

Ripple Effect: A Complexity Measure for Object Oriented Software

Haider Bilal¹
Supervised by: Dr Sue Black¹

¹ Centre for Systems and Software Engineering,
Faculty of Business, Computing and Information Management,
London South Bank University
{Bilalhz, Blackse}@lsbu.ac.uk

Abstract. Software metrics can provide us with information regarding the quality of software. The ripple effect metric shows what impact changes to software will have on the rest of the system. The computation of ripple effect is based on the effect that a change to a single variable will have on the rest of a program; it provides a measure of the program's complexity. The original algorithm used to compute ripple effect has been reformulated to provide clarity in the operations involved and the measurement of ripple effect for procedural software. This paper describes the ripple effect metric and considers its applicability as a software complexity measure for object oriented software. Extensions are proposed to the computation of ripple effect to accommodate different aspects of the object oriented paradigm.

Keywords: Software Metrics, Change Impact Analysis, Ripple Effect, OOP.

1 Introduction

As the software industry has matured, resources have shifted from being devoted to developing new software systems to making modifications to evolving software systems: software maintenance. A major problem for developers in a changing environment is that small changes can ripple through software to cause major unintended impacts elsewhere. Therefore, software developers need mechanisms to understand how a change to a software system will impact the rest of the system. This process is called change impact analysis (CIA) [9]. Making software changes without understanding their effects can lead to unreliable software products. CIA can be used to reduce the amount of maintenance required; thereby increasing the reliability of the software, since fewer errors would have been introduced.

CIA information can be used for planning changes, making changes and tracing through the effects of changes. Research into impact analysis has been concerned mostly with procedural software: function-based programs not class-based. However, this work will be concerned with object oriented software, since most current software development projects commonly use object oriented programming rather than procedural programming. Object oriented software is all about objects that are found in the real world. An object can be thought of as a black box that can receive and send messages. A black box is based on the concept of data structure that encapsulates a set

of routines, called methods, which operate on the data [9]. In procedural software, code and data are kept apart. For example, in the C language, functions (units of code) and structures (units of data) are not formally connected. This is not the case for object oriented software where code and data are combined into a single object. Object oriented software is intended to be more modular and therefore, more maintainable and reusable than procedural software.

2 Technical Problems

Thus far, research built on the work of Black [3] has focused on the automatic computation of ripple effect measures within a practical timescale for procedural software, using the C programming language. Yau and Collofello's original ripple effect algorithm was reformulated in 2001, using matrix arithmetic, to simplify the calculation and make it more explicit [3]. A tool, REST (Ripple Effect and Stability Tool), was produced which uses an approximation algorithm to compute the ripple effect for the C programming language [3]. The current research focuses on measuring ripple effect for C++ object oriented software and possibly software written using other object oriented programming languages, for example Java.

By being able to compute ripple effect for object oriented software to identify potential impacts before making a change pays high dividends by reducing the risk of costly rework and the potential for introducing further errors in the software [9]. Therefore, the risks associated with embarking on a costly change can be reduced, since the cost of unexpected problems generally increases with the lateness of their discovery [9]. Carrying out ripple effect computation for object oriented software will allow an assessment of the cost of the change and help management to choose between alternative changes. It will also allow managers and engineers to evaluate the appropriateness of a proposed modification. If a proposed change has the possibility of impacting large, disjoint sections of a program, the change will need to be re-examined to determine whether a safer change is possible [9].

3 Related and Prior Work

In 1978, Yau and Collofello introduced their version of ripple effect which uses ideas from Haney [7], Myers [12] and Soong's [13] work. It is proposed as a measure of complexity as opposed to probability, which could amongst other things be used during software maintenance to evaluate and compare various program modifications to source code [15]. Computation of ripple effect involved using error flow analysis where all program variable definitions involved in an initial modification represented primary error sources from which inconsistency could propagate to other program areas. Propagation continued until no new error sources were created.

The computation of ripple effect was reformulated in 2001 to make the calculation more explicit [3]. The reformulation revealed how the algorithm's structure can be broken down into separate parts thus providing clarity and enhancing the understanding of its structure. To facilitate the software implementation of the new

algorithm an approximation was made, greatly simplifying the calculation that is important for automatic ripple effect computation.

Elish and Rine [6] present an algorithm for computing ripple effect for object oriented programs at the design level, i.e. at a more coarse grained level than the ripple effect presented in this paper. Chauman et al [5] study the impact of changes across an object oriented system written in C++ by making one change at a time and studying the resulting impact. Li and Offut [10] carry out CIA for object oriented programs with the aim of highlighting modules that need to be re-tested during regression testing.

