Journal of Computer Sciences 2 (1): 07-12, 2006
ISSN 1549-3636
© 2006 Science Publications

# Qualitative Assessments of the Software Architectures of Configuration Management Systems

[1]Abhay Kothari, [2]A.K.Ramani and [3]P.K. Chande
[1]ICFAI Business School, Indore, India
[2] School of Computer Science, DAVV, Indore, India
[3] Maulana Azad National Institute of Technology, Bhopal, India

**Abstract:** The Configuration Management (CM) is a very important area of concentration in software development and maintenance processes. Quality parameters for Configuration Management Tools' architectural designs require rigorous identification and measurement. Two methods for quantitative assessment of quality parameters of the software architectures are proposed. These methods are based upon modularization properties, like, cohesion type, coupling type, module complexity, module size and others. Looking at the problems with the studied tools, like, low flexibility, interoperability, etc., and recent requirements for the CM tools, a new architectural model has been proposed[1]. All the three models are assessed on quality parameters applying both the methods with an objective to validate the superiority of the proposed model.

**Key words:** Quality parameters, configuration management tools, architectural designs

## INTRODUCTION

Software Configuration Management (SCM) is the supportive activities, which go along the whole software development and maintenance cycles. SCM takes care of making the changes in a managed way. Some of the fundamental activities of software configuration management are configuration identification, version control, change control, status accounting and reporting, and configuration audits. Firstly, configuration identification and version control deals with storing software process artifacts with proper version numbers in a data bank called repository. Secondly, change control deals with making changes in the established artifacts in a systematic way. Thirdly, status accounting and reporting works regard storing information, like, when the change was made, who made the change, etc. Lastly, CM audits verify and validate that the changes have been made as desired.

Software architecture is an area, which referrers to the architecture's components and their interconnectivity The software architectural design is the first artifact of the design phase.

Although the Quality attributes of software are specified by different sources, like, McCall and FURPS[2]. They address combination of product revision, transition, and operational aspects of the software. Out of the available quality attributes, it requires filtering to decide the quality attributes (parameters), which are relevant to the architectural design of the SCM tools. Under the present research, the quality parameters relevant to software architectural designs of SCM tools are identified, they are flexibility, interoperability, availability (ease of repair), testability, traceability, portability, and simplicity.

Most of the existing CM systems have three layered architecture[3]. The layers from top to bottom are policy support, basic CM services, and repository. This is referred as the architectural design of the 2nd generation tools. This suffers from lack of flexibility and interoperability due to large module sizes, as modules here, implement numerous functionalities and mixed features[4]. A better architecture from Software Engineering Institute, CMU, USA is the 'CM Services Model,' it offers number of atomic CM services (reducing the module sizes), out of which users can choose the relevant ones. It is a client/server system. It has a problem of over modularization (very small module sizes), which leads to large integration testing efforts. Also, it does not have any specific provision to interoperate with other CM tools to take advantage of their stronger features.

## METHODS FOR QUALITATIVE ASSESSMENTS OF ARCHITECTURES (SCM TOOLS)

Here, method # 1 and method # 2 are proposed, for evaluating the quality attributes of architecture.

**Method No. 1:** Here, we try to define some quality metrics. A metrics should be measurable and be able to quantitatively reveal the differentiation in the values of attributes of the entities for which metrics are defined.
Note: Pre-requisites for these methods are that the requirements should be well defined and software architectures of the various tools must be understood well by the person applying these methods.

**Corresponding Author:** Abhay Kothari , ICFAI Business School, Indore

The methods proposed are collective form of metrics. with basis derived from basic modularization properties, like, cohesion, coupling, etc. The steps for this method are as follows:

1. For constructing Table 1 we use following abbreviations:
   **A** is Type of Cohesion (Degree of functional independence); **M1** is rating given between 0 to 10; M2 and M3 are ratings from 0 to 1 ; **B** is Types of Coupling ;**C** is Application type: Scientific is CS, Business Analysis is CB and Transaction processing is CT ; **FO** is fan-out of the module; **D** is module complexity Factor#1 (MF1); **E** is module size; **F** is module complexity Factor #2 (MF2)

   **Application Type:** They are decided based on how simple the module is in terms of type of functionality it is representing. The types are discussed as follows:

   **Scientific:** This category includes embedded systems, involving complex algorithms, pertaining to systems programming, numerical analysis ( integration or differentiation algorithms); embedded software development, etc.

