Abstract—Software needs to be maintained and changed to cope with new requirement, existing faults and change requests as software evolves. One particular issue in software maintenance is how to deal with a change proposal before change implementation? Changes to software often cause unexpected ripple effects. To avoid this and alleviate the risk of performing undesirable changes, some predictive measurement should be conducted and a change scheme of the change proposal should be presented. This research intends to provide a unified framework for change analysis, which includes dependencies extraction, change impact analysis, changeability assessment, etc. We expect that our change analysis framework will contribute directly to the improvement of the accuracy of these predictive measures before change implementation, and thus provide more accurate change analysis results for software maintainers, improve quality of software evolution and reduce the software maintenance effort and cost.

I. INTRODUCTION

Software maintenance has been recognized as the most difficult, costly and labor-intensive activity in software development life cycle [1]. Software change is a fundamental ingredient of software maintenance. Lehman and Belady proposed and refined five laws that characterize the dynamics of program evolution, in which the first law is “change is continual” [2]. Changes can be stemmed from new requirement, existing faults, change requests, etc. A change proposal will inevitably have some unpredicted and potential effects on software and may cause inconsistencies to other parts of the original software. If a change proposal involves significant ripple effects over the whole system, or even leads to undesirable effects, we may reject this change proposal, or consider another change schedule, or even redevelop a new system. Otherwise, we may accept the change proposal. Hence, one particular issue in software maintenance is to perform change analysis before change implementation to conduct software change management. Change analysis is a necessary phase to software maintenance, and is a sum of some activities before change implementation. These activities related to change analysis include:

- dependencies extraction, which tries to identify all possible dependencies (e.g., call, inheritance) between various program entities (e.g., class, methods and fields) [3], [4];
- change impact analysis, which is used to identify the potential ripple effects caused by changes made to software [5];
- changeability assessment, an evaluation of the ease to implement a change proposal [6]; and
- modification advice, which gives some constructive change proposals to reduce the software maintenance effort and cost.

Change analysis provides the guidance for change propagation, change effects traceability and regression testing during and/or after change implementation [5], [7], [8], [9], [10]. It is one of the important factors to support better quality of software evolution. Given the foregoing discussion, our research has an overall goal of providing a framework to support change analysis for a given change proposal before change implementation. Our research attempts to use Formal Concept Analysis (FCA) technique to support various change analysis activities in the framework. This framework aims to address some predictive measurement and provide a guided change scheme for the given change proposal, that is, estimating which parts of the original system may be affected by this change proposal, evaluating to what degree the change proposal affecting original system, and providing some maintenance guidance which can reduce the maintenance cost. All of the above will be validated through empirical studies on several open projects with respect to some other change analysis techniques.

II. RELATED WORK

In this section, we introduce some current work on change analysis from two aspects: 1) change impact analysis, and 2) changeability assessment.

A. Change Impact Analysis

Change impact analysis (CIA) starts with a set of changed elements in a software system, called the change set, and attempts to determine a possibly larger set of elements, called the impact set, which requires attention or maintenance effort due to these changes [5]. Current research in CIA varies from relying on static information [11], [12], [13], [14] to dynamic information [15], [16], [17]. Some also utilize both static and dynamic information in combination [9], [18]. Our research mainly focuses on static analysis of the program, hence, we introduce some related work on static CIA techniques.

Static CIA techniques take all possible behaviors and inputs into account, and they are often performed by analyzing the...
syntax and semantic dependence of the program [11], [13]. Concept analysis has been combined with program slicing technique to perform a fine-grain CIA in intraprocedural level [11]. In addition, some work tries to generate a ranked list of impact results, which facilitates the use of these impact results. Poshyvanyk et al. proposed a novel conceptual coupling measure to capture some dependencies which can not be captured by structural coupling measures [12]. The coupling measures are then used to produce a ranked list of classes based on different types of dependencies among classes. Recently, there is an increasing interest in Mining Software Repositories (MSR) technique to support CIA [19]. Some evolutionary dependencies between program entities that can not be distilled by traditional program analysis technique can be mined from these repositories. These CIA techniques often generate the impact results at a coarse class (file) level. Kagdi et al. utilized both single and multiple versions analysis for impact analysis [14]. They investigated two different combinations, i.e., disjunctive and conjunctive, to compute the impact set. Their research results showed that such combined methods provide improvements to the accuracy of impact set compared to single and multiple versions CIA, respectively.

B. Changeability Assessment

Changeability is an important software quality attribute to measure software maintainability [6]. Research on software changeability assessment includes proposing changeability predictors based on measurable factors that have a bearing on the software maintenance activity [20].

