SOFTWARE MAINTENANCE: RESEARCH AND PRACTICE, VOL. 6, 35–52 (1994)

An Early Impact Analysis Technique for Software Maintenance

RICHARD J. TURVER AND MALCOLM MUNRO

Centre for Software Maintenance, School of Engineering and Computer Science, University of Durham, Science Laboratories, South Road, Durham.

SUMMARY

The accurate estimation of the resources required to implement a change in software is a difficult task. A method for doing this should include the analysis of the impact of the change on the existing system. A number of techniques for analysing the impact of a change on the source code have been described in the literature. While these techniques provide a good example of how to apply ripple effect analysis to source code, a weakness in these approaches is that they can be difficult to apply in the risk assessment phase of a project. This is because the source code is often not very well understood at this phase, and change proposals are written at a much higher level of abstraction than the code. It is therefore often the case that in practice subjective impact analysis methods are used for risk assessment and project investment appraisal. The underestimated resources for dealing with the ripple effects of a change can result in project schedules becoming so tight that only the minimal quality is achieved. This paper surveys existing ripple analysis techniques and then presents a new technique for the early detection of ripple effects based on a simple graph-theoretic model of documentation and the themes within the documentation.

The objective is to investigate the basis of a technique for analysing and measuring the impact of a change on the entire system that includes not only the source code but the specification and design documentation of a system, and an early phase in the maintenance process.

KEY WORDS Documentation Impact analysis Ripple effect Software maintenance

1. INTRODUCTION

In developing software a number of competing quality factors such as reliability, portability, efficiency, understandability, reusability and maintainability have to be taken into account. The increasing cost of developing software has meant that the life of existing systems has been extended, thus putting a greater emphasis on software maintenance. Over the total life of software the software maintenance effort has been estimated to be frequently more than 50% of the life cycle costs (Lientz and Swanson, 1980). This maintenance cost shows no sign of declining (Nosek and Prashant, 1990).

One of the reasons for the high cost of maintenance is the way in which proposals for change are processed. If change proposals are serviced in the order in which they are made then this may involve extra cost because some changes may overlap with the others, resulting in duplication of work and increased complexity. Costs can be reduced by scheduling change proposals and batching similar proposals together. In order to achieve

CCC 1040–550X/94/010035–18

Received 4 November 1991
Revised 19 August 1993

this, detailed analysis of the system is required to determine the effect of each change on other programs and documentation. A simple model of the software maintenance process from the release planning perspective is (Arthur, 1988):

(1) rank change proposals into a priority order;
(2) select highest priority changes that can be made with available resources;
(3) secure agreement on the content and timing of system releases;
(4) obtain approval to implement the changes;
(5) schedule work into groups to maximize productivity;
(6) prepare release information.

The objectives of release planning are to establish a schedule of system releases and to determine the contents of each release. It is important to rank the changes in some priority order so that, for example, those changes which most improve the profitability of the business may be given a higher priority. The ranking of changes is an important activity which has an impact on the profitability of a system. However, in practice subjective ranking methods are the norm (Foster, 1989). The software maintenance process can be optimized by the use of release-engineering techniques. Examples of such optimizations are a reduction in both project costs and introduced defects by scheduling and batching of releases, based on change impact or ripple effect information. The software maintenance process can only be optimized if precise and unambiguous information is available about the potential ripple effects of a change on an existing system. Measurement techniques can be used in release planning in order to assign numbers to attributes of change proposals in such a way as to describe them according to clearly defined rules. This assignment of numbers to change proposal attributes allows comparisons to be made between the change proposals. The earlier the information is produced in a project the greater will be the size of the process optimization.

This paper investigates ripple effect analysis on source code and documentation of an existing software system. The motivation behind this work is to improve the maintainability of software systems, optimize the release planning activity and thus reduce the maintenance costs. Reduction in costs can be achieved by reducing the time between a proposed change, its implementation and its delivery, whilst at the same maintaining quality.

2. PROCESSING CHANGE PROPOSALS

A software system should not be considered only in terms of its source code. It consists of many other related items such as specification and design documentation. Often a change has system wide ramifications which are not obvious. When considering a change to the source code of a system it is important to assess the impact of that change, not only on the source code but on the other elements of the system. If this process is not carried out then the different elements of the system will become out of step and will lead to a system that is much more difficult to maintain. For example the design documentation may not match the implementation in the source code, thus making the system more difficult to understand.

In carrying out software maintenance it is important to have a model of the process for dealing with change proposals and scheduling releases. Most maintenance models are request-driven showing how a proposal is dealt with and how it flows through the model

(Liu, 1976; Swanson, 1976; Sharpley, 1977; Yau *et al.*, 1978; Parikh, 1982; Martin and McClure, 1983; Patkau, 1983; Osborne, 1987).

A simple idealised model of the change proposal evaluation process can be:

(1) inspect the change proposal;
(2) assess how desirable the change is to the business process;
(3) assess whether the change is a legal requirement;
(4) produce a business feasibility study;
(5) translate the business feasibility study into system terminology;
(6) produce a technical proposal feasibility study;
(7) obtain user agreement on the translated technical proposal feasibility study;
(8) trace the impact on other system components;
(9) assess the risk of introducing defects along with the change and the cost of removing these defects based on past experience;
(10) assess the financial impact on the organization if the subsystem being maintained fails at any time when the system goes live;
(11) compare results produced from the preceding two steps and determine the risk involved in financial terms;
(12) develop an initial resource estimate based on the manpower cost of processing this change proposal;
(13) subtract the initial resource estimate from the expected cost benefit;
(14) assess whether the maintenance investment opportunity should be accepted or not;
(15) report to the user the scope of the change, resources required and the expected return on investment which will accrue to the organization as a result of making the change and any risk involved.

