International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 4– No.3, September 2012 – www.ijais.org

# Economic Effect of Cloning on Software Maintenance

Amanpreet Kaur Goraya
Hoshiarpur, Punjab
India

Ajitpal Singh Chela
Mahilpur, Hoshiarpur,
Punjab, India

## ABSTRACT
Software maintenance accounts for the majority of the total life cycle costs of successful software systems. Half of the maintenance effort is not spent on bug fixing or adaptation to changes of the technical environment, but on evolving and new functionality. The demand of software's has increased with the development of technology and communication systems. With this the maintenance effects which is a vital factor. The software's are not identical if we contrast them with past, present and future, due to development of new programming languages and their principles. To improve the quality of any software, maintenance is must. Cloning in source code files makes it difficult to modify. Several models are designed to overcome this problem. This paper presents the extension of analytical cost model to evaluate the cloning. As the size and the complexity of software increase, it also becomes essential to develop high-quality software, cost-effectively within a specified period. This paper presents a study on the cloned code, the large open source systems are used, various other new parameters are added to calculate clone.

## Keywords
Source Code, Clone, modifications, maintenance and fragment

## 1. INTRODUCTION
A code clone is a pair (or set) of code fragments in source files of a software product. It is pointed out that code clone makes software maintenance difficult. The code clone problems sometimes become serious one, especially for the large scale software. Maintenance thus preserves and increases the value that software provides to its users. Reducing the number of changes that performed during maintenance threatens to reduce this value. An important goal of software engineering is thus to facilitate the construction of systems that are easy-and thus more economic- to maintain.

The developer cannot even find out code clones by hand [20]. Successful software depends on the total cost of life cycle and software maintenance plays a main role in it. Duplicated code creates a problem in software development. Maintenance of software system is defined as changes of a software product after delivery to correct faults, to improve performance. The maintenance is the most luxurious phase of software life cycle. For example: SRS (Software Requirement Specifications) are read and changed often (for requirement elicitation, software design and test case specification) [2]. Now days programmers are busier they just develop the logic once and they reuse it after some modifications. Cost of medium and large software projects is calculated by the cost of developing the software plus the cost of hardware and supplies. The programmer delivers the largest amount of cloned data to the user in a system. It generates the large sized software but with less effort of programmer. Mainly cloning occurs in programming languages and in development work.

There are some root causes of cloning that exist in programming [21].

- Systems are modularized based on principles such as information hiding, minimizing coupling and maximizing cohesion.
- Programmers often reuse the copied code/ text as a template and then customize the template in the pasted content.

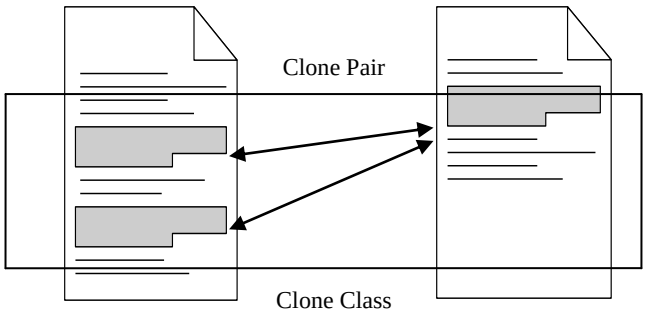


**Figure 1 Cloning**

Disadvantages of cloning are following:

- Several unwanted duplicates of code increase maintenance cost.
- Incompatible changes to cloned code can create error and lead to incorrect program performance.
- Maintenance is the most costly part of the software lifecycle.

## 2. LITERATURE SURVEY
Juergens and Deissenboeck [2] represents that duplication in source code has been recognized as a problem for software maintenance by both the research community and practioner as it increases program size and thus the effort for size related activities such as inspection that clone detection, a technique widely applied to source code, is promising to assess one important quality aspect in an automated way, namely redundancy that stems from copy &paste operations. Clone Tracker [19], developers can specify clone groups they wish to track, and the tool will automatically generate a clone model that is robust to changes to the source code, and can be shared with other collaborators of the project.

