COMBINING DIFFERENT CHANGE PREDICTION TECHNIQUES

Daniel Cabrero
Spanish Ministry of Internal Affairs, Traffic Division, C/ Josefa Valcárcel 44, Madrid, Spain

Javier Garzás
Kybele Consulting S. L. Madrid, Spain

Mario Piattini
Alarcos Research Group, University of Castilla-La Mancha, Ciudad Real, Spain

Keywords: Change prediction, object-oriented design, selectors of change.

Abstract: This work contributes to software change prediction research and practice in three ways. Firstly, it reviews and classifies the different types of techniques used to predict change. Secondly, it provides a framework for testing those techniques in different contexts and for doing so automatically. This framework is used to find the best combination of techniques for a specific project (or group of projects) scenario. In third place, it provides a new prediction technique based on what the expectation of change is, from the user's point of view. This new proposal is based on a gap found in the relevant research, during the course of a review of the relevant literature.

1 INTRODUCTION

The maintenance cost of a software system is directly related to how often is it expected to change in the future (Wiederhold, 2006), and thus to how expensive it is to modify that system. In this sense, in order to optimize maintenance costs, it is possible to find improvement possibilities for a given design, but only "*improvements related to artifacts having a bigger change expectancy will really save costs in later phases*" (Cabrero et al., 2007).

This is an example of what can be found right across the pertinent literature. Change prediction techniques can be used for a wide range of purposes, such as testing priorization, reengineering planning, or artifact tracing.

Given the importance of change prediction, many proposals to do with this have been presented in the last decades. Thus, in order to summarize all the available information, this paper reviews the related research work regarding change prediction and provides a **classification of change prediction techniques,** consisting of three categories. This classification depends on the source of information used to carry out the estimation.

During the literature review, we realised that there is a need for more research that addresses the accuracy of the different proposals. In other words, the literature did not give us an insight into which techniques are supposed to be the most efficient for each specific context of development.

This paper proposes a new technique that gathers together all the preceding research work on predicting change in object-oriented systems. This new proposal is called **Automatic Heterogeneous change prediction (AHCP)**. This name comes from the fact that we propose to evaluate, automatically, the behaviour of each change prediction approach on the precedent releases. The aim is also to use this information to apply the best combination of techniques to the next releases.

The literature review also highlighted a new gap in the research. We did not actually find any change prediction technique based on the user input. So in this paper we also propose a fresh approach that

Cabrero D., Garzás J. and Piattini M. (2008).
COMBINING DIFFERENT CHANGE PREDICTION TECHNIQUES.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - ISAS*, pages 57-63
DOI: 10.5220/0001675800570063
Copyright © SciTePress

identifies which design artifacts will change, using the estimated requirement changeability that is extracted from the different stakeholders.

The remainder of the paper is organized as follows. Section 2 describes the related research work on change prediction, including our new proposal based on user input. Section 3 points out what is lacking in the existing approaches. Section 4 describes in more detail the AHCP technique proposed in this paper. Finally, section 5 draws some conclusions and identifies future research work.

2 CHANGE PREDICTION APPROACHES

In the context of this work, we performed a review of the literature on change prediction techniques. As a result of the review, we identified several interesting contributions focusing on the probability of change. We classified those proposals into three different approaches:

- Review of historical information
- Analysis of static structure and properties
- Extraction of user information

As regards the last one of the above approaches, a new method called CORT (Change Oriented Requirement Tracing) is proposed in this paper.

2.1 Historical Information Review

Predicting the future is a hard task indeed. We can, however, study in detail what happened in the past, and expect a similar behaviour in the near future. In terms of change prediction, (Girba et al., 2004) concludes that there is an *"empirical observation that classes which changed the most in the recent past also suffer important changes in the near future"*.

This technique reviews which artifacts have changed throughout the system's history. A good option in the application of this technique is to divide the whole life of the project into releases. Table 1 shows an example of data extraction with three releases: R1, R2, and R3. The idea is to count the registered changes for each release in order to estimate the next release changes.

