A Change Impact Model Encompassing Ripple Effect and Regression Testing

Hind Kabaili, Rudolf K. Keller and François Lustman
Département IRO
Université de Montréal
C.P. 6128, succursale Centre-ville
Montréal, Québec H3C 3J7, Canada
E-mail: {kabaili | keller | lustman}@iro.umontreal.ca

Abstract

Maintenance is one of the major concerns of software developers and industries. The success of this critical phase depends highly on the changeability of the software. In order to evaluate the flexibility of a software to accommodate changes, we have defined a change impact model for object-oriented systems. As previously defined, it calculates the impacted classes due to an atomic change. In this paper we present an extension of the change impact model to obtain a fine-grained assessment of system changeability. As a first extension, we take into account the classes that are impacted by ripple effect. In a second extension, we also count classes that need to be re-tested even if they are not directly affected. An experimentation is planned to study the relationship between software's changeability assessed by the extended change impact model and design properties.

Keywords: object-oriented, changeability, design properties, change, impact, ripple effect, regression testing.

1. Introduction

Changeability, major subject of this paper, is key in application areas such as telecommunications, in which software systems are evolving at a rapid pace. Moreover there are organizations which do not develop the software they operate, but purchase it. Thus they are interested in a software's ability to sustain an on-going flow of changes. In this paper, we define changeability of a system as its capacity to absorb changes.

In the SPOOL[1] project, a joint project with Bell Canada, we are investigating the relationship between the changeability of software systems and their design. The hypothesis that design

[1] This research was supported by the SPOOL project organized by CSER (Consortium for Software Engineering Research) which is funded by Bell Canada, NSERC (National Sciences and Research Council of Canada), and NRC (National Research Council of Canada).

properties have an influence on changeability has received a first confirmation in [CKK+00]. This encourages us to pursue this matter further. The change impact model used in previous work is basic [CKK+00] [KKL01]. To obtain a fine-grained assessment of system changeability, we decided to extended the change impact model in two directions: ripple effect and regression testing.

In the remainder of this paper, we first give an overview of the change impact model and the major class dependencies used throughout the paper. Section 3 describes the ripple effect model, while Section 4 presents the regression testing model. In section 5, we introduce the set up of the on-going experimentation with the extended change impact model. The last section summarizes the paper work and provides an outlook into future works.

2. Overview of Change Impact Model

The objects artifacts available in the design phase are classes and their relationships. A class is defined as a group of variables and a group of methods. A change applies to a class, to a variable or to a method. Examples of a change are adding a variable, changing a method's scope from public to protected or removing the relationship between a class and its parent. The changes considered in this paper are atomic changes, and we call impact of change (or impacted classes) the set of classes that require correction as a result of that change. The impact depends on the type of the change and the type of the relationship between classes. The impacted classes are linked by one of the following link to the changed class: association (**S**), aggregation (**G**), inheritance (**H**) and/or invocation (**I**). We introduce an artificial link called "local" (**L**) to denote an impact inside the changed class. Since, the systems under consideration are in C++, we added the friendship link (**F**) to reflect the existence of this feature in C++.

In Table 1, we illustrate these links to better understand the reasoning used in the next sections. Ci is a class, Ci_vj is a variable of class Ci and Ci_mj is a method of class Ci:

(1) C2 is associated (**S**) to C1 if one of the following declarations appears in the body of C2:

```
class C2 {
  C2_m1( ) {
    C2_v1 = a.C1_v1; //a is an object of class C1
    C2_v1 = b->C1_v1; // b is an object of class C1
    …}
  …
  // C1_v1 is used in parameter list
  C2_m2 (…, a.C1_v1, …);
  C2_m3 (…, b->C1_v1, …);
   …
};

  //a method of C2 return an object of class C1
C1 C2::C2_m4(….){…};

// a method of C2 call a method using an object of
// class   C1
…C2::C2_m4 (…){
  …func(…, a, …);
…}
```

(2) C2 is linked by invocation (**I**) to C1 if one of the following declarations appears in the body of C2:

```
class C2 {
  …
  // o1 is either a object or
  // object reference of type C1.
  o1.C1_m (…);
  o1->C1_m (…);
…};
```