The return on investment can be calculated by subtracting the cost of processing the change proposal from the expected cost benefit. It is therefore imperative to derive an accurate assessment of the change impact from the code and documentation affected and to also produce this estimate early in the maintenance process. If the impact of a change is not detected early then the project schedule may become so tight that only minimal quality will be possible. An important decision point has been identified within the software maintenance process detailed above at which impact analysis is needed for objective comparison and for project resource estimation. Existing techniques for analysing the ripple effect of a change concentrate on the impact on the source code. If the problems described above are to be overcome then this analysis must be extended to include changes to documentation.

3. IMPACT ANALYSIS

The ripple effect of a change to the source code of a software system is defined as the consequential effects on other parts of the system resulting from that change. These effects can be classified into a number of categories such as logical effects, performance effects, or understanding effects. For example a logical ripple effect may be caused by a change in the definition of a data item, which may then cause inconsistencies in the definition and use of other data items.

Impact analysis is the assessment of the effect of a change, to the source code of a module, on the other modules of the system. It determines the scope of a change and provides a measure of its complexity. This type of analysis allows the maintenance managers and programmers to assess the consequences of a particular change to the source code. It can be used as a measure of the cost of a change. The more the change causes other changes to be made ('ripples') then, in general, the higher the cost. Carrying out this analysis before a change is made allows an assessment of the cost of the change and allows management to make a tradeoff between alternative changes.

However, the usual use of impact analysis is to determine the ripple effect of a change after it has been made. None of the existing impact analysis systems (Yau *et al.*, 1978; Haney, 1972; Wild *et al.*, 1991) use analysis to control the maintenance process (Pfleeger and Shawn, 1990). This suggests that the level of information provided by these systems is not sufficiently extensive to allow resource estimation and reasoning about the proposed change.

The most basic ripple effect analysis can be performed using a text editor. However, data flow analysis tools, program slicers and program database tools can provide more detailed information. It is even better to use dedicated tools and techniques for ripple effect analysis. Examples of such ripple effect analysis tools and techniques are described below.

A technique developed by Yau, Collofello, and MacGregor (Yau *et al.*, 1978) analyses the ripple effect from the functional and performance point of view. One aspect of this technique involves identification of the program areas which require additional maintenance activity to ensure their consistency with the initial change. The analysis of performance ripple effect requires the identification of modules whose performance may change as a consequence of software modification. Quantitative estimates for effort involved in making a change are not discussed in this paper.

An approach to impact analysis based on the extended use of traceability to the tracking of requirements to make predictions about the effects of changes on requirements were made using ALCIA (RADC, 1986). The granularity of the unit of analysis is document information content.

A ripple effect analysis technique based on both semantic and syntactic information is presented by Collofello and Vennergrund (1987). In this approach attempts are made to reason about the impact of a change. Semantic descriptions are linked to syntactic dependencies. If a procedure is changed and the semantic condition linked to it is affected, then all other components which depend on that condition are possibly impacted.

Pfleeger and Bohner (1990) recognize impact analysis as a primary activity in software maintenance and present a framework for software metrics which could be used as a basis for measuring stability of the whole system including documentation. The framework is based on a graph, called the traceability graph, which shows the interconnections among source code, test cases, design documents and requirements. This framework provides a good example of the inclusion of software work products as part of the system, although it is anticipated that the level of detail on a diagram is insufficient to make detailed stability measurements. The theoretical basis for extracting impact measures within the meaning of measurement theory has also not been addressed in this work.

A decision view of documentation is presented by Wild, Maly and Liu (Wild *et al.*, 1991). The view structures the document base according to the decisions made during the problem solving process and provides a new way of viewing the software document base. The decision dependency graph links requirements to the documentation which can be used to assess the impact of a proposed change. This paradigm helps identify the parts of the documentation affected by a change. However, the assessment of effort is left to human judgement.

A large amount of information is produced by some techniques such as logical ripple effect techniques (Yau *et al.*, 1978). This information can often be the worst-case impact. The source-code-based techniques are difficult to apply at the production engineering phase because the level of abstraction of the system knowledge at this phase is too high-level. Semantic ripple effect techniques are useful to reason about changes, although the power of the information used to reason is only as useful as the information entered. These semantic conditions can also be linked to source code. However, large systems will make the presentation of all the information a practical problem unless sophisticated windowing systems are used. Impact analysis based on identifying a decision closure provides a useful approach for maintaining documentation. The location of these decisions must be recorded in the development phase. To summarize, the early detection of ripple propagation is difficult during the production engineering phase of software maintenance using the existing techniques unless the semantic information or decision structure is already known.

The main criticisms that can be levelled against existing impact analysis systems is that they do not provide sufficient detail for obtaining accurate estimates of the effect of a change. This is because most of the emphasis is on the impact on the source code. Another weakness is that existing techniques are difficult to apply at the risk assessment phase of a project because not all of the information is known at that time. Once a maintenance project has been committed, then it is of little use for maintenance team to proceed to the detailed design phase or to the analysis of detailed technical design documentation only to find that the change is considerably more involved than intuitive judgement led them to believe.

