement effort beyond that explained by other integration metrics. Our paper would provide insight into how application complexity evolves and how it can be managed through the use of metrics.

FUTURE WORKS AND CONCLUSION
An effort for contributing towards improved quality through reduced complexity by providing new metrics for components integration has been made in this paper. In this paper, we've proposed some metrics to measure the complexity of software systems integrated through component-based software paradigm and also discussed how a good metric for the component complexity can be of great use for software developers and managers. The component metrics proposed above can further guide component complexity management in component-based systems, by reducing problems encountered during software development.

REFERENCES
[10] Ivica Crnkovic, Magnus Larsson, and Otto Preis, Concerning Predictability in Dependable Component-Based Systems: Classification of Quality Attributes.