Rahman et al. [16] suggests that the analyses relationship between cloning and defect proneness. We find that, first; the great majority of bugs are not significantly associated with clones. Second, clones may be less defect prone than non-

cloned code. Finally, we find little evidence that clones with more copies are actually more error prone. He used four open source code projects- apache httpd, Nautilus, Evolution, Gimp. He also said that buggy code refers to a set of source code lines which were modified to fix a bug. ConQAT's [8] architecture conform assessment capabilities. The combination and implementation in an analysis framework is in our opinion a valuable contribution to the community. The graphical editor simplifies the creation of architecture descriptions and the interpretation of assessment results. The combination with the flexible ConQAT framework allows the assessment regarding different types of dependency with low configuration efforts.

Juergens and Deissenboeck [1] presents changes to code, such as a bug fix, often need to be performed to its duplicates as well, thereby increasing modification effort. If duplicates are missed when cloned code is modified, inconsistencies can be introduced into the system that can lead to faults, or existing faults can fail to be removed from the system. As long as we do not know the costs cloning causes, clone control is prone to be neglected- even though cloning could be the root cause and open bugs and change requests the symptoms. Roy and Cordy [7] presented the detailed survey of the state of the art in clone detection research. *First*, we describe the clone terms commonly used in the literature along with their corresponding mappings to the commonly used clone types. *Second*, it provides a review of the existing clone taxonomies, detection approaches and experimental evaluations of clone detection tools. Applications of clone detection research to other domains of software engineering and in the same time how other domain can assist clone detection research have also been pointed out. *Finally*, this paper concludes by pointing out several open problems related to clone detection research.

The paper [14] deals with the identification of duplicated parts in models. Like Triangle (gain) multiply with a constant. Circle (add), sum up their inputs. Squares have different meaning depending on their icon. It represents the techniques which are used to improve scalability by an adapted subsystem detection, to improve relevance of detected by clones by providing use case specific ranking and finally tool support to ease inspection of the instances of the detected clones.[15] Represents inconsistent clones constitute a major source of faults, which means that cloning can be a substantial problem during development and maintenance unless special care is taken to find and track existing clones and their evolution. Rahman et al. [16] analysed relationship between cloning and defect proneness. We find that, first the great majority of bugs are not significantly associated with clones. Second we find that clones may be less defect prone than non-cloned code. Finally, we find little evidence that clones with more copies are actually more error prone. Buggy code refers to a source code lines which were modified to fix a bug.

## 3. PRELIMIARIES

In a post-development phase, it is difficult to say which fragment is original and which one is copied and therefore, fragments of code which are exactly the same as or parallel to each other are called *code clones*. The term *clone* denotes similar code regions that contain redundant implementation of one or more concepts. A *clone group* is a set of clones. Clones in a single group are referred to as *siblings*. *Source statements* (SS) are the number of all source code statements, not taking commented or blank lines and code formatting into account. *Redundancy Free Source Statements* (RFSS) are the number of source statements, if cloned source statements are only counted once.

## 4. CLONE DETECTION TOOL

The Continuous Quality Assessment Toolkit ConQAT provides the tool-support required to enact continuous quality control in practice. Through its flexible architecture, ConQAT can be customized to address the quality requirements that are truly relevant for a software project. Flexible architecture means that length of the Clone varies according to the user. Either the length of clone fragment is less than 5 or more than 50. Small clones can be easily found out. There by, it helps to successfully counter quality decay of software systems. Modern programming languages offer various abstraction mechanisms to facilitate reuse of code fragments; copy-paste is still a widely used reuse strategy. CONQAT detects code fragments that differ up to utter or relative edit distance as clones. ConQAT was designed to provide support for the core activities in continuous quality control [22].