   **Business analysis:** This category includes large inventory management software, operation research methods' programming implementing decision making, string manipulation, etc.

   **Transaction process:** This category used simple and small programs of transaction processing or batch processing, which does not involve much of decision making, but they are composed of simple data representation and access. Programs here do not have much decision making criterion.

   **M1:**Assessment of cohesion and example: Let us consider an example of low cohesion, a module that performs error processing for an engineering analysis package. The module is invoked when some input validity check is violated. Then, it performs following functions: Generate supplementary data form original, Generate error reports for user, perform calculations as requested by user, update data in database and displays main menu for further processing request. All these preceding tasks are loosely connected, because they are different in nature. Here, we can say that each is a different and functionality independent and will perform best in individual mode. In-case we combine these into one module, then chances of error propagation are high, if one of them is modified. This makes the collective cohesion to be of low magnitude. For such cases cohesion type could be assumed to be procedural or a bit less than that.
2. We derive two formulas, here, multiplication is used instead of average, since in most of the cases individual cost drives are multiplied to achieve the final values:
   Modularity (MOD) = M1*M2*M3; MOD is a metrics trying to represent how well consolidated and functionally independent the modules is. Best MOD situation is: high cohesion, low coupling and low fan out of modules. Complexity (CMPX)= MF1*MF2.
3. Here, an attempt is made to understand following quality parameters of CM tools[2]: Simplicity (SMP) it depends on the complexity of the module, which could be calculated by formula for CMPX = MF1*MF2, so we will use the same rating of CMPX to rate simplicity in a reciprocated way (10-CMPX). Traceability for correctness and communicativeness, it will depend on the modularity, MOD = M1*M2*M3, so we will use the same rating of MOD to rate traceability.
4. The other quality parameters, which are considered for the purpose of the assessing the quality of any CM tools are also based on "modularity" quality metrics as mentioned in the relationship of McCall's software quality factors and quality metrics[2]. They are reliability, maintainability, flexibility, testability, portability, reusability, and interoperability. They are all dependent on modularity quality metrics. We will primarily use MOD as their full or partial rating basis. At the moment we will be including flexibility, testability, portability and interoperability only for assessing the CM tools, as others are not the implicit or explicit requirements of CM tools.
5. As per the relationship between McCall's software quality factors and quality metrics[2], we find that flexibility and testability are dependent on the complexity also, this is in addition to their primary dependency on modularity, so for their ratings CMPX and MOD both are incorporated. Now we introduce another metrics COM, which will indicate logical simplicity and strength of modularization of a module:
   Rating (COM) = sqrt ((10-CMPX)*MOD), if module coupling and/or fan-out information is not available at architectural design level, then we will use M1 instead of MOD. So, Rating (COM) = sqrt (M1*(10-CMPX))
6. For interoperability we will use MOD and data commonality and communication commonality, but since at architectural design can not do much to incorporate the later two parameters, we shall be confined to MOD only; if module coupling/fan-out information is not available at architectural design level we will use M1 instead of the whole formula.

Table 1 Modular Properties

| A | M1 | B | M2 | FO | M3 | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|
| Functional | 10 | No direct coupling | 1.00 | 1-2 | 1 | CS | 10 | Large | 1.0 |
| Sequential | 8.5 | Data | 0.85 | 3-4 | .75 | CB | 7 | Above average | 0.8 |
| Communicational | 7 | Stamp | 0.7 | 5-6 | .5 | CT | 4 | Average | 0.6 |
| Procedural | 5.5 | Control | 0.55 | Above 6 | .25 | | | Small | 0.4 |
| Temporal | 4 | External | 0.4 | | | | | Very small | 0.2 |
| Logical | 2.5 | Common | 0.25 | | | | | | |
| Coincidental | 1 | Content | 0.1 | | | | | | |

On the same lines portability is to be assessed, since other parameters apart from modularity are not incorporable at architectural design stage.