Table 1: Extraction per release of historical information data.

Artifact	Number of Changes per Release		
	R1	R2	R3
Art. 1	3	0	1
Art. 2	5	1	3
Art. 3	0	4	2

Table 1 presents information that can be used in several ways to predict changes. Release change ($Changes_{n+1}$) is calculated, then, as the average of the previous changes in the releases, that is, the sum of the changes of the previous releases (Σ_n $Changes_{n}$) divided by the number of releases (n), as set out in Equation 1.

$$Changes_{n+1} = (\Sigma_{i=(1\ldots n)} Changes_i) / n \qquad (1)$$

Table 2 shows the application of Equation 1 in the data presented in Table 1.

Table 2: Change prediction for the next release, using historical information.

Artifact	Estimated number of changes
Art. 1	4/3 = 1,25
Art. 2	9/3 = 3
Art. 3	6/3 = 2

To complete this approach, we can also take into account that recent changes may have more relative importance than old changes. (Girba et al., 2004) used a technique called *"Yesterday's Weather"*, which uses different metrics that assign a different importance to changes, depending on when they occur. For further information we would refer you to their work.

(Sharafat & Tahvildari, 2007) noted that this estimation would depend on the Time Between Releases (TBR). "*When the time between consecutive releases is very short, an overestimation can be observed; the opposite is true when this period is longer than average*". To achieve the prediction of change per unit of time, they proposed the use of a polinomial technique.

2.1.1 Advantages and Drawbacks

The main advantage of this technique is that it can be easily automated, but there are two pre-requisites for its use. First of all, we need to have all this information available in a Configuration Management Tool. Secondly, this tool must have already been used for a long enough period of time to receive a representative amount of change requests.

2.2 Static Structure/Properties Analysis

Some researchers have realized that structures and properties of an object-oriented design can identify change-prone objects. An example of change prediction based on static structure could be a typical "god object" scenario where an object sends messages to many other objects. This object is likely to have a high probability of change, because when a referenced object changes its interface, the change may be propagated to the first object.

Properties or code structures could also point to a change-prone object. The size and number of methods of an object can also be an indicator of its probability of change. The existence of code structures, such as big case statements or any other Bad Smells (Fowler, 1999) can also highlight a bigger probability of change.

Those techniques have proved to be useful in predicting change in each design artifact and component. (Tsantalis et al., 2005) proposed a new method based on *"Axes of Change"*, which assigned probability of change, taking into account the structure and dependencies of the static structure. Their work compared their proposal to many other change prediction methods based on static structure and properties. Among this set of techniques we can highlight Coupling Measures and Size Measures.

Coupling Measures have already been referenced throughout Impact Analysis literature for some time now. (Briand et al., 2002) provide a list of techniques that aim to identify dependencies among classes. Initially, this information was used to analyse the impact of different alternatives, but later on, a new utility of change prediction was discovered. Recent research suggests that if a class can be impacted by changes in other classes, this will raise its probability of change. Among Impact Analysis techniques, we can highlight (Chidamber et al., 1998), who proposed a suite of OO metrics, called C&K metrics: DIT (depth of inheritance tree), NOC (number of children), CBO (coupling be-tween objects), and RFC (response for a class), and two intra-class metrics, WMC (weighted methods per class), and LCOM (lack of cohesion in methods).

(Chen & Rajlich, 2001) also proposed a technique in the context of Impact Analysis, based on the construction of an *"Abstract System Dependence Graph"* (ASDG) representing dependencies between software components and domain concepts. They also proposed a tool called RIPPLES.

On the other hand, **Size Measures** are based on the fact that the bigger a class is, the less modularised its design is. This is the reason why some design heuristics recommend keeping classes simple and small. The Number of Methods per Class-NOO used in (Arisholm et al., 2004) to identify change prone classes, or the Class Size-CS, used in (Wilkie & Kitchenham, 2000) to investigate its relationship to the effort to implement changes, figure in this group of techniques.