(3) C2 inherits (**H**) from C1 in one of the following manners:

```
// in a private manner,
class C2: private C1 {…};
// in a public manner,
class C2: public C1 {…};
// in a protected manner.
class C2: protected C1{…};
```

(4) C2 is aggregated (**G**) to C1 if one of the following declarations appears in the body of C2:

```
class C2 {
  // An instance of C1 is part of C2
  C1 a;
  // b is a declared variable of type pointer to C1
  C1 * b;
…};
// In constructor,
C2::C2() {
// b is dynamically created.
b = new C1();}
```

(5) C2 is a friend (**F**) of C1 if the following declaration appears in the body of C1:

```
class C1 {
  …
  // C2 can access any member of C1.
  friend class C2;
  …};
```

Table 1: Definitions of the links

Thus, for a given change ch_i in class cl_j, the set of impacted classes is expressed as a Boolean expression in which the variables stand for the links. For example, the impact formula for such a hypothetical change may be given by

Impact (cl_j, ch_i) = **SH'+G**,

meaning that classes which are in association (**S**) with, and not derived (**H'**) from the changed class cl_j, or classes which are in aggregation (**G**) with cl_j, are impacted. A complete description of the change impact model will be found in [Cha98].

It is worth noting that we search only for direct impact, i.e., we search for impact only in classes which directly interact with the changed class.

3. Ripple Effect Model

The ripple effect of a change to the source code of a software system is defined as the propagation of its impacts on the rest of the system. In 1972, Haney mentioned for the first time the ripple effect in software engineering [Han92]. He used a probabilistic connection matrix which subjectively models the dependencies between modules to determine how a change in one module necessitates changes in the others. Yau and Collofello introduced a stability measure [YC80]. They define stability of a system as the resistance to the potential ripple effect that the system would have when it is modified. Black reformulated Yau and Collofello's algorithm to improve the computation process [Bla99]. Li and Offut defined potential changes in an object-oriented software, then they presented five algorithms to calculate the impact for these changes and their ripple effect [LO96].

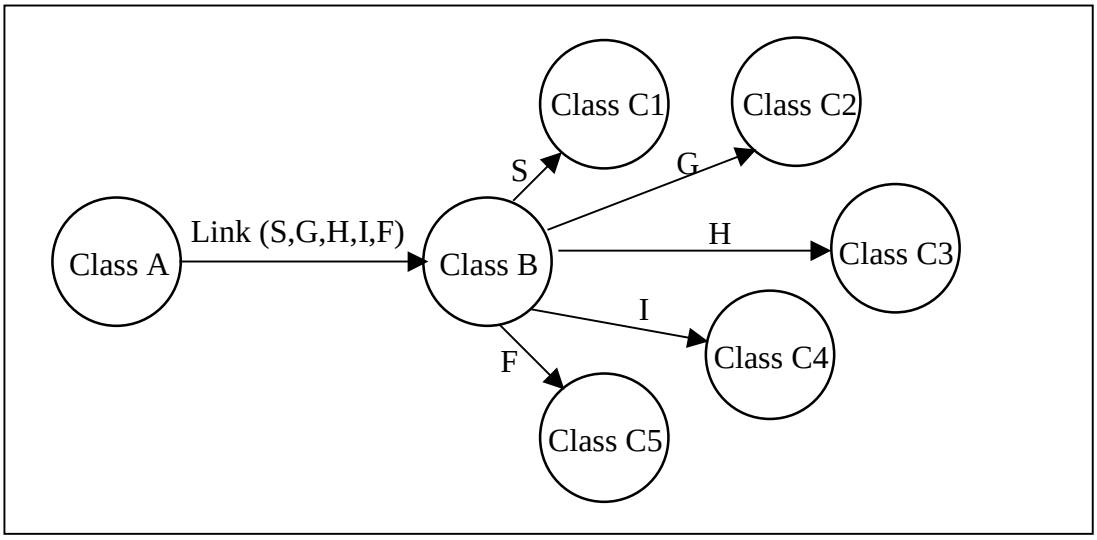

Figure 1: Dependencies class example

The concept of ripple effect is better described in Figure 1. Initially, Class_A is changed, impacted classes are calculated with the change impact model. Class_B is one of the impacted classes. To determine the ripple effect (impacted classes among Class_Ci, i=1,…5) of the initial change, we have to answer the question: "How is Class_B affected?". We propose a ripple effect model consisting on two steps. In the first step, we try to identify the nature of the change or changes in the class directly impacted (Class_B in Figure 1). Once this achieve, the original change impact model is applied to class_B, based on the change or changes identified in step 1 above. In Class_B, change identification is done by reasoning on the type of the link between Class_A and Class_B. Below it is an example for calculating a ripple effect:

(a) association link:

Let's say that Class_B is impacted by a change ch_i in Class_A. If Class_B is linked by association to Class_A, it means that Class_B refers to one or more variables of Class_A trough the signature, the return type or the body of one of its methods (see Table 1).

Class_B may be affected by one of these changes:

- ch_1: Method signature change
- ch_2: Method implementation change
- ch_3: Method return type change

According to the impact formula of the change impact model,

- *Impact* (Class_B, ch_1) = **I + H**
- *Impact* (Class_B, ch_2) = (-) (no impact)
- *Impact* (Class_B, ch_3) = **I + H**

Consequently, the ripple effect of ch_i is the union of the impacts of ch_1, ch_2, and ch_3:

- *Impact* (Class_B, ch_i) = **I + H**

The ripple effect model has been validated on 31 class system. Each ripple effect formula was tested for several atomic changes. The changes are implemented in the code one at a time. The impact is obtained by re-compiling the code. A formula is validated if the set of classes that need correction is included in the set of classes obtained by the ripple effect formula.

4. Regression Testing Model

Regression testing is the process of testing changes to programs to make sure that the modified code behaves correctly, and that modifications have not affected existing functionalities. Unlike development testing, it uses existing test suites and tests only the components that might be affected by modifications.

The regression test process consists of 5 phases, (1) identification of changed classes, (2) identification of affected classes, (3) generation of a class test order, (4) selection of test cases, and (5) test case modification and generation [KGH+95].

Most regression testing techniques are based on the call graph concept [WL92], the control flow graph concept [RH97] [WCK99], the dependence graph concept [RH94] and the class firewall concept [KGH+95].

In this work, we address the problem of determining classes to retest. Our approach is based on class dependencies presented in Section 2. Let's consider the graph in Figure 1 as an example. Class_A is changed. In which case the related Class_B needs to be re-tested ? If Class_B need to be changed, then it is identified by the change impact model. However, even if Class_B is not impacted, in some cases its behavior might be affected by the change in Class_A. Thus, it should be tested to ensure that it did indeed not change in behavior.

The behavior of a class may change if one of the following declaration appears in the body of Class_B:

(1) **Class_A** Meth (…){…}; //A method of Class_B return an object of type Class_A

(2) … Meth (…,Class_A **obj**,…){…}; //A method of Class_B has an object of type Class_A as argument

(3) … Meth (…){ …**objA.var**…}; ou … Meth (…){ …**objA->var**…}; //A method of Class_B uses a variable of an object of type Class_A

(4) … Meth (…){ …**objA.func(…)**…}; ou … Meth (…){ …**objA->func(…)**…}; // A method of Class_B calls a method of an object of type Class_A

(5) … Meth (…) { … func(…,**objA**,…); …); // A method of Class_B calls a method that uses an object of type Class_A

Cases (1), (2), (3) and (5) represent the association link. Case (4) represents invocation. Thus, for every considered change in Class_A, classes linked by association and/or invocation link to Class_A need to be re-tested.

5. Experimental Setup

Currently, we are putting together an experimentation with the extended change impact model. The goal of the experimentation addresses the needs of our research, which is finding some design properties that can be used as changeability indicator. The change impact model counts 66 changes, we plan to choose a change subset to increase the batch of changes, since it is not realistic to consider the whole changes. A set of design metrics will be used to quantify the design properties. Actually, the pool of design metrics under consideration consists of C&K metrics and their extension [CH94], MOOD set of metrics [AGE95] and Briand et al. metrics [BDM97]. We considered these three set of metrics because they are based on different approaches. As systems under test, we actually have three C++ systems, and a forth one is envisioned.