7. For coupling or any other parameter if multiple answers are received we take the average of the ratings.
8. We can take another parameter in consideration termed as "availability" is the probability that system will be available in operational state. Here, we mean to take availability directly in terms of ease of repairing or in general maintainability. It is proportional to reliability so, again as per the relationship between McCall's quality factors and metrics[2] the metrics for maintainability are taken for the same purpose for availability/ease of repairing, again as it is seen that maintainability depends on modularity and complexity (simplicity). Good cohesion helps to reduce mean time to repair (MTTR) by helping fault diagnosis. This makes the system available for operations. So, availability will be treated the same way flexibility is, Rating = sqrt(MOD*(10-CMPX) or Rating = sqrt(M1*(10-CMPX)); if required information to calculate MOD is not available at architecture design stage.
9. So, the final list of quality parameters for CM tool architectural design comparison is: Flexibility, testability, portability, availability, simplicity, traceability for correctness and communication, and interoperability
10. Based on Table 1, entries for every architecture, for each of their components, an assessment of the five parameters degree of cohesion, coupling, fan-out, complexity and module size is done, and corresponding ratings, like, M1, M2, M3, MF1 and MF2 are determined with the help of Table 1.

    Number of components = n; For each component compute following:

    CMPX_n = MF1*MF2 ; SMP_n =1-CMPX_n; MOD_n = M1*M2*M3 ; COM_n = Sqrt(MOD_n*(10-CMPX_n) ;

    Here, M1,M2,M3, MF1 and MF2 referrers to a particular component. An after calculating the CMPX, SMP, MOD, and COM for all individual components of any architecture, their averages can be taken.
11. The quality parameters can be computed as per the following formulas:

    Flexibility=Testability=Availability=Average(COM)
    Traceability=Interoperability= Portability = Average(MOD)
    Simplicity = Average(SMP)

**Method No. 2:** This method remains same as method 1. Only the difference is in processing the results for any special purpose component present in any of the architectures for that three options are suggested below. The final results of method 1 can be used directly for any architecture if no special purpose component is available for any of the quality attribute support.

**Option 1**

1. The COM_n and/or MOD_n rating of the relevant special purposed component (Cn) is to be incremented by 1 and then do the computations.
2. In case there are more than one special components for a single quality parameter, COM or MOD for both the components will be increased by 1. Along with this if one special purpose component supports more than one quality parameter then its rating should be increase for both the parameters in terms of MOD and/or COM as the case may be.

**Option 2**

1. If there is a component which is dedicated for any one quality parameter. It will receive weight-age of 80%, rest of the COMs or MODs related to rest of the components will collectively get a weight-age of 20% (Parato principle: 80% of output are caused by 20% of components, and rest 80% of components cause 20% of the output ).
2. If there are two components C3 and C5, contributing towards one quality parameter, the weight-age of 80% shall be divided between the two preferably equal. Unless there is a strong reason given.
3. If one component supports more than one quality parameter then in both the cases and will get a weight-age of 80% in both the cases.

**Option 3:** Now, we can specify any special adjustment to be done on the rating for a particular component DEDICATED FOR SOME specific quality parameter, after it has been populated as per the formulas given. This special adjustment could be increasing the existing rating by 25% or any other amount, due to the presence of a special component for a particular quality

Table 2: Quality Parameters for Architectures

| Quality Parameter | 2nd Generation CM tool (either of the methods) | CM services Model (either of the methods) | Proposed architecture (method#1) | Proposed architecture (Method#2, option 1) | Proposed architecture (Method#2, option 2) | Proposed Architecture (Method#2, option 3) |
|---|---|---|---|---|---|---|
| Flexibility | 5.77 | 6.32 | 7.3 | 7.44 | 8.66 | 10 |
| Testability | 5.77 | 6.32 | 7.3 | 7.3 | 7.3 | 7.3 |
| Portability | 6.24 | 4.65 | 7.8 | 7.8 | 7.8 | 7.8 |
| Availability/ Ease of repair | 5.77 | 6.32 | 7.3 | 7.3 | 7.3 | 7.3 |
| Simplicity | 5.34 | 8.6 | 6.88 | 6.88 | 6.88 | 6.88 |
| Traceability | 6.24 | 4.65 | 7.8 | 7.8 | 7.8 | 7.8 |
| Interoperability | 6.24 | 4.65 | 7.8 | 7.98 | 8.04 | 10 |

parameter. Also, you need to give this component weight-age of 80% and spread rest 20% to rest of the components just the way option 2 has been processed.