In addition, (Sharafat & Tahvildari, 2007) proposed a combination of the "*Axes of Change*" technique with the historical information, using both probabilities together.

The above work is an interesting starting point for our change prediction proposal. A new approach that aims to gather together the previous research work on change prediction must take into account the analysis of structures and properties of design artifacts.

2.2.1 Advantages and Drawbacks

We can point out that one great advantage of those methods is that static structures and properties can be analysed automatically. Some tools already use metrics presented here to improve code and design. Unfortunately, those tools do not focus on the probability of change, and do not assign a different value to the improvement opportunities.

(Tsantalis et al., 2005) made tests on two "open source" Java projects, and identified the accuracy of all those different techniques. It is difficult to ensure the applicability of those accuracy rates to any software system, however. Different software, such as Real-Time system or Business Management Systems, may have different cycles of change. Thus, different methods should be used to predict their changes.

2.3 Extracting Change Tendencies from Stakeholders

In the preceding sections, we have presented a review of the different techniques used to predict changes in an object- oriented system. Those techniques are based on concepts such as historical information or static properties; that is, they are based on technical data. We could easily imagine a situation, however, where two projects with a similar history and static properties could have a different probability of change for non-technical reasons.

Sometimes, only final users and other stakeholders know about the possibility of some changes ocurring. We did not find any technique focusing on the change expectancy provided by users. In response to that lack, we propose a new technique called **CORT (Change Oriented Requirement Tracing)** which uses user input to estimate which design artifact requirement will change. Our technique is inspired by a case study presented by (Srikanth & Williams, 2005), which used a method called VBRT (Value Based Requirement Tracing). This sets a requirement priorization based on the risk and relative Value of each requirement.

In our case, we use a similar approach, but focused on changeability. We propose a technique that aims to ask the user about future possibility of change in the case of each approved requirement. We will later use tracing techniques to translate this Value from requirements to design artifacts.

2.3.1 CORT: Requirement Extraction

First of all, we need to identify all the different stakeholders of the project. Then these individuals are asked to assign "changeability" variables to each requirement and use case. They are asked explicitly about the possibility of several possible future scenarios. The identification of those variables will be based on work done on Software Product Lines.

"*Software Product Lines (SPL) engineering gathers the analysis, design and implementation of a family of systems in order to improve the reuse of the commonality among them*" (Clements & Northrop, 2001). A Product Line is thus a group of "*similar*" systems. In the SPL field, there is significant experience in models of variability. The experience in this field grouped differences, called "*discriminants*" of a family of products, into "mutually exclusive", "optional" and "multiple" differences (Keepence & Mannion, 1999). If we want to express this in change prediction terms, those types of changes would be **extension** (*multiple discriminant*), **change** (*option discriminant*) and **suppression** (*mutually exclusive discriminant*).

As far as our case is concerned, stakeholders are asked to estimate the "**probability of change**", the "**probability of extension**" and the "**probability of suppression**". This will generate three matrices of requirement/stakeholders with an "estimated variability" of each requirement from the point of view of each stakeholder. Table 3 shows an example of one of those matrices, focusing on "probability of extension".

Table 3: Estimated Extension matrix.

	User 1	**User 2**	**User 3**
Req. 1	28%	71%	0%
Req. 2	28%	100%	14%
Req. 3	0%	14%	0%

The next step is to calculate the average probability (or adjusted average, which assigns weights to different users if each stakeholder is not equally important). For the sake of simplicity, in our case we will consider all users to have the same relative importance and we will not use weights. Each column of Table 4 represents a type of change extracted from a matrix such as the one presented in Table 3.

Table 4: Estimated Probability of change for each requirement.

	Change	**Extension**	**Suppression**
Req. 1	33%	12%	15%
Req. 2	47%	6%	80%
Req. 3	5%	5%	1%

As denoted by (Keepence & Mannion, 1999), different types of change can be solved using different types of design patterns. This is the main reason why it could be a good idea to specify what sort of change is to be expected (change, extension and suppression).This facility is not available from other change prediction techniques.