The software maintenance projects have a poor reputation for coping with the ripple effect. The impact analysis phase of risk assessment is either conducted subjectively or by simply making contingency plans and as a result, projects often fail to meet deadlines and cost targets. As maintenance projects become larger and more structurally complicated and as software systems evolve, so the need for impact analysis techniques will be increased particularly when projects have large capital expenditure. The problems associated with existing impact analysis techniques will therefore become more important and will merit further investigation as software systems increase in size and complexity.

4. STABILITY ANALYSIS

Stability analysis differs from impact analysis in that it considers the sum of the potential ripple effects rather than a particular ripple effect caused by a change. Stability is considered as one of the major attributes of maintainability because the understanding and modification of a program, and the calculation of possible ripple effects constitutes a major proportion of the software maintenance effort. Program stability has been defined by Yau and Collofello (1980) as: 'the resistance of a program to the amplification of changes in the program'. A number of stability measures have been proposed, all concentrating on the stability of the program source code by directly scanning the source code or by predicting program source code stability from higher-level design documentation. None of the existing techniques address the stabilization of documentation itself. This is surprising as documentation such as data flow diagrams, test, and algebraic specifications must also be maintained with the source code. Documentation such as this is also prone to a rippling effect during maintenance.

Haney (1972) describes a technique which models the stabilization of a large system as a function of internal structure. This technique includes a matrix formula for modelling the rippling effect of changes in a system. The matrix records the probability that a change in one module will necessitate a change in any other module in the system. For example a release information sheet can be used to revise the probabilities of module ripple propagation by recording the changes made to modules and to the other modules affected. It is intuitive that the more this technique is used the more predictive value the model will have. The matrix formula based on these probabilities can be used for explaining why the process of changing a system is more involved than intuition leads maintainers to believe. The technique can be used to estimate the number of changes required as a result of changing one particular module. No data is presented to show that the estimated total changes is related to cost or actual changes required. A weakness in this technique is the assumption that all modifications to a module have the same ripple effect and could make the estimation of change resources inaccurate.

A program stability measure was also developed by Song (1977). A technique is presented to quantify the quality of a program in terms of its information structure which is based on the sharing of information between components of a program. The technique is based on a connectivity matrix and random Markovian processes and also makes a similar assumption to the technique described by Haney, in which all modules share the same information content and also sharing of information between modules is equal. This technique is more complicated than that of Haney and has also not been validated.

A measure of the logical stability of software systems consisting of modules was proposed by Yau and Collofello (1980). The logical stability of a module is a measure of the resistance to the impact of such a modification on other modules in the program in terms of logical considerations. In this work logical stability measures are developed for a program and the modules of which the program is composed. There are two parts of logical ripple effect considered by Yau and Collofello in their stability measure. One part concerns intra-module change propagation and the other concerns inter-module change propagation. The metric is based partly on a complexity metric, and partly on probabilities that a modification to a module will affect a particular variable. This stability measure is useful; however, a weakness is that it is very time-consuming to apply the technique to large programs since all the source code must be statically analysed.

Yau and Chang (1984) developed a new stability analysis technique which is similar to the technique described above but easier to apply to large programs. It is not a code-based technique and it is intended to be applied to the design phase of software project. The technique is more efficient because the source code is not statically analysed. The complete information regarding variable definition and usage is not a requirement of this technique. Instead dependencies between global variables and modules are extracted from design documents. This approach uses whatever design information is available.

Yau and Collofello (1985) also developed design stability measures that are intended to be used for assessing the quality of program designs based on design documentation in the design phase of software development. They are based on the counting of assumptions made about module interfaces and shared global data structures. This information is taken from program design documentation.

It is evident that the existing techniques for stability analysis can only be applied to source code and source code detailed documentation. It would be useful to know just how interconnected is the higher level technical documentation, since it is the higher level documentation which is assimilated by maintenance staff first in the processing of a change proposal.

5. DOCUMENTATION STABILITY

The previous sections have described how maintenance process costs can be reduced by batching and scheduling projects in an optimum order. This optimization can be achieved by analysing the impact of a proposed change on an existing system and servicing change proposals which have least impact and have the largest cost benefit to the organization. The weaknesses of existing techniques for impact and stability analysis are:

- the measurement of documentation is not included in the analysis;
- it is difficult to use the existing techniques at an early phase in the maintenance process.

Existing ripple effect techniques extended to include the analysis of documentation would facilitate more accurate estimates for maintenance work and earlier estimates of the potential ripple effect of a change. The remainder of this paper investigates a technique for analysing the impact of a change on the entire system. It includes not only the source code but the specification and design documentation.

As a system evolves the source code changes but the documentation usually either remains the same or lags behind. Even when an effort is made to update to documentation subtle points are often overlooked in the rush to produce a new release. The accumulation of errors and omissions in documentation produces documentation that cannot be relied upon thus making the system harder to understand and thus maintain. Only by systematic analysis of the resistance to the potential ripple effect of document entities on other document entities can specification and design documentation be effectively maintained. The factors which affect the stability of documentation are the ways in which:

- the source code modules are connected;
- the source code modules are connected to the documentation;
- the documentation is connected together;
- the information is shared between source code modules;
- the information is shared between documentation.

The structure, form and organization of documentation is defined by the way in which its document entities are put together and by the mutual relations between the documents. The mutual relations represent document coupling.

The stability analysis techniques described above do not have a natural extension into documentation because of the difficulty in analysing the structure and content of documents and the lack of a suitable formal model of documentation from which a theoretical basis for analysis can be developed. The sheer diversity of the constructs and facilities which form documentation poses a challenge to formal models of the structure of documentation.