## RESULTS

On applying these methods on three different architectures of configuration management tools following results were obtained (Table 2). Brief descriptions of the architectures are given here:

**Architecture 1: 2nd generation CM tools' architecture:** It is a layered architecture, the bottom layer is repository, the middle layer offers all configuration management services and top layer offers management of configuration management documents, like, plan, policy and procedure. Each layer here is treated as a separate component.

**Architecture 2: CM services model:** It is a client-server model. All basic CM services, like, configuration identification, change control, CM audits, and status accounting and reporting are implemented in around 50-55 small functions (sub-modules), available on server. Different users (role players) copy the required services from the server in their workspace available (private work area having the relevant files) on the client machine.

**Architecture 3: Proposed architecture:** Although the proposed architecture consists of 15 components, but only 11 have been taken into consideration. This is due to the fact that others are optional services[5]. The components are listed below:

**C1**: Users/subcontractor (general) interaction and relationship management

**Discussion:** This module will be of substantial size and will have facility to interact with the users, like, project manager, configuration manager, software engineer and the customers. We can see that this module will be doing some predictive analysis regarding, who can request what, it will keep a good communication and co-ordination, so the following ratings could be given.
Type of cohesion = functional, since this module is independent of other modules
Types of coupling= no direct coupling and data coupling
MF1 = 7, since the application type is business analysis( prediction and co-ordination.)
MF2 = .8 since module size is above average. So,
Fan-out (Appx.)= 2, so M3=1
Modularity will be equal to MOD=M1*M2*M3 = 10*(1 +.85)/2= 9.25
and, Complexity CMPX = 7*.7=4.9 = 5(appx.),

**C2**: Change request and component traceability, change dependency analysis, distributed development feature (change set)[6,7].

**C3**: Artifact Access brokering/Access permission/Repository access control:

**C4**: Dynamic Process Specification: CM process Specification[8-10]

**C5**: Process Monitoring: Status reporting and accounting

**C6**: Process Control: CM reviews and audits

**C7**: Component based development support module[11,12]

**C8**: Deployment function Module[13,14]

**C9**: Rapid Application Development process Support Module[15, 16]

**C10**: Interface based on event driven implicit invocation to make CM system interoperable, this would consist of a rule component to select appropriate CM tool for its required feature.

**C11:** Configurable Data router to route the data to appropriate tool.

### Advantages

* Methods are based on most fundamental modular properties of cohesion, coupling and fan-out. The set of metrics used for complexity calculation is size and the type of application, which have very little component of subjectivity in them. So, there is no need for taking multiple samples of ratings from different people, this saves time and effort, so methods are very efficient. Also, the use of

fundamental properties to form the basis of the methods, makes the methods direct and applied.

* The other metrics, which are derived, like, COM, MOD, SMP, and CMPX are based on simple mathematical operations. They do not involve major statistical tools. Only sum and averages are applied to them to take out the values of quality parameters (flexibility, interoperability, and others).
* In second method three options are pursued to handle special purpose components, the last two options are based on widely used 80, 20 rule given by Pareto, and at the end to make comparisons the average of all the three options used, which makes the method very reasonable and statistically sound.
* While working out these methods the total time required to apply both the methods, and come out with quality parameter ratings for all the three architectures was about 2 hours. This was with an assumption that the complete idea of architectural designs was available in advance to the person applying the methods, which in any case is the prerequisite..

**Drawbacks:** These methods can be used only by people, who are knowledgeable in areas, like, software architecture, modularization and its properties, types of cohesions and coupling, and definitely the domains of software configuration management or domain of the architecture application (in general). This all is required as we need to do a rating by recognizing the cohesion and coupling types present in any component (module) of the architecture. That's why it is difficult to get ratings form multiple people and observe the deviations or do statistics on them.