Currently, most research on changeability assessment considers some design property metrics as changeability indicators, e.g., cohesion, complexity, coupling, etc. [21]. In addition, Fluri proposed a changeability assessment model based on a taxonomy of different change types and a classification of these in terms of change significance levels for consecutive versions of software entities [22]. With this changeability model, each source code entity is classified as low, medium, or high. Then maintainers select appropriate modification strategy according to the changeability of the source code.

Changeability assessment above mainly focuses on the changeability of the evolving software without considering the software to absorb the individual change proposal. Chaumun et al. proposed a changeability assessment model which relies on computing the impact of classes changes [23]. They defined the change impact model for each class change type by analyzing the types of the dependencies between classes. Then changeability assessment is predicted based on the impact results of the classes changes. And usefulness of the changeability assessment result relies on the accuracy of the impact results. So such approach may often get wrong assessment results. And our approach tries to fill this gap.

III. GOALS AND APPROACHES

In this section, we present the overall goal and approach of our research.

A. Research Goal

Software change is a key operation for software evolution. And these changes are made to implement various change proposals. When given a change proposal, maintainers need to analyze and evaluate this change proposal to predict the effects of this change proposal, make a decision on this change proposal, and give some modification advice to decrease the change cost. All these activities should be carefully conducted before implementing the change proposal. All these activities are included in the change analysis framework. Our research tries to give an effective and comprehensive solution to these activities related to change analysis. The overall goal of our work is to provide a framework to support change analysis for a given change proposal before change implementation. Figure 1 gives an overview of our research. In this figure, we focus on the following activities and their corresponding goals related to change analysis:

1) **Dependencies extraction.** It firstly needs to construct an intermediate representation for original software. This representation should be easy and effective to build, and try to represent all possible dependencies that exist in original software. In addition, this representation can effectively facilitate the following change analysis activities.

2) **Change impact analysis.** Through change impact analysis, maintainers will have an initial overview of which parts in original software may be affected by the change proposal, and thus may cause inconsistency to the software. Accuracy is one of the key factors to evaluate the effectiveness of the impact analysis technique. That is, we want the impact results have fewer false-positives (i.e., the elements predicted by the impact analysis technique are not really impacted) and false-negatives (i.e., some of the real impacted elements are not identified by the impact analysis technique).

3) **Changeability assessment.** Changeability assessment is an evaluation of the ease of implementing the change...
proposal [6]. And the changeability result can help maintainers to answer whether a change proposal is accepted or to determine which change schedule is more suitable to employ. We hope that the changeability result on the change proposal can accurately reflect the real modification status during change implementation.

4) Modification scheme report. Results obtained from previous analysis and evaluation may be helpful in providing feedback to give a modification scheme report on the change proposal. The content in the report includes: which parts in original system is affected, whether we accept or reject the change proposal, which parts (sensitive part) in original system should not be modified to reduce the maintenance cost and effort, and which test cases should be added to the test suite to validate the consistence of these elements potentially affected by the change proposal. Through this report, some appropriate modification strategy may be proposed.

B. Research Approach

Our work uses Formal Concept Analysis (FCA) technique to support change analysis for original object oriented programs. FCA is a field of applying mathematics dealing with the study of the relation between entities and entity properties to infer a hierarchy of concepts [24]. For every binary relation between entities and its properties, a lattice can be constructed to provide a remarkable insight into the structure of the original relation [24]. In has been shown recently that FCA is an elegant and powerful code analysis technique for software maintenance [25].

We firstly construct the intermediate representations of original programs using concept lattice by providing different entities and entity properties. Based on the representation, dependencies between program elements are extracted, and the impact results are computed. With the impact results, A new metric indicating the changeability of this change proposal is defined. After all these activities, analysis of these results is conducted. And finally, report of these analysis and results are provided to maintainers. In addition, maintainers can use these results and analysis to suggest appropriate modification strategy.

IV. PRELIMINARY WORK

So far, we have investigated the dependencies extraction and change impact analysis techniques. The focus of our research is on various change analysis activities for the code-level change proposal. In the following sections, we discuss our efforts for each activity, as well as ongoing work if applicable.