6. Conclusion

In this paper we extend the change impact model defined in [Cha98]. The extension consists in taking into account the ripple effect and the regression testing to obtain a fine-grained assessment of system changeability. The ripple effect model identifies the affected classes among classes which are not linked to the changed class (called impacted classes). The affected classes are identified by predicting the possible changes in the impacted classes. Then, the change impact model is applied. The approach used in regression testing detects classes that need to be re-tested without necessitating corrections. We suspect that the behavior of such classes may be changed.

An experiment is currently designed to explore the relationship between changeability and design metrics of the pool mentioned above. It will use the enhanced change impact model. It will be interesting to see if this more precise model confirms or not the relationship already detected [CKK+00] and allows us to find new ones. As future work, we will try to upgrade the concept of atomic change to the more realistic notion of change request.

7. Reference

[AGE95] Fernando Brito Abreu, Miguel Goulâo, and Rita Esteves. Toward the Design Quality Evaluation of Object-Oriented Software. In *Proceedings of the 5th International conference on Software Quality,* Austin Texas, October 1995.

[BDM97] Lionel Briand, Prem Devanbu, and Walcelio Melo. An Investigation into Coupling Measures for C++. In *Proceedings of the 9th International Conference on Software Engineering*, Boston, MA, pages 412-421, May 1997.

[Bla99] Sue Black. Measuring Ripple Effect for Software Maintenance. In *Proceedings of International Conference on Software Maintenance,* Oxford, England, August 1999. IEEE.

[Cha98] Ajmal Chaumun. Change Impact Analysis in Object-Oriented Systems: Conceptual Model and Application on C++. *Master's thesis*, November 1998.

[CK94] Shyam R. Chidamber and Chris F. Kemerer. A Metrics Suite for Object Oriented Design. In *IEEE Transactions on Software Engineering*, 20(6):476-493, June 1994.

[CKK+00] M. Ajmal Chaumun, Hind Kabaili, Rudolf K. Keller, François Lustman, and Guy St-Denis. Design Properties and Object-Oriented Software Changeability. In *Proceedings of the 4th European Conference on Software Maintenance and Reengineering*, pages 45-54, Zurich, Switzerland, February 2000. IEEE.

[Han72] F. M. Haney. Module Connection Analysis - a Tool for Scheduling Software debugging activities. In *Proceedings of AFIPS Joint Computer Conference*, pages 173-179, December, 1972.

[KKL01] Hind Kabaili, Rudolf K. Keller, and Francois Lustman. Cohesion as Changeability Indicator in Object-Oriented Systems. In *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, pages 39-46, Lisbon, Portugal, March 2001. IEEE.

[KGH+95] D. C. Kung, J. Gao, P. Hsi, J. Lin, and Y. Toyoshima. Class Firewall, Test Order and Regression Testing for Object-Oriented Programs. In *Journal of Object-Oriented Programming*, pages 1-65, 1995.

[LO96] Li Li and A. Jefferson Offutt. Algorithmic Analysis of the Impact of Changes to Object-Oriented Software. *In Proceedings of the International Conference on Software Maintenance*, pages 171-184, Monterey, CA, November 1996. IEEE.

[RH94] G. Rothermel and M. J. Harrold. Selecting regression tests for object-oriented software. In *Proceedings of the International Conference on Software Maintenance*, pages 14-25, Victoria, B.C., Canada, September 1994.

[RH97] G. Rothermel and M. J. Harrold. A Safe, Efficient Regression Test Set Selection Technique. In *ACM Transactions on Software Engineering and Methodology*, 6(2):173-210, April 1997.

[WCK99] Ye Wu and Mei-Hwa Chen and Howard M. Kao. Regression Testing on Object-Oriented Programs. In *Proceedings of the 10th International Symposium on Software Reliability Engineering*, pages 270-279, Boca Raton, Florida, November 1999.

[WL92] L. J. White and H. K. N. Leung. A firewall concept for both Control-Flow and Data-Flow Regression Testing. In *Proceedings of the International Conference on Software Maintenance*, pages 262-270. IEEE, November 1992.

[YC80] Stephens S. Yau and James S. Collofello. Some Stability Measures for Software Maintenance. In *IEEE Transactions on Software Engineering*, 6(6):545-552, November 1980.