4 Research Hypothesis

The Previous work [3] focused on the automatic computation of ripple effect for procedural software. However, this work will show:

- That it is applicable and useful to automatically compute ripple effect for object oriented software.
- The ability to validate the use of ripple effect to measure the stability of an object oriented software system after going through a modification.

5 Research Contributions

Software maintenance has been recognized as the most costly phase in the software life cycle [11]. This work has the potential to improve the maintenance of object oriented software, thereby reducing its cost. Measurement of object oriented software using ripple effect computation can help in determining the quality of the software [2], predicting the maintainability of the software and validating best practices for software development.

It is not always clear where modifications will have to be made to code, or what the impact of any type of change to code may have across the whole system. Ripple effect computation shows the maintainer what the effect of any change will be on the rest of the software system. It highlights modules with high ripple effect as possible problem modules. This proposed research offers the potential to improve the stability and efficiency of object oriented software and cut the cost of software maintenance. It can, also, give useful feedback which can then be used to improve future versions of the software system.

6 Ripple Effect Computation

The current computation of ripple effect is based on the effect that a change to a single variable will have on the rest of the program, it is used to determine the scope of the change and to provide a measure of the program's complexity. The effect of the

change may not necessarily be local to the modification, but may also propagate to other parts of the program.

There are two types of change propagation that are used to calculate ripple effect values [3]:

- **Intramodule change propagation:** Propagation from one variable to another within a function, (Fig. 1), e.g. propagation between y , x and z in Function1.
- **Intermodule change propagation:** Propagation from one function to another, (Fig. 1), e.g. propagation of z from Function1 to Function2.

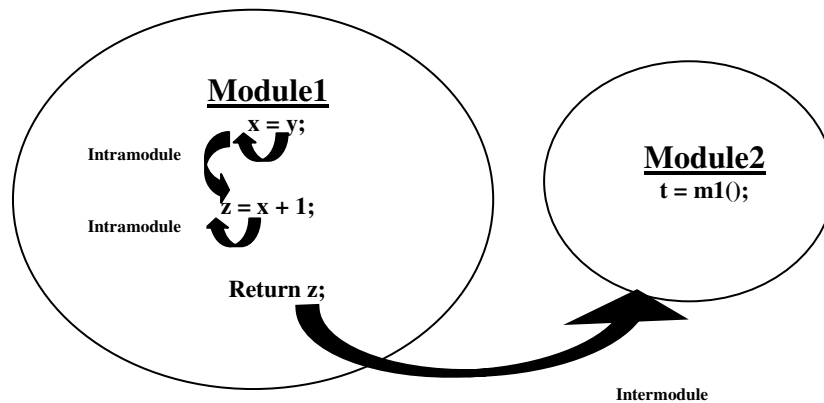


Fig. 1. Intramodule & Intermodule Change Propagation for Procedural Software

Intramodule change propagation is used to identify all variables which are affected by ripple effect as a consequence of a modification within the module. Intramodule change propagation uses information from *assignment*, i.e. change propagation from the right-hand side of an assignment to the left-hand side; and *definition-use*, i.e. change propagation from the definition of a variable to subsequent use of that variable, (Fig. 2). The combination of information from assignment and definition-use pairings supply the required information for calculating intramodule change propagation.

Once a variable is affected, the flow of program changes crosses module boundaries. Intermodule change propagation is then used to calculate all affected modules that are involved in the intermodule change propagation as a consequence of the modified variable.

7 Object Oriented Obstructs

The computation of intramodule change propagation of variables for functions in a procedural language is being equally applied to methods in object oriented software, i.e. each method is being considered as a module in the same context.

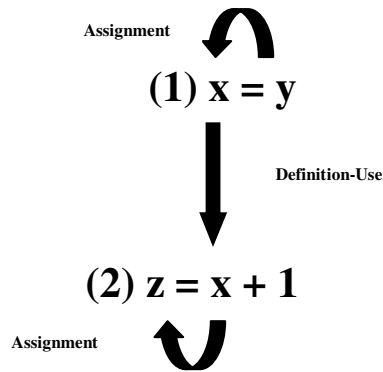


Fig. 2. Assignment and Definition-Use Pairings

For the intermodule propagation, a range of method call types will be recognized and used by the ripple effect computation. For example, in C++, these call types can take one of the following forms: Member-functions called in the same class or superclass; Public member-function calls on another object; Calls to static member-functions; Calls to constructors.