A. Dependencies Extraction

Various activities in the change analysis framework are usually performed based on an intermediate representation, and one challenge to these techniques is an effective representation for original software. Various dependence graphs, e.g., system dependence graph (SDG [26]), are commonly used representations for program analysis. System dependence graph is one of the most commonly used representations to represent object oriented programs, and it analyzes program elements and their dependencies at a very fine level, which provides much information about the program elements and dependencies between program elements [26]. However, to construct such a representation needs much cost and it is possible to get wrong results. Here, we use Formal Concept Analysis (FCA) technique to construct a more simple representation which neglects the dependence types information between program elements. In our initial study, we construct a lattice to represent the original system. This representation is called lattice of class and method dependence (LoCMD) [27]. And dependence between classes, dependence between classes and methods, and dependence between methods can be uncovered based on concept analysis. In spite of the loss of an amount of information in our representation, our representation can meet the demands needed by various activities in the change analysis framework. Our experimental results show that dependencies between classes and methods are well covered with our representation to some extent [28]. On the other hand, the size of our representation is reasonable, and not larger compared to traditional dependence graph. Also, the variability of the number of lattice nodes does not increase as obvious as that of the number of the elements in original system. This reflects that our representation may scale to a large-scale program with much fewer nodes [28].

B. Change Impact Analysis

Change impact analysis is an important predictive measurement of the ripple effects induced by the proposed changes. Given the proposed changed entities, most current CIA techniques compute the ripple effects, which are composed of some potential impacted entities [11], [29], [13]. These generated results may confuse the maintainers with the problem that where could be the start point from these estimated impacted entities. Our research employs concept lattice to compute a ranked list of potential impact set from the proposed changed methods and/or classes. These potential impacted entities are ranked based on the hierarchical feature of concept lattice, and assigned with a metric, called impact factor (IF), which can be used to prioritize these methods to be inspected. The impact factor is proposed based on two assumptions: (1) Upward reachable methods are shared by an increasing number of unchanged classes, they are expected to be less and less affected by these changes; and (2) If upward methods are reachable from the nodes labeled by an increasing number of changed classes, these methods are more probably affected by these joint classes changes. Case studies on some real-world programs validate the reasonability of our two assumptions, and demonstrate that the predicted impacted methods with higher IF values also have higher probability to be affected by the changes [28].

With such impact results, maintainers can sequentially select the methods with IF values from high to low for check. When they select the impacted methods in the impact set with higher IF values, the impact set can reduce many false-positives; When they select the impacted methods in the impact set
with lower $IF$ values, the impact set can include some false-negatives that are really impacted. Thus our approach provides an eclectic way for impact analysis.

C. Changeability Assessment

Changeability assessment is an instructive metric to assess to what degree this change proposal affect other parts of the original system. Changes made to software will inevitably have some unpredicted and undesirable ripple-effects on other parts of the software. These ripples effects may sometimes confines to a small scope, or sometimes affect a large part of the system. The changeability result can help maintainers to answer whether a change proposal is accepted or to determine which change schedule is more suitable to employ. Currently, some researchers rely on some design complexity metrics or coupling measurement to check the changeability of the original system [30], [31], [20], [21]. These work considers few about the concrete change proposal to conduct the predication of the changeability of the system. But the maintenance effort and cost of different change proposals may be different. Therefore, the changeability assessment in our study is closely related to individual change proposal.

We propose an impactness metric to assess the changeability of the change proposal. Impactness is defined based on the impact set predicted by the CIA process, and measures to what degree a change proposal affect the original system. The smaller the impactness of the changes to the system is, the better for the changeability to the system. And this metric can guide maintainers to make a decision on the change proposal. Initial case studies show that: when given two or more change proposals, our changeability assessment model can accurately reflect which change proposal has fewer impact on original system. In addition, the experimental studies demonstrate that our approach can give maintainers more useful assessment results than traditional assessment approach on purely the impact results [32].

V. Expected Contributions

Given a change proposal, through change analysis, we hope to obtain: 1) the information about the dependencies in original system, 2) the potential ripple effects induced by the change proposal, 3) the changeability of the original system to absorb this change proposal, and 4) the appropriate modification strategy to reduce the software maintenance cost.

Based on the discussions above, our research is expected to make the following contributions:

- A novel metric that quantifies the changeability of the change proposal.
- An instructive modification strategy that effectively reduces the maintenance effort and cost to implement the change proposal.

ACKNOWLEDGMENT

The authors would like to thank in particular Paolo Tonella, Taweesup Apiwattanapong, Vaclav Rajlich, Radu Vanciu, Sai Zhang and Xiang Chen for their feedback and advice on our research. I would also like to acknowledge the colleagues in our lab for their endless support on the technical and empirical studies on this research.

This work is supported partially by National Natural Science Foundation of China under Grant No. No. 60973149, partially by the Open Funds of State Key Laboratory of Computer Science of Chinese Academy of Sciences under Grant No. SYSKF1110, partially by Doctoral Fund of Ministry of Education of China under Grant No. 20100092110022, and partially by the Scientific Research Foundation of Graduate School of Southeast University under Grant No. YBJJ1102.

REFERENCES