**Monitoring** To control quality, the current state of a system's quality needs to be assessed and monitored over time. To make quality control feasible, the wealth of assessment data generated during quality analysis needs to be aggregated and visualized in a comprehensive, yet concise manner. ConQAT supports this with a highly flexible composition mechanism and a rich set of building blocks that allows the rapid development of quality dashboards that integrate diverse quality analysis methods and tools.

**In-Depth Analysis** ConQAT provides a set of interactive tools that support the in-depth inspection of identified quality defects and help to prevent the introduction of further deficiencies. For this, ConQAT provides an advanced clone detection tool that can be integrated with different development environments.

**Tailoring** No two software projects are equal and today's software project is different from tomorrow. To support the adaption to changing quality requirements, ConQAT serves as a development platform that allows the efficient and effective development of innovative analyses as well as the project-specific tailoring of existing ones.

Code is behaviorally equal but not representational similar. [5] This often leads to numerous duplicated code fragments so called clones—in large software systems. Cloning is problematic for software quality for several reasons:

- Cloning unnecessarily increases program size and thus effort for size-related activities such as inspections or tests.
- Changes, including bug fixes, to one clone typically need to be made to the other clones as well, again increasing required effort.
- If changes to duplicated source code fragments are performed inconsistently, this can introduce bugs.

### 4.1 Background

We begin with a basic prologue to clone detection terminology. [11]

Definition 1: Code Fragment. A code fragment (CF) is any sequence of code lines (with or without comments). It can be of any granularity, e.g., function definition, begin- end block, or sequence of statements. A CF is identified by its file name and begin-end line numbers in the original code base and is denoted as a triple (CF.FileName, CF.BeginLine, CF.EndLine).

NUnit/src/NUnitCore/core/Extensibility/SuiteBuilderCollection.cs

```
namespace NUnit.Core.Extensibility
{
    /// <summary>
    /// SuiteBuilderCollection is an ExtensionPoint for SuiteBuilders
    /// implements the ISuiteBuilder interface itself, passing calls
    /// on to the individual builders.
    ///
    /// The builders are added to the collection by inserting them at
    /// the start, as to take precedence over those added earlier.
    /// </summary>
    public class SuiteBuilderCollection : ISuiteBuilder, IExtensionPo
    {
        private ArrayList builders = new ArrayList();

        #region Constructors
        /// <summary>
        /// Default constructor
        /// </summary>
        public SuiteBuilderCollection() { }
```

**(A)**

NUnit/src/NUnitCore/core/Extensibility/TestCaseBuilderCollection.cs

```
namespace NUnit.Core.Extensibility
{
    /// <summary>
    /// TestCaseBuilderCollection is an ExtensionPoint for TestCas
    /// and implements the ITestCaseBuilder interface itself, pass
    /// on to the individual builders.
    ///
    /// The builders are added to the collection by inserting them
    /// the start, as to take precedence over those added earlier.
    /// </summary>
    public class TestCaseBuilderCollection : ITestCaseBuilder, IEx
    {
        private ArrayList builders = new ArrayList();

        #region Constructors
        /// <summary>
        /// Default Constructor
        /// </summary>
        public TestCaseBuilderCollection() { }
```

**(B)**

**Figure 2 Cloning in two different files A and B**

**Definition 2: Code Clone.** A code fragment CF2 is a clone of another code fragment CF1 if they are similar by some given definition of similarity, that is, f(CF1) = f(CF2) where f is the similarity function (see clone types below). Two fragments that are similar to each other form a clone pair (CF1; CF2), and when many fragments are similar, they form a clone class or clone group.

**Definition 3: Clone Types.** There are two most important type of similarity between code fragments. Fragments can be similar based on the similarity of their program text, or they can be similar based on their functionality (independent of their text). The first type of clone is often the result of copying a code fragment and pasting into another location. The overall processing phases of cloning are Code Base, Preprocessed Code, Transformed Code, Clone on Transformed code, Clone Pairs, Filtered Clone Pairs and Filtered Clone Classes which are the major phases. Preprocessing, Transformation, Match Detection, Formatting, Filtering, Aggregation are the operations which perform on the source code. In the subsequent we endow with the types of clones based on both the textual (Types 1 to 3) and functional (Type 4) similarities:

**Type-1:** Identical code fragments except for variations in whitespace, layout and comments.

**Type-2:** Syntactically identical fragments except for variations in identifiers, literals, types, whitespace, layout and comments.