2.3.2 CORT: Requirement Tracing

"*Requirements tracing is the ability to follow the life of a requirement in a forward and backward direction"* (Gotel & Finkelstein, 1994). For our proposal, we are only interested in requirement tracing techniques that establish relationships between requirements and design artifacts in a forward direction. This relationship will allow us to analyse which objects will change if a given requirement changes, and to translate that "probability of change" from requirements to objects. For example, if a requirement 1 has a 33% of probability of change and this change will affect both the A and B objects, we could conclude that objects A and B have a 33% of probability of change.

Several techniques have been proposed over the last years. An interesting summary of tracing techniques is provided by (Cleland-Huang et al., 2004). A further detailed analysis of each tracing technique is beyond the scope of this document. A simple approach is recommended in this case, however, because at this stage of the process we already know the relative importance of each requirement, and we can trace only requirements that have a higher probability of change.

2.3.3 Advantages and Drawbacks

One of the main advantages of this approach is that many software requirement tools, which already have variables associated with requirements such as importance or frequency, can use it with ease. In addition, as we extract information on change directly from stakeholders, our technique is the only one that allows an identification of the specific kind of change that may be expected (Change, Extension or Suppression). On the other hand, we need access to the software requirement specification and it is essential to have direct contact with stakeholders, which is not always available. Another important drawback is that processes involving stakeholders are expensive in terms of both time and resources.

3 LACKS IN EXISTING TECHNIQUES

A number of tests for change prediction accuracy have been done in specific contexts. However, we found a gap in the research as regards when to apply each of those techniques. In other words, some techniques are more valuable than others, depending on the specific kind of project.

For example, reviewing historical data and user input techniques can be very useful in some Business Management Systems, where final users could constantly add functionality incrementally, or redesign some parts of the application user interface. On the other hand, structure analysis techniques can achieve a better accuracy in other contexts, such as a real-time system. This raises the following issues:

- When should we use each technique? Which one adds more value?
- Can we use several techniques at the same time?
- If so, how much accuracy does each technique provide?

(Chaumun et al., 1999) claims that *"In summary, most results on the influence of design on changeability come out of small systems, and the change impact models we found in the literature are incomplete or not systematic"*.

We believe that this lack of knowledge, concerning when and how is it efficient to use all those different approaches, makes it difficult for the software industry to use change prediction techniques. We also believe that further research in this field would help to reduce maintenance costs, and facilitate the daily work of the industry.

4 AHCP (AUTOMATIC HETEROGENEOUS CHANGE PREDICTION)

In the sections above, we have given an overview of the available research work that is related to change prediction. There exist a lot of metrics and techniques that could help when trying to accomplish this task. In fact, a given technique could successfully predict changes in a given scenario and yet it might not achieve this accuracy in other context. The problem then, as stated before, is that it is difficult to know what the accuracy of each approach will be.

This new approach sets out to identify which techniques of change prediction predict the future better than others, for a given scenario. In addition, it uses this information to select the most appropiate techniques for making new predictions. To achieve this, we will use the concept of “*change selector*” to identify (or “*select*”) change prone classes.

4.1 Change-Prone Selectors

When talking about change prediction, each metric or technique that aims to identify a change-prone class can be modelled as a selector. For example, the Class Size metric can be modelled as “Classes larger than 5Kb of source code (without comments)”. We call this a “selector”, because it “selects” classes complying with this size, and marks them as change-prone classes.

Table 5 summarizes an initial example of a catalogue of selectors, where some previous research work has been modelled. Each selector can be configured through variables. For example, CS selector can be configured using the “Size in KB” variable. When this variable is configured to a very high value, only really big classes will be selected.

Table 5: Example of Selector Catalogue.