It is possible, however, to analyse documents provided that the following observations about documentation are taken into consideration:

- documentation has some sort of topology which can be subjected to analysis;
- documentation consists of one or more documents;
- documentation can be hierarchical in structure;
- documents can contain hierarchically structured sections;
- documents are logically connected through the mutual information they share;
- documents can contain text and/or graphics;
- sections at the bottom level can be decomposed into segments;
- the irreducible segments can represent the smallest primitive component of a document;
- documents can contain inter- and intra-document dependencies;
- some documents may be missing or incomplete;
- some sections of documents may be missing or incomplete;
- document segments may contain subjects or topics which will be called themes.

From these observations the hierarchical structure of documentation can be modelled for subsequent analysis. The structure of documentation can be modelled by considering its hierarchical structure and by observing a pattern of co-occurrences of themes in document segments. This will provide a model of the thematic structure of documentation. The provision of such a model of the interconnection of documentation will enable quantitative statements to be made about the content. Such statements are necessary in order to facilitate:

- the description of the structure of the content of documentation precisely and concisely;
- the precise description of the ripple effect of a change on documentation.

Using this structure of documentation to facilitate impact analysis makes this approach different from other work in the field. The uniqueness can be summarized as:

- the technique is based on the notion of the structure of documentation;
- the ripple effects are analysed by determining the themes of documentation entities;
- the ripple effect technique can be applied in the investment appraisal phase and production engineering phase of software maintenance for batching and scheduling projects, rather than in the implementation phase of software maintenance;
- the technique consolidates the existing ideas of calculating the logical worst-case impact (Yau *et al.*, 1978) with the idea of calculating the stochastic or most likely impact based on past experience (Haney, 1972) into a hybrid ripple effect analysis technique which can be applied to higher levels of abstraction than module connection information and source code;
- consideration of the problem from first principles has led to the development of a new kind of interconnection graph called a ripple propagation graph which forms the theoretical basis of this approach.

The following section investigates the utility of combining hierarchical documentation structure with a categorical and thematic structure for the purpose of analysing the impact of a change on one or more document sections. The analysis of the thematic impact on documentation will indicate the source code impacted by a change and also the risk of introducing defects at a much earlier phase in the maintenance process than existing techniques allow.

6. EARLY DETECTION OF RIPPLE PROPAGATION

The observations about documentation described above were very general in nature. In order to devise and describe algorithms for impact analysis these general observations have to be transformed into a logical model of documentation. This logical model will have to reflect the specific hierarchy of documentation; such a model is:

(1) Documentation Libraries
(2) Volume Entities
(3) Chapter Entities
(4) Section Entities
(5) Subsections Entities
(6) Segment Entities

In this model the relationship between each level of the hierarchy is that of '*consists-of*'. So that, for example, Chapter Entities *consists-of* one or more Section Entities. This '*consists-of*' relationship can represent the hierarchy of any documentation. Any entity in the hierarchy above the segment entity is called composite entity.

The desirable approach for impact analysis is to make the technique rigorous and applicable to any development or maintenance method and the abstractions the method manipulates. Any measurements made should be measurements of mathematical objects and not measurements of features of a particular documentation method.

6.1 Ripple propagation graphs

The documentation hierarchical structure described above can be modelled with an acyclic directed graph which can be labelled. A graph consists of a set of elements called vertices, and a set of edges connecting the vertices. An acyclic graph does not contain a sequence of vertices which connect back to the start vertex. A directed graph has edges with an arrow on the edge indicating the direction of a flow or dependency between two vertices. When information is associated with the vertices and edges of a graph the graph, is called a labelled graph.

In the model, the graph vertices represent documentation entities and the graph edges represent a relation of the vertices. The relation is *consists-of* which represents documentation decomposition. The graph model consist of layers of hierarchical interconnection graphs (HIGs). For example vertex V_2 is a root vertex of the graph $\{(V_2,C_1), (V_2,C_2)\}$ (see Figure 1).

Further information can be added to the Segment Entities such as theme or themes in that segment. This can be defined with additional vertices called theme vertices. The relationship between a Segment Entity and a theme vertex is that of *has-theme*. Dependenc-