**Type-3:** Copied fragments with further modifications such as changed, added or removed statements, in addition to variations in identifiers, literals, types, whitespace, layout and comments.

**Type-4:** Two or more code fragments that perform the same computation but are implemented by different syntactic variants.

## 5. CAUSES FOR CLONING

Clones are typically created by copy & paste. Many different causes can trigger the decision to copy, paste (and possibly modify) an artifact fragment. There are two types (i) Inherent cause (ii) Maintenance Environment

### 5.1 Inherent Cause

Creating software is a difficult, intellectually challenging task. Inherent causes for cloning are those that originate in the inherent complexity of software engineering even ideal processes and tools cannot eliminate them completely. One inherent reason is that creating reusable abstraction is hard. It requires a detailed understanding of the commonalities and differences among their instances. When implementing a new feature that is similar to an existing one, their commonalities and differences are not always clear. A second reason is that understanding the effect of a change is hard for large software. An exploratory prototypical implementation of the change is one way to gain understanding of its impact.

### 5.2 Maintenance Environment

The maintenance environment comprises the processes, languages and tools employed to maintain the software system. Maintainers can decide to clone code to work around a problem. First, to reuse code, an organization needs a reuse process that governs its evolution and quality assurance. Missing or unsuitable reuse processes hinder maintainers in sharing code. In response, they reuse code through duplication. Second, short-sighted project management practices can trigger cloning. Third, to make code reusable in a new context, it sometimes needs to be adapted. Poor quality assurance techniques can make the consequences of the necessary changes difficult to validate.

## 6. CASE STUDY

We have to evaluating the cost of Open Source Systems Code (OSS). The three essential steps are mandatory:

- Acquire the Open Source System (OSS) on the internet.
- Calculate approximately the software lines of code (SLOC) in the OSS.
    - SLOC counters are language sentient and approximate lines of code by opening up every predictable source file and modularized each line as source, comments/ blank lines, depends on language specific recognition pattern.
- Use the analytical model [1] to evaluate cost by various new parameters.

There are few ways available to save maintenance like-abolish dead code and cloned code, try to avoid the bugs and reduce them, major area of consideration is their test activities, and reducing complexity leads to improve maintainability. Few types of code are also there –

1. New Code (which is manually written)
2. Reuse Code (Same as it is)
3. Adapted Code (with changes)
4. COTS (Commercial, Leased, Licensed Source Code)
5. Automatically Translated Code (Already existed code for which translation helped by automated tools)

**Table 1 CloneLOC is reported in softwares**

| Software | C# | PHP | JAVA |
|---|---|---|---|
| OS1 | 28,064 | 5,560 | 30,403 |
| OS2 | 2,335 | 15,413 | 9,992 |
| OS3 | 1,396 | 40,058 | 31,584 |
| OS4 | 924 | 2,407 | 15,403 |

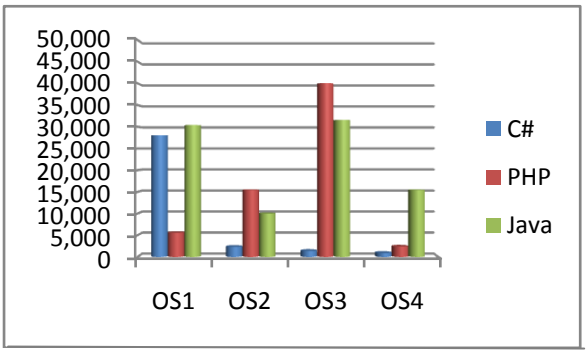


**Figure 3 Amount of CloneLOC in software's of different languages.**

Table 1 and Figure 3 show the amount of CloneLOC in software of different languages. The average amount of CloneLOC is in Java but in case of OS3 the PHP has maximum amount of CloneLoc.