## DISCUSSIONS AND CONCLUSIONS

Here we compare the results of the two methods, the results comprise of quantitative rating of various quality parameters present in three different architectures of CM tools: "2nd generation," CM Services Model" and the "Proposed Architecture Model". As per Table 2 the value of interoperability is significantly high in case of proposed model, this due to the presence of special components, C10 (Interface to other tools) and C11 (configuration data router). Same results could be seen for flexibility of proposed model, due to average module sizes and special component, C11 (configuration data router). In other parameters, like, traceability, testability, portability, the proposed model has also done the best. As we see "CM services Model," is best in case of simplicity, because it is offering discrete atomic service for each of the requirements (mostly representing configuration management activities). The proposed model having average number of modules and average module size, thus it is free from the problems of under-modularization and over-modularization. Whereas, the

CM services model suffers from over modularization, due to atomic module sizes and 2nd generation model suffers from under modularization, due to large module sizes. Big module sizes, inherently reduces the testability due to the presence of multiple features, which require large and complex test cases, which are difficult to compose and apply. In case of small module sizes, as present in "CM Services Model", the integration testing overheads (stubs and driver preparation) takes the efforts to the higher side. Although, no special attempt has been made in terms of introduction of special components in case of 'availability' in proposed model, but due to better cohesion and limited complexity the ratings are far better in case of the proposed model. This validates the fact that the proposed model is superior than the other models in majority of quality parameter of its architectural design.

## REFERENCES

1. Abhay Kothari, A.K. Ramani, 2004. Architecture for new generation configuration management tools and its validation. J. Systems Management, ICFAI Press, Hydrabad, India. (status: accepted for publication).
2. Roger S. Pressman, 1997. Software Engineering: A Practitioner's Approach. 4th Edn., McGraw-Hill International.
3. Estublier, J. Software Configuration Management: A Road Map. www.cs.ucl.ac.uk/staff/A.Finkelstein/fose/ finalestublier.pdf
4. Andr'e Van der Hoek, Dennis Heimbigner and Alexander L. Wolf, 1995. Does configuration management research have a future. Department of Computer Science, University of Colarado, Boulder Publication.
5. Susan Dart, 1992. Configuration Management Services. Software Engineering Institute-Carnegie Mellon University, USA.
6. Stephen A Mackey. Change Sets Revisited and Configuration Management of Complex Documents. Institute of Information Technology, National Research Council of Canada Publication.
7. Van der Hoek, A., D. Heimbigner and A.L. Wolf, 1996. A generic peer to peer repository for distributed configuration management. Proc. 18th Intl. Conf. Software Engineering. IEEE Computer Society.
8. Andr'e Van der Hoek, Carzaniga Antonio, Dennis Heimbigner and Alexander L. Wolf, 2002. A testbed for configuration management policy programming. IEEE Tran. Software Engineering, 28: 1.

9. Jacky Eslublier, 2001. Defining and supporting concurrent engineering policies in SCM. Proc. SCM-10.
10. Ronald van der Lingen and A. van der Hoek, 2003. Dissecting configuration management policies. Proc. Eleventh Intl. Conf. Software Configuration Management.
11. Richards S. Hall, Dennis Heimbigner, A. van der Hoek and A.L. Wolf, 1997. An architecture for post-development configuration management in wide-area network. SERL, Department of Computer Science, University of Colorado.
12. van der Hoek, A., 2001. Integrating configuration management and software deployment. Proc. Working Conf. Complex and Dynamic Systems Architecture.
13. Darcy W. Weber, 2001. Requirements for an SCM architecture to enable component based development. Proc. SCM-10.
14. Sundararajan Sowrirajan and A. Van der Hoek, 2003. Managing the evolution of distributed and inter-related components. Proc. Eleventh Intl. Workshop on Software Configuration Management.
15. Christensa B. Henerk, 2001. Tracking changes in rapid and extreme development: A challenge to SCM tools. Proc. SCM-10.
16. Abhay Kothari, 2005. Research Potential in Software Configuration Management Systems and Software Architecture: An Investigation. The ICFAI Journal of Systems Mangement.