Prev. Work	**Change Selector**	**Variables**
Historical	Number of changes per release > M	M = Num of Changes
CORT	The probability obtained < N	N = Thrshld.
Axes of Change	The probability obtained < N	N = Thrshld.
CS (Class Size)	Classes larger than 5Kb of source code (without comments)	N = Size in KB
CBO(Coupling Measure)	A class must be coupled with at least N other classes	N = Num. of classes
NOO (Num of Methods)	Every class must contain at least N methods	N = Num. of methods
Refactorings	Switch or if sentence with more than N statements	N = Num. of statem.

This example of a list is only an initial step. Selectors will be added or deleted through experimentation, as described in the next section.

4.2 Assigning Value to Selectors

Even if software architects know about techniques that help to estimate the probability of change, they must deal with the problem of selecting the appropriate technique or set of techniques for its specific situation. In fact, right up to the present time, no work addressing this issue has been published.

The advantage of change prediction techniques is that we can estimate the accuracy of each type of technique using historical information, and use selector's variables and weights to find out the best combination of techniques for each kind of project.

Let us imagine an Enterprise Management project. We can use our technique to perform simulations of estimation of change at the end of the third release, in order to estimate changes in the fourth release. Our approach proposes to compare the results of each technique with the changes that actually happened. Depending on the *Overall Accuracy-OA* and *Sensitivity-S (Percentage of correct change classifications),* we will set up the **weight** variable for each technique. The bigger the OA and S are, the higher the weight will be.

False Positive Ratio-FPR (Percentage of incorrect classifications of changes that did not occur) will help to set the Threshold and other variables, such as **number of changes**, number of methods and so on. In this case, the bigger the FPR is, the higher the Threshold and number of changes must be. OA, S and FPR ratios were extracted from (Tsantalis et al., 2005) accuracy tests.

We can thus make the comparison using several possibilities, to find the best combination of techniques for this specific project. Table 6 shows an example of this data.

Table 6: Weighting and configuring Selectors.

Selector	**Threshold**	**Weight**
Historical	Changes = 3	0,5
CORT	Threshold = 0,6	0,3
Axes of Change	Threshold = 0,3	0,2

Another interesting aspect is the automation of this technique. This would allow us to repeat this process for a set of projects in order to segment them into different groups, depending on which techniques predict the Probability of Change with greater accuracy. This will provide new results that should help to guide future research.

In the example described in Table 6, accuracy values have automatically discarded the rest of the techniques. Note that each technique is selected or discarded automatically. In this way, whether or not to apply a technique will be based on empirical data instead of on personal opinions.

4.3 Why is this Information Useful?

This information obtained by comparing expected results with historical data can be useful in order to:

- Estimate costs in future releases of a project.
- Choose between the different kinds of estimation techniques depending on the characteristics of our specific project.
- Invest more effort in the testing and tracing of where the Probability of Change is higher.
- Design applications to make them easier to maintain, introducing patterns where the application is expected to change.

Change prediction techniques can also be useful in focusing efforts on change-prone artifacts. This is possible in several ways. For example, (Girba et al., 2004) used it to guide the reverse engineering process of large systems. (Kung et al., 1995) were also interested in change prediction for regression-testing purposes.

5 CONCLUSIONS AND FUTURE WORK

The problem of software change prediction is not new. In the last few years many research papers on this issue have been presented. However, our experiences in several software factories reveal that industrial practice doesn't reflect this research effort.

We believe that the misuse of change prediction techniques is due to the fact that developers don't know which techniques are available. Apart from that, they don't know which techniques are supposed to be the most efficient ones for their specific context of development.

This work has presented a review and classification of the different types of change prediction techniques. It also provides a framework for testing those techniques automatically, in different contexts.

The result of this work is directly applicable to different lines of research. For example, the relative importance of a test or a design decision will be bigger if it focuses on a change-prone component. In other words, the Return of Investment will be more profitable if we focus our efforts correctly, using change prediction techniques.

In future work, we plan to build an agent that both automates and assists in the different steps proposed in this paper. More specifically, in the context of the Traffic Division of the Spanish Ministry of Internal Affairs, we plan to apply those techniques to guide software improvements. In this way, we plan choose which part of applications should be refactored (change-prone classes) in order to improve their maintainability.