Building on the work of Black [3], ripple effect will be computed for object oriented software. The following characteristics of object oriented software have been identified as being important for the calculation of ripple effect:

1. Implicit Parameters [4].
2. Polymorphic Function/Method Calls [4].
3. Class Relationships & Links [8].

So far, there appear to be no major obstacles to computing ripple effect for object oriented software. The most important issues that needs further investigation and resolving have been the treatment of implicit variables in all object calls and the intermodule change propagation computation for polymorphic calls.

8 Implementation Model

The proposed research is, therefore, the implementation and possible reformulation of the ripple effect algorithm to produce the completely automatic computation of ripple effect for object oriented software and the subsequent evaluation of its potential benefits.

The following software analysis tools are being used to assist in collecting the required information for the measurement of ripple effect for object oriented software systems:

- **REST** [3]: A software tool that was developed at the Centre for Systems and Software Engineering to automate the production of ripple effect measures for C

code. A C++ parser for REST is currently being developed to allow computation of ripple effect for object oriented software.

- **CodeSurfer** [14]: A C/C++ source code analysis and navigation tool. It is a code browser produced by GammaTech that understands pointers and indirect function calls. It can be used for program understanding, maintenance, impact analysis, re-engineering and reuse [1].

Using the above software analysis tools, different versions of the ripple effect algorithm for object oriented software will be implemented and compared for validation, (Fig. 3):

1. Concatenating all code within a class, omitting calls to local methods. Calculating ripple effect between this class and other classes, (i.e. ripple effect calculation at the *class* granularity).
2. Looking at ripples across methods and classes, ignoring all propagations within methods, (i.e. ripple effect calculation at the *class* granularity, taking methods into account).
3. Looking at ripples across methods and classes, calculating all propagations within methods, between methods and between classes, (i.e. ripple effect calculation at both *method* and *class* granularity).
4. Looking at ripples across methods within each class, ignoring all propagations between classes, (i.e. ripple effect calculation at the *method* granularity).

An example application of computing ripple effect for a small C++ program has already shown that it is applicable and useful [4]. However, to compute ripple effect for the object oriented program using the current REST parser, the C++ code had to be first converted to C. This involved removing all classes from the code and converting all member-functions and member data into regular C functions and global variables respectively [4]. A future version of the REST tool may be needed in order to parse the C++ code to compute the ripple effect directly without the need of conversion.

9 Evaluation and Credibility

Work will be evaluated as it progresses by comparing the output of the implementation of all 4 different versions of the ripple effect algorithm and the difference in the ripple effect measure between object oriented and procedural software systems. To the benefit of the author, who has come to this work with almost five years of industrial experience in software maintenance, the ideas proposed and future results can be thoroughly debated and self-criticized.

This work has been presented at the ASTReNeT workshop [16] and at the 'Research Student Mini-Plenary Day' at London South Bank University. Further credibility will be derived, as the work progresses, from attending future events, workshops and conferences, for example: SQM 2006, CSM 2006, ICSE 2007. Results achieved will be presented to the software engineering community in the form of journal papers, for example the IEEE Transactions on Software Engineering and the Journal of Software Maintenance and Evolution.

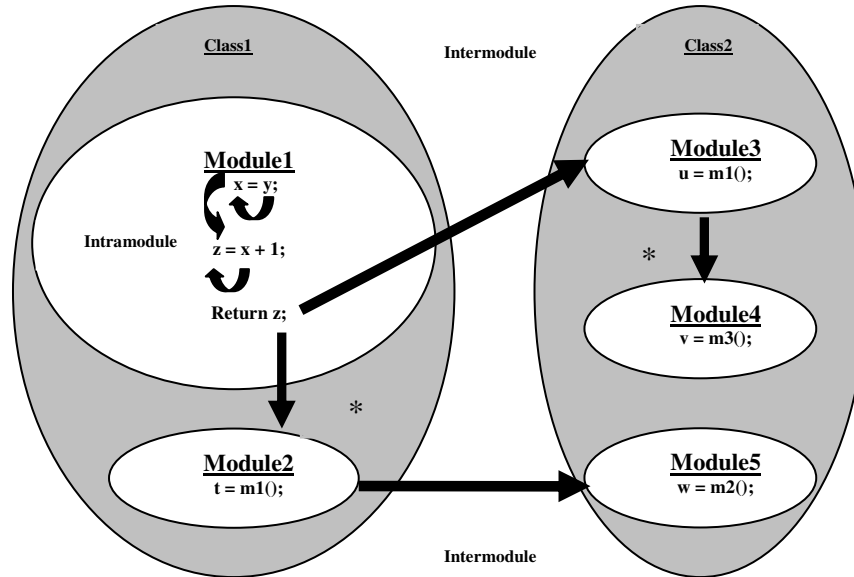


Fig. 3. Intramodule & Intermodule Change Propagation for Object Oriented Software (* intra-module/intermodule change propagation depending on the version of the ripple effect algorithm used)