Clone Code involves both reuse and adapted types of code. The parameters developed in this paper CloneLOC, CloneUnilts, CloneCount etc. with reuse taken into account overhead, detailed cost model, Quality Assurance, Software Maintenance process based on analytical cost model. In this clone study six open source systems are analyzed, two in Java, two in PHP and two in C#. The clone analysis in based on type-2 of cloning and the minimum fragment size is set to ten. The small size fragments can be easily detected without use of any tool.

In each OSS conqat run configuration (.cqr) file and .cqb (conqat block) file are required. .cqb displays the flow control of software. In this the input is given by source code and final output goes into the html presentation.

The connectivity of each node displays the previous node from where the input comes. The output of one node is input for next node. The graphical representation, different colors make it clear for the user to evaluate the amount of clone. It represents the value of several parameters.

Several new software metrics are developed for different concepts but for cloning SLOC is used. Source Line of Code (SLOC) is software metric used to calculate the size of a program. The two types of Source Line of Code (SLOC) are available, Physical and Logical SLOC. In this research work Physical SLOC is used because it is a count of text lines of program source code including comment lines, blank lines. Cloning has no limited area; it is widely used in source code either in the form of functions, identifiers, literals, comments etc.

Table 2 represents the result of evaluation of cloning. Every programming language has its own paradigms and principals to reuse the data. Reusability of data increase the fetching and execution speed of software. It creates problem when there are modifications in data. The difference between cloned and non-cloned data is: Cloned text can be read faster, whereas similar has read before. Another difference is reading a non-cloned text is much easier than cloned text.

## 7. FUTURE WORK

The origin of word cloning is usually related to the biology. On one side cloning is very difficult to identify in any sort of data where as on other side it is easier to fetch out the similar data which acts behaviorally and representational same. There is need to develop an application which will help to find out the cloned data like in various applications find and replace privileges are available. It makes system easier for the user because he can find the cloning from any software and then by using any cost evaluation model like COCOMO. They evaluate the actual cost of software. Now programmers do not make them mad.

## 8. CONCLUSION

The negative impact of cloning on program correctness has been stated qualitatively many times, its quantitative impact—and thus its significance—in practice remained unclear. Furthermore while cloning in source code had been studied intensely, little was known about its extent and consequences in other software. A lack of awareness of cloning is a threat to program correctness. While the analyzed systems varied in their share of unintentional differences—and thus the amount of cloning awareness among their developers—the negative impact of unintentionally inconsistent change was uniform: about every second unintentionally inconsistent change had a direct impact on program correctness. These results thus give strong indication that awareness of cloning is crucial during software maintenance. Clone control is required to achieve and maintain awareness of cloning to alleviate the negative impact of existing clones.

Cloning is not limited to source code and neither its negative impact. An analytical cost model quantifies the economic effect of cloning on maintenance efforts and field faults. It can be used as a basis for assessment and trade-off decisions. The model produces a result relative to a system without cloning and thus requires substantially less parameters—and