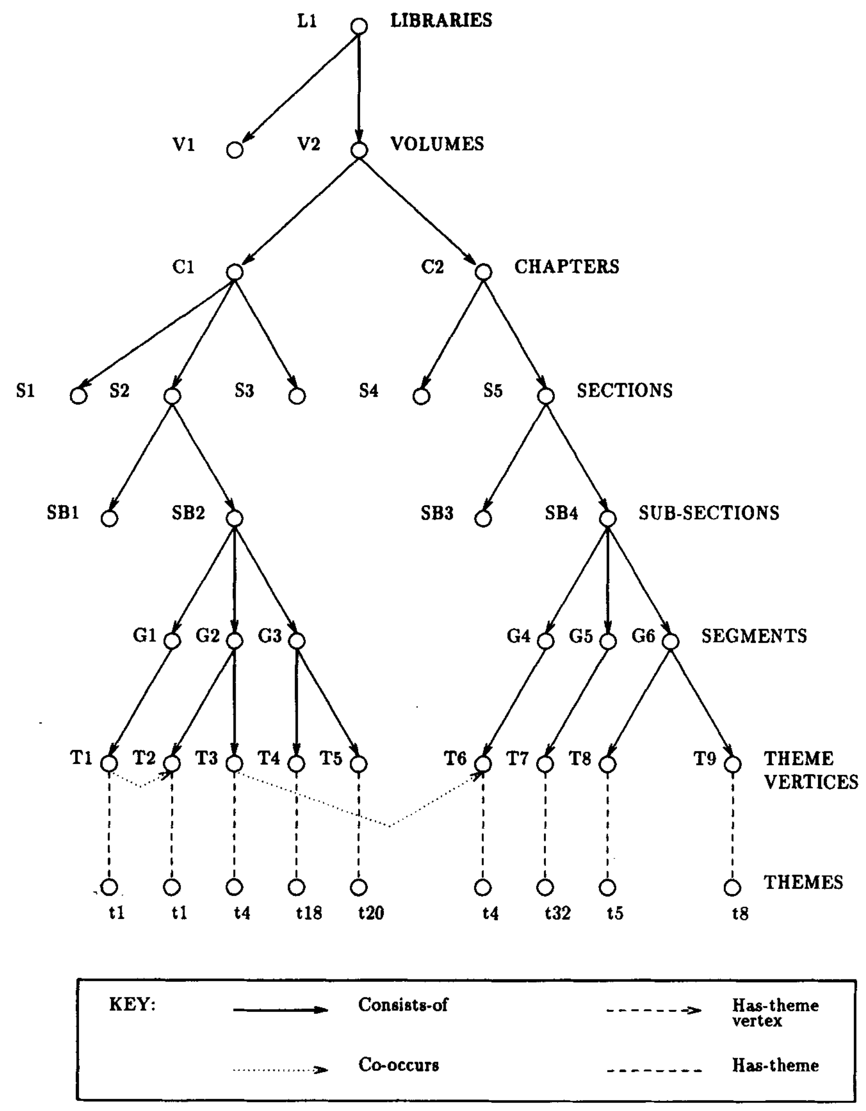

Figure 1. Ripple propagation graph

ies between theme vertices represent co-occurrences of these themes and form the thematic structure of documentation. The relationship between co-occurrences of themes is that of *co-occurs*.

A thematic interconnection graph (TIG) represents the connection between two segment entities for example $\{(G_1,G_2),(G_2,G_4)\}$ (see Figure 1). It is the TIG which records the potential ripple propagation in a documentation system.

Collectively the acyclic graph combining the hierarchical and thematic structure is called the ripple propagation graph (RPG) as it records how potential ripple effects can be propagated through the connectivity in a documentation hierarchy if one document entity is perturbed with a change. The RPG is a selection of documentation features which would otherwise not be visible because of the size and complexity of documentation. Formally the RPG is a tuple which consists of two sets:

RPG = (*Documentation Entities, Relations*)

(1) a set of *Documentation Entities* that are the components of interconnection, i.e., a set of graph vertices $\{n_1, ..., n_\xi\}$
(2) a set of *Relations* that define the interconnection that exists among documentation entities, i.e. a set of ordered pairs representing the graph edges $\{(n_i,n_j)|n_i,n_j \in$ *Documentation Entities*$\}$, called an edge set.

This can be further formalized:

(1) *Documentation Entities* =
Composite Components ∪ *Segment Components* ∪ *Thematic Components* ∪ *Themes*
(2) *Relations* =
Consists-of Relations ∪ *Has-theme Vertex Relations* ∪ *Has-theme Relations* ∪ *Co-occurs Relations*
   (a) *Consists-of Relation* =
   $\{(i,j) \mid i \in$ *Composite Components*, $j \in$ *Composite Components* ∪ *Segment Components*$\}$
   (b) *Has-theme Vertex Relation* =
   $\{(G,T) \mid G \in$ *Segment Components*, $T \in$ *Thematic Components*$\}$
   (c) *Has-theme Relation* =
   $\{(T,t) \mid T \in$ *Thematic Components*, $t \in$ *Themes*$\}$
   (d) *Co-occurs Relation* =
   $\{(t_1,t_2) \mid t_1,t_2 \in$ *Thematic Components*$\}$

The graph structure records different types of documentation such as data flow diagrams, algebraic specification, and other formally structured textual documentation. The RPGs provide an appropriate theoretical basis for empirical and observational studies and will also enable the development of new structural hypothesis about all material which serves to describe the semantics of software.

6.2. RPG Crystallization

The objective of RPG crystallization is to determine the constituent entities of the documentation. The technique partitions the document into entities, which are then further subdivided into other entities. This represents the *consists-of* dependency. For each document entity factored a new HIG is derived. This process is repeated until all the irreducible segment entities are identified. A ripple propagation graph can be crystallized from a documentation system or subsystem using the following steps:

(1) Create the hierarchical graph structure of the documentation.
(2) Define a set of application themes (data objects in the documentation).

(3) Analyse each Segment Entity for themes and for each theme found, create a theme vertex and attach it to the segment vertex. This records the thematic dependencies.
(4) Connect together each co-occurrences of themes.

The resulting frequencies of theme occurrences can be observed to determine how interconnected the documentation is in terms of the themes. Structured documentation and documentation with formally defined syntax and semantics are suitable for theoretical analysis and measurement. The mapping from the documentation domain to the ripple propagation graph co-domain 'θ: *Documentation* $\rightarrow$ RPG' must be bijective (symmetric) to facilitate objective analysis.