We expect that increasing the maintainability of change-prone component, the cost of maintenance will decrease, and a higher Return of Investment will be provided face to changes.

ACKNOWLEDGEMENTS

This research is partially supported by the ESFINGE project of the General Research Council (Dirección General de Investigación) of the Spanish Ministry of Education and Science and FEDER (TIC 2003-02737-C02-02)

REFERENCES

Arisholm, E., Briand, L. C. & Føyen, A. (2004) Dynamic Coupling Measurement for Object-Oriented Software. *IEEE Transactions on Software Engineering*, 30**,** 491-506.

Briand, L. C., Wüst, J. & Lounis, H. (2002) Using Coupling Measurement for Impact Analysis in Object-Oriented Systems. *Science of Computer Prtogramming*, 45**,** 155-174.

Cabrero, D., Garzás, J. & Piattini, M. (2007) Maintenance Cost of a Software Design. A Value-Based Approach. In *9th International Conference on Enterprise Information Systems (ICEIS)*, Funchal, Madeira. Portugal,

Cleland-Huang, J., Zemont, G. & Lukasik, W. (2004) A Heterogeneous Solution for Improving the Return on Investment of Requirements Traceability. In *Requirements Engineering Conference, 12th IEEE International (RE'04)*, IEEE Computer Society

Clements, P. & Northrop, L. (2001) *Software Product Lines: Practices and Patterns*, Addison-Wesley.

Chaumun, M. A., Kabaili, H., Keller, R. K. & Lustman, F. (1999) A Change Impact Model for Changeability Assessment in Object-Oriented Software Systems In *European Conference on Software Maintenance and Reengineering*, Washington, DC, USA IEEE Computer Society.

Chen, K. & Rajlich, V. (2001) RIPPLES: tool for change in legacy software. In *International Conference on Software Maintenance*, Florence, Italy, IEEE Computer Society.

Chidamber, S. R., Darcy, D. P. & Kemerer, C. F. (1998) Managerial Use of Metrics for Object-Oriented Software. *IEEE Transactions on Software Engineering*, 24**,** 629-639.

Fowler, M. (1999) *Refactoring: Improving the Design of Existing Code*, Menlo Park, California, Addison Wesley.

Girba, T., Ducasse, S. & Lanza, M. (2004) Yesterday's Weather: Guiding Early Reverse Engineering Efforts by Summarizing the Evolution of Changes. In *20th IEEE International Conference on Software Maintenance* Washington, DC, USA IEEE Computer Society.

Gotel, O. C. Z. & Finkelstein, A. C. W. (1994) An analysis of the requirements traceability problem. In *1st International Conference on Requirements Engineering*, Colorado Springs, CO, USA, IEEE Computer Society.

Keepence, B. & Mannion, M. (1999) Using patterns to model variability in product families. *IEEE Software*, 16**,** 102-108.

Kung, D., Gao, J., Hsia, P., Wen, F. & Toyoshima, Y. (1995) Class firewall, test order, and regression testing of object-oriented programs. *Object Oriented Programming*, 8**,** 51-65.

Sharafat, A. R. & Tahvildari, L. (2007) A Probabilistic Approach to Predict Changes in Object-Oriented Software Systems. In *International Conference in Software Maintenance and Reengineering*, Amsterdam, IEEE Computer Society.

Srikanth, H. & Williams, L. (2005) On the economics of requirements-based test case prioritization. In *7th international workshop on Economics-driven software engineering research* St. Louis, Missouri ACM Press

Tsantalis, N., Chantzigeorgiou, A. & Stephanides, G. (2005) Predicting the Probability of Change in Object-Oriented Systems. *IEEE Transactions on Software Engineering*, 31**,** 601-614.

Wiederhold, G. (2006) What is your Software Worth? *Communications of the ACM*, 49**,** 65-75.

Wilkie, F. G. & Kitchenham, B. A. (2000) Coupling Measures and Change Ripples in C++ Application Software. *Systems and Software*, 52**,** 157-164.