**Table 2 Evaluation Results**

| Software | Language | LOC | Clone LOC | Units | Clone Units | RFSS | Unit Coverage | Clone Count |
|---|---|---|---|---|---|---|---|---|
| Common Collection Code Analysis | Java | 109,415 | 30,403 | 41,582 | 16,949 | 30,673 | 0.408 | 1,608 |
| JRefCodeAnalysis | Java | 117,359 | 9,992 | 50,499 | 5,740 | 46,913 | 0.114 | 625 |
| Nunit | C# | 41,001 | 2,335 | 11,916 | 696 | 11,534 | 0.058 | 44 |
| PDFSharp | C# | 207,089 | 28,064 | 63,128 | 12,874 | 55,024 | 0.204 | 1,382 |
| CakePHP | PHP | 191,154 | 15,413 | 72,089 | 9,112 | 65,216 | 0.216 | 1,008 |
| Pyrocms | PHP | 137,463 | 40,058 | 53,434 | 25,043 | 30,103 | 0.469 | 7,891 |
| Clairion | Java | 116,077 | 31,584 | 66,379 | 23,019 | 48,937 | 0.347 | 5,004 |
| GLPK | Java | 19,148 | 15,403 | 9,895 | 9,469 | 1,399 | 0.957 | 576 |
| C#ID3Lib | C# | 12,667 | 1,396 | 3,470 | 600 | 3,116 | 0.173 | 41 |
| RPGMaker | C# | 17,817 | 924 | 9,366 | 554 | 9,050 | 0.059 | 42 |
| PHPFusion | PHP | 2,874 | 2,407 | 2,364 | 1,998 | 405 | 0.845 | 1,063 |
| PHPList | PHP | 49,126 | 5,560 | 23,146 | 2,520 | 21,444 | 0.109 | 348 |

instantiation effort—than general purpose cost models that produce absolute results. First, it completes our understanding of the impact of cloning: instead of focusing on isolated aspects or activities, it quantifies its impact on all maintenance activities and thus on maintenance efforts and faults as a whole. Second, it makes our observations, speculation and assumptions explicit. This explicitness offers an objective basis for scientific discourse about the consequences of cloning.

ConQAT provides support and flexibility for all phases of clone detection: from preprocessing, detection and post processing, to result presentation and interactive inspection in state of the art IDEs. ConQAT implements several novel detection algorithms: the first algorithm to detect clones in dataflow models; an index-based approach for type-2 clone detection that is both incremental and scalable. Clone detection is limited to copy & paste—independently developed program fragments with similar behavior are out of reach of existing clone detection approaches. During clone control, clone detection can be applied to find regions in artifacts that have been created through copy, paste & modify. It cannot be, however expected to detect behavioral similarities that have been implemented independently. Clone management tools, thus, cannot be expected to work on simions. Instead of facilitating their consistent evolution during maintenance, clone control thus needs to focus on the avoidance of simions.

## 9. REFERENCES

[1] Juergens, E. and Deissenboeck F. 2010, "How Much is a Clone?" 4th International Workshop on Software Quality and Maintainability. Spain.

[2] Elmar Juergens, Florian Deissenboeck, Can Clone Detection Support Quality Assessments of Requirements Specifications?, International Conference on Software Engineering (ICSE), May2010

[3] Pham N. H.; Nguyen, H. A.; Nguyen, T. T.; Al-Kofahi, J.M and Nguyen, T.N., 2009. "Complete and Accurate Clone Detection in Graph-based Models". Proceeding of 31st International Conference on Software Engineering, Vancouver, Canada, pp.276-286.

[4] Juergens, E., Deissenboeck, F., and Hummel, B., 2009 "CloneDetective – A Workbench for Clone Detection Research", Proceedings of 31st International Conference on Software Engineering, Vancouver, Canada, pp. 603-606.

[5] Juergens, E.; Deissenboeck, F. and Hummel, B., 2010 "Code Similarities Beyond Copy & Paste", Proceedings of 14th European Conference on Software Maintenance and Reengineering, Madrid, pp. 78-87.

[6] Deissenboeck, F.; Heinemann, L.; Herrmannsdoerfer, M.; Lochmann, K.; and Wagner, S. 2011. "The Quamoco Tool Chain for Quality Modeling and Assessment", Proceeding of 33rd International Conference of Software Engineering, Honolulu, USA. pp. 1007-1009.

[7] Roy C.K. and Cordy, J.R., 2007 "A Survey on Software Clone Detection Research", Technical Report 2007-541, Queen's University at Kingston Ontario, Canada, 115pp.

[8] Deissenboeck, F.; Heinemann, L.; Hummel, B.; Juergens, E.; 2010. "Flexible Architecture Conformance Assessment with ConQAT", Proceedings of IEEE 32nd International Conference of Software Engineering, Cape Town, Vol. 2, pp. 247-250.