6.3. Set representation of the RPG

A notation is required which describes the possible connections between documentation entities and which can be analysed to deduce the possible consequence of modifying a particular documentation entity. Set notation is a suitable notation to represent the RPGs. A standard set representation of graphs can be used for describing the hierarchical structure from Library Entities down to Theme Vertices. Additional information is required in order to represent the co-occurrence relation between the theme vertices. This can be achieved by associating each theme vertex with a set of segments which refer to it. For example the RPG of Figure 1 can be described in the following way:

Composite Components = $\{L_1,V_1,V_2,C_1,C_2,S_1,S_2,S_3,S_4,S_5,SB_1,SB_2,SB_3,SB_4\}$

Segment Components = $\{G_1,G_2,G_3,G_4,G_5,G_6\}$

Thematic Components = $\{T_1,T_2,T_3,T_4,T_5,Y_6,T_7,T_8,T_9\}$

Themes = $\{t_1,t_4,t_5,t_8,t_{18},t_{20},t_{32}\}$
Hierarchical Interconnection Graph:

Consists-of Relations =
$\{(L_1,V_1),(L_1,V_2),(V_2,C_1),(V_2,C_2),(C_1,S_1),(C_1,S_2),(C_1,S_3),(C_2,S_4),(C_2,S_5),$
$(S_2,SB_1),(S_2,SB_2),(S_5,SB_3),(S_5,SB_4),(SB_2,G_1),(SB_2,G_2),(SB_2,G_3),$
$(SB_4,G_4),(SB_4,G_5),(SB_4,G_6)\}$

Thematic Interconnection Graph:

Has-theme Vertex Relations =
$\{(G_1,T_1),(G_2,T_2),(G_2,T_3),(G_3,T_4),(G_3,T_5),(G_4,T_6),(G_5,T_7),(G_6,T_8),(G_6,T_9)\}$

Has-theme Relations =

$\{(T_1,t_1),(T_2,t_1),(T_3,t_4),(T_4,t_{18}),(T_5,t_{20}),(T_6,t_4),(T_7,t_{32}),(T_8,t_5),(T_9,t_8)\}$

Co-occurs Relations = $\{(T_1,T_2),(T_3,T_6)\}$

6.4. Documentation ripple effect analysis

A logical ripple effect algorithum can be used to determine the transitive closure of a proposed change, thus determining all the parts of a document affected by a change. Each of the themes mentioned in a change proposal P are identified and recorded in a change set S, where

$$S = \{t \mid t \in \text{Themes and } t \text{ mentioned in } P\}$$

The impact of a change is derived using a ripple effect algorithm over the RPG. A change set which contains all the elicited themes from the change proposal are analysed in terms of their co-occurrences in the RPG to determine the worst possible ripple effect. The ripple effect algorithm described below is based on a graph slicing operation and is similar to the program slice introduced by Weiser (1984). The RPG slicing operation uses set intersection operations as a means of creating a new set from two existing sets, one of these sets being the sets contained within the RPG. For example an elementary slicing criterion of a documentation system RPG is a tuple (g,T), where g denotes a specific segment in RPG and T is a subset of themes in RPG. An elementary slicing criterion determines a projection function from a documentation segment trajectory in which only the value of themes in T are preserved. This RPG slicing operation provides a restriction transformation which provides a mapping from one set to another and removes any vertices in the RPG not satisfying a particular slicing criterion.

6.4.1. Documentation ripple effect analysis algorithm

The applications of algorithms over the RPGs can provide the maintainer with views of an application system. The graph slicing operation is denoted as:

$$\textit{Thematic Slice}(\text{RPG}(\textit{Documentation Entities, Relations}),S)$$

where S is the slicing criterion that restrict the vertices and edges which appear in the resulting graph. The graph slicing operation for a conducting documentation ripple effect analysis can be specified with the following sets which represent the applications of RPG slicing to remove RPG vertices and edges not impacted by the change. The algorithm for this slicing operation can be specified as follows:

Thematic Slice : RPG $\times$ $S \rightarrow$ RPG (V)

Thematic Slice (RPG, S) is defined as:

(1) Determine the *Thematic Components Impacted:*
  (a) *Thematic Components Impacted* =
    $\{T_i \mid (T_i,T_n) \in$ *Has-theme Relation* and $t_n \in S\}$
  (b) Extract all *Thematic Components Impacted* by S
(2) Determine the *Segment Components Impacted:*
  (a) *Segment Components Impacted* =

$\{G_i \mid (G_i,T_n) \in$ *Has-theme Vertex Relation* and
$T_n \in$ *Thematic Components Impacted*$\}$
(b) Extract all *Segment Components Impacted* by *Thematic Components Impacted*
(3) Determine the *Composite Components Impacted*:
(a) (i) *Composite Components Impacted* =
$\{A_i \mid (A_i,G_n) \in$ *Consists-of Relation* and $G_n \in$ *Segment Components*$\}$
(ii) Extract the *Composite Components Impacted;*
(b) (i) *Composite Components Impacted'* =
$\{A_i \mid (A_i,A_j) \in$ *Consists-of Relation* and $A_j \in$ *Composite Components Impacted*$\}$
(ii) Extract other *Composite Components Impacted*;
(c) Repeat step (3b) for each hierarchical level in the RPG until the newly calculated set of *Composite Components Impacted* does not change
(4) Determine all *Documentation Entitites Impacted*:
(a) *Documentation Entities Impacted* =
Composite Components Impacted $\cup$ *Segment Components Impacted*
(b) Determine the total *Documentation Entities Impacted*;
(5) Determine the size of the ripple effect in terms of segments affected:
(a) *Ripple Magnitude* = | *Segment Components Impacted* |
(b) Determine the magnitude of the ripple effect

6.4.2. *Example*

For example, given the RPG in (Figure 1) and given the set of themes S extracted from a change proposal describing changes to C_1, where $S = \{t_1,t_4,t_{20}\}$ the following sets can be derived by applying the above algorithm over the RPG.