10 Conclusion and Future Work

Because software now plays a very important role in our lives we need to ensure that our software products are of good quality. Using CIA and specifically ripple effect as part of a software measurement program can give useful feedback which can then be used to improve future iterations of the product. Previous work has concentrated on measuring ripple effect for procedural software. This research will focus on implementing ripple effect measurement for object oriented software, for example C++, to ensure that the quality of the software is enhanced and maintained. A brief description of CIA, object oriented paradigm and ripple effect have been given. Explanation of the two fundamental features of ripple effect computation: intramodule and intermodule change propagation have been presented. Also, object oriented constructs relevant to computing ripple effect have been discussed.

The ideas presented in this paper will be taken further by drawing up a much more detailed framework of ripple effect measurement for object oriented software. For this work to be useful, guidelines for the practical implementation of the ideas presented are being drawn up and will be utilized. This work will enable and show the automatic computation of ripple effect for object oriented software.

References

1. Anderson, P., Reps, T., Teitelbaum, T., Zarins, M.: Tool Support for Fine-Grained Software Inspection. *IEEE Software* 20(4): 2003, 42-50
2. Bilal, H.Z., Black, S.E.: Using the Ripple Effect to Measure Software Quality. SQM 2005, Cheltenham, Gloucestershire, UK, 21st-23rd March 2005
3. Black, S.E.: Computation of Ripple Effect Measures for Software. Ph.D. thesis, London South Bank University, London, UK, 2001
4. Black, S.E., Rosner, P.E.: Measuring Ripple Effect for the Object Oriented Paradigm. IASTED International Conference on Software Engineering, 15th-17th Innsbruck, Austria, February 2005
5. Chaumon, M.A., Kabaili, H., Keller, R.K., Lustman, F.A.: Change Impact Model for Changeability Assessment in Object-Oriented Software Systems. *Science of Computer Programming*, 45(2-3), 2002, 155-174
6. Elish, M.O., Rine, D.: Investigation of Metrics for Object Oriented Design Logical Stability. In Proceedings of the Seventh European Conference on Software Maintenance and Reengineering, 26-28 March 2003, 193-200
7. Haney, F.M.: Module Connection Analysis - a Tool for Scheduling of Software Debugging Activities. Proceedings Fall Joint Computer Conference, 1972, 173-179
8. Kabaili, H., Keller, R.K., Lustman, R.A.: Change Impact Model Encompassing Ripple Effect and Regression Testing. In Proceedings of the Fifth International Workshop on Quantitative Approaches in Object-Oriented Software Engineering, Budapest, Hungary, 2001, 25-33
9. Lee, M.L.: Change Impact Analysis of Object-Oriented Software. Technical Report ISE-TR-99-06, George Mason University, 1998
10. Li, L., Offutt, A.J.: Algorithmic Analysis of the Impact of Changes to Object-Oriented Software. In Proceedings of the International Conference on Software Maintenance, IEEE, Monterey, CA, USA, November 1996, 171-184
11. Lientz, B.P., Swanson, E.B., Tompkins, G.E.: Characteristics of Application Software Maintenance. *Communications of the ACM* 21(6) 1978, 466-471
12. Myers, G.J.: A Model of Program Stability. Van Nostrand Reinhold Company, 135 West 50th Street, NY 10020, Chapter 10, 1980, 137-155
13. Soong, N.L.: A Program Stability Measure. Proceedings 1977 Annual ACM conference, Boulder, Colorado, 1977, 163-173
14. Teitelbaum, T., Reps, T. CodeSurfer. GrammaTech Inc., <http://www.grammatech.com/products/codesurfer/overview.html>, last accessed 9th April 2006
15. Yau, S.S., Collofello, J.S.: Some Stability Measures for Software Maintenance. *IEEE Trans. On Software Engineering*, Vol. SE-6, No. 6, November 1980, 545-552
16. 2nd Analysis, Slicing & Transformation Network Workshop, ASTReNet, London, UK, June 2005, <http://www.dcs.kcl.ac.uk/staff/zheng/astrenet/index.html>, last accessed 9th April 2006