[9] Duala-Ekoko, E. and Robillard, M.P., 2008 "CloneTracker: Tool Support for Code Clone Management" Proceeding of 30th International Conference on Software Engineering, Leipzig, pp. 843-846.

[10] Deissenboeck, F.; Wagner, S.; Pizka, M.; Teuchert, S. and Girard, J.F., 2007 "An Activity-Based Quality Model for Maintainability", Proceedings of International Conference on Software Maintenance, Paris, pp. 184-193.

[11] Roy C.K., Cordy J.R. and Koschke, R., 2009. "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach", Journal, Science of Computer Programming, Vol. 74, No.7, pp. 470-495.

[12] Michael Pfahler, Improving clone detection for models, Master Thesis, Nov 2009

[13] Holger, S.; Martin J. and Horst, L.; 2009 "Tool Support for User-Defined Quality Assessment Models", Proceedings of METRKON. www.docstoc.com/docs/79810751/Tool-Support-for-User-Defined-quality-Assessment-Models.

[14] Deissenboeck, F.; Hummel, B.; Juergens, E.; Pfahler, M. and Schaetz, B., 2010. "Model Clone Detection in Practice", Proceeding of 4th International Workshop on Software Clones (IWSC), New York, U.S.A., pp. 37-44.

[15] Juergens, E.; Deissenboeck, F.; Hummel, B. and Wagner, S., 2009 "Do Code Clone Matters?" Proceedings of 31st International Conference on Software Engineering, Vancouver, B.C., pp. 485-495.

[16] Rahman, F.; Bird, C. and Devanbu, P., 2010 "Clones: What is that Smell?" Proceedings of 7th IEEE working conference on Mining Software Repositories., Cape Town, pp. 72-81

[17] Higo, Y.; Ueda Y.; Kamiya, T.; Kusumoto, S.; and Inoue, S., 2002 "On Software Maintenance Process Improvement Based on Code Clone Analysis", Proceedings of 4th International Conference on Product Focused Software Process Improvement, London.

[18] Ueda, Y.; Kamiya, T.; Kusumoto, S.; Inoue, K., 2002 "Gemini: Maintenance Support Environment Based on Code Clone Analysis", Proceedings. Eighth IEEE Symposium on software metrics, pp. 67-76.

[19] Duala-Ekoko, E. and Robillard, M.P., 2008 "CloneTracker: Tool Support for Code Clone Management" Proceeding of 30th International Conference on Software Engineering, Leipzig, pp. 843-846.

[20] Code Clone Related Tools Software Engineering Laboratory, Department of Computer Science, Graduate School of Information Science and Technology, Osaka University, March, 2005

[21] Rainer Koschke, Survey of Research on Software Clones, Dagstuhl Seminar Proceedings, 2007

[22] Deissenboeck, F. and Feilkas, M. 2010 ConQAT Book, TUM, Technische Universität München.

[23] Hummel, B.; Juergens, E. and Steidl, D., 2011. "Index-Based Model Clone Detection", Proceeding of 5th International Workshop on Software Clones, Honolulu, USA, pp. 21-27.

[24] Juergens, E.; Deissenboeck, F.; Feilkas, M.; Hummel, B.; Schaetz, B.; Wagner, S.; Domann, C. and Streit, J., 2010 "Can Clone Detection Support Quality Assessments of Requirements Specifications?", 32nd International Conference on Software Engineering, Cape Town, Vol. 2, pp. 79-88.

[25] Koschke R., 2008 "Survey of Research on Software Clones", Duplication, Redundancy and Similarity in Software, Dagstuhl Seminar Proceedings, pp. 24.

[26] Koschke R.; Falke, R. and Frenzel, P., 2006. "Clone Detection Using Abstract Syntax Suffix Trees", Proceedings of 13th Working Conference on Reverse Engineering, Benevento, Italy, pp. 253-262.