(1) *Thematic Components Impacted* = $\{T_1,T_2,T_3,T_5,T_6\}$
(2) *Segment Components Impacted* $=\{G_1,G_2,G_3,G_4\}$
(3) *Composite Components Impacted* = five iterations of slicing operation 3:
(a) *Composite Components Impacted'* $=\{SB_2,SB_4\}$
(b) *Composite Components Impacted'* $=\{SB_2,SB_4,S_2,S_5\}$
(c) *Composite Components Impacted'* $=\{SB_2,SB_4,S_2,S_5,C_1,C_2\}$
(d) *Composite Components Impacted'* $=\{SB_2,SB_4S_2,S_5,C_1,C_2,V_2\}$
(e) *Composite Components Impacted'* $=\{SB_2,SB_4,S_2,S_5,C_1,C_2,V_2,L_1\}$
(4) *Documentation Entities Impacted* = $\{L_1,V_2,C_1,C_2,S_2,S_5,SB_2,SB_4,G_1,G_2,G_3,G_4\}$
(5) *Ripple Magnitude* = four segments affected

In this example although the change proposal S is directed towards the application documented in C_1, owing to the thematic interconnectivity recorded in the RPG, the ripple effect propagated to C_2 is detected.

In order to produce a graph-theoretic model RPG of the impact of the change proposal the sliced RPG can be reconstructed using the sets above. To summarize the algorithm above, the RPG′ is created by slicing a subgraph from a given RPG according to the documentation entities impacted by the change, i.e., set S. After applying this slicing algorithm over the RPG, the set of impacted segments are identified as the *Segment*

Components Impacted and the total cost of dealing with the ripple effect of the impact can be estimated. The cost of maintaining individual segments will be dependent on the specification and design documentation methods used. These costs can be determined experimentally. For example a segment in an algebraic specification will have a maintenance cost different from a segment in a data flow diagram method.

6.4.3. Probability connection matrix for documentation

Since the ripple effect algorithm above is based on existing ideas for ripple effect analysis on source code and module connection information (Yau and Collofello, 1980) and therefore detects all documentation containing themes that are present in a change proposal, it is possible that not all document entities will require maintaining.

The above algorithm detects the worst-case manifestation of the potential ripple effect. However, a situation could occur where a business feasibility study makes a strong case for a maintenance project but the impact of the technical proposal on the existing system indicates a net loss. This would mean that the project would not be carried out, although it is possible that the actual magnitude impact of the change may not be as bad as the ripple effect analysis indicates. To overcome this weakness the thematic ripple effect technique is extended to include some of the stochastic characteristics of the stability analysis techniques described in Song (1977) and Haney (1972). It seems logical to adjust the ripple effect interpretation with a stochastic element to produce the probable maximum ripple effect. This will allow the maintenance manager to make a logical inference about the probable maximum ripple effect of the project based on recorded past experience and thus help the maintenance manager focus on the important impacted documentation and source code. By combining change probability with the HIGs and TIGs a new kind of interconnection graph is contributed which concentrates on ripple propagation. This is why the name ripple propagation graph has been suggested.

A probability connection matrix for documentation can be updated for each project. It can be assued that documentation consists of n segments and that there are n^2 pairwise relationships of the form:

P_{ij} = Probability that a change in segment entity i propagates the requirement for a change in document segment entity j

The matrix is an $n*n$ structure with the documentation segment entities labelled on each side of the matrix G_1 to G_n. Each cell in the matrix represents the probability of a ripple effect between segment entities. Associated with each segment entity are segment attributes and are the following:

(1) Technology interfaces affected by a change to the segment.
(2) System interfaces affected by a change to the segment.
(3) Other application systems affected by change to the segment.
(4) Maximum source code defects introduced by changes to the segment.
(5) Maximum documentation code defects introduced by changes to the segment.
(6) Test files affected.
(7) Average number of lines of code affected by changes to the segment.
(8) Average cost of changing the segment and associated source code.

At the release phase of a project the release information is stored in the probability connection matrix. This information will be used to enhance the thematic ripple effect information particularly for risk assessment. Currently it is very difficult to obtain this type of information at an early phase in a maintenance project such as risk assessment.

The interconnection matrix can be used in the following way. For each pair of segment entities (G_1,G_2) impacted in the RPG:

(1) Consult the matrix to find the probability of the ripple effect actually occurring between G_1 and G_2.
(2) If the ripple effect is highly probable, then the attributes associated with G_2 can then be investigated.
(3) The attributes associated with G_2 provide information on the impact of G_2 on the system, the impact on other systems, the impact on the source code, and the likely defects which may be introduced when changing this particular segment.

The information in the probable impacted segment attributes should be accumulated over the life of the system and be used to help a maintenance manager to answer the following types of questions concerning project priorities:

(1) What are the technology interfaces affected by the change?
(2) What are the system interfaces affected by the change?
(3) What are the other application systems affected by a proposed change?
(4) What is the magnitude of the affected areas of documentation and source code?
(5) How much source code will need to be understood to make the change?
(6) What is the risk of introducing defects if these areas are maintained?
(7) How much program testing will be required?
(8) What is the probable cost of dealing with the affected areas of documentation and source code?

The thematic slicing algorithm determines the segment of documentation which might be impacted by a proposed software change. By interpreting the probability connection matrix which contains document segment change history information, the above questions can be answered. The answers to the above questions will provide the maintenance manager with an understanding of the potential spreading of ripple effects and their consequences. This will enable a quick validation of the maintenance manager's own intuitive understanding of the effect of a proposed change. If an application subsystem interface is affected by the change proposal but the subsystem does not seem intuitively affected then this module can be investigated at an early phase in the maintenance process.

The linking of syntactic constructs in the source code with the thematic structure will also improve the precision of the ripple effect information produced at this early phase.

7. CONCLUSIONS

This paper investigates a technique for the early detection of ripple propagation when making a change to software as part of the software maintenance process. Identifying change proposals which affect documentation and source code components which are potential high-risk ripple propagators, can allow a maintenance manager to optimize the allocation of budget and effort.

Existing impact and stability analysis measurement techniques have been reviewed and evaluated. While existing techniques provide a good example of how to conduct ripple effect and stability analysis, a weakness in these approaches is that only the source code is considered when analysing contending change proposals. The main thesis of this work rejects the idea of analysing the impact of a software change on a system by only considering the application source modules and excluding documentation. Source-code-based ripple effect techniques imply that the source code is well understood when the techniques are applied. This is not the case at the investment appraisal phase of the project as only some preliminary high-level design work will have been conducted at this phase. Therefore, if impact analysis techniques are to be used at this early phase, they must address the part of the software which is at a similar level of abstraction to that of the feasibility study or change proposal, that is the documentation.

A new ripple effect technique based on a graph-theoretic approach to the modelling of documentation has been presented. Using this approach facilitates a more objective analysis of the impact of a change on the intrinsic structure of documentation at the risk assessment phase. To analyse these graphs a ripple effect algorithm has been produced based on existing techniques for ripple effect analysis and the thematic structure of documentation. This algorithm is also combined with historical data to provide an assessment of a more probable impact of a change in addition to the worst case impact of a change.

The limitation of the present approach is that the ripple propagation graph crystallization process may not produce graphs which are 100% reproducible. This is because the allocation of a segment document entity to a theme category depends on the segment theme interpretation of the human and therefore is currently subjective. An efficient segment document characterization and classification scheme is important in order to obtain a bijective mapping between the real world documentation and the theoretical graph model. Document entity content description graphs called thematic property models are being investigated for such a characterization and classification scheme.

Using thematic ripple effect analysis an ordered preference relationship can be developed for contending change proposals. It is argued that it is possible to get some indication of the impact of a change to documentation and source code at an early phase in the maintenance process using thematic structure algorithms, but this must be revised with existing techniques later in the maintenance project. With this hybrid ripple effect analysis approach applied earlier in maintenance projects, the value added to project scheduling should be increased.

References

Arthur, J. (1988) *Software Evolution*, John Wiley, New York, pp. 74–75.

Collofello, J. S. and Vennergrund, D. A. (1987) 'Ripple effect based on semantic information', *Proceedings AFIPS Joint Computer Conference*, Vol. 56, pp. 675–682.

Foster, J. R. (1989) 'Priority control in software maintenance', *Proceedings of the 7th International Conference Software Engineering for Telecommunications Switching Systems*, Bournemouth, UK, pp. 163–167.

Haney, F. M. (1972) 'Module connection analysis', *Proceedings AFIPS Joint Computer Conference*, Vol. 41(5), pp. 173–179.

Lientz, B. P. and Burton Swanson, E. (1980) *Software Maintenance Management*, Addison-Wesley, Reading, MA.

Liu, C. (1976) 'A look at software maintenance', *Datamation*, **22**(11), 51–55.

Martin, J. and McClure, C. (1983) *Software Maintenance — The Problem and its Solution*, Prentice Hall, Englewood Cliffs, NJ.

Nosek, J. T. and Prashant, P. (1990) 'Software maintenance management: change in the last decade', *Journal of Software Maintenance Research and Practice*, **2**(3), 157–174.

Osborne, W. M. (1987) 'Building and sustaining software maintainability', *Proceedings, Conference Software Maintenance*, 1987, IEEE Computer Society Press, Washington, DC, pp. 13–23.

Parikh, G. (1982) *Some Tips, Techniques, and Guidelines for Program and System Maintenance*, Winthrop Publishers, Cambridge, MA, pp. 65–70.

Patkau, B. H. (1983) 'A foundation for software maintenance', Ph.D. Thesis, Department of Computer Science, University of Toronto.

Pfleeger, S. L. and Bohner, S. A. (1990) 'A framework for software maintenance metrics', *Proceedings, Conference Software Maintenance*, 1989, IEEE Computer Society Press, Washington, DC, pp. 320–327.

Pfleeger, S. L. and Shawn, A. B. (1990) 'A framework for software maintenance metrics' *Proceedings, Conference Software Maintenance*, 1989, IEEE Computer Society Press, Washington, DC, pp. 320–321.

RADC (1986) 'Automated life cycle impact analysis system', Technical Report, RADC-TR-86-197, Rome Air Development Center, Air Force Systems Command, Griffiths Air Force Base, Rome, New York.

Robillard, P. N. and Boloix, G. (1989). 'The interconnectivity metrics: a new metric showing how a program is organised', *Journal of Systems and Software*, **10**, 29–38.

Sharpley, W. K. (1977) 'Software maintenance planning for embedded computer systems', *Proceedings, IEEE COMPSAC 77*, pp. 520–526.

Song, N. L. (1977) 'A program stability measure', *Proceedings. 1977 Annual ACM Conference*, pp. 163–173.

Swanson, E. B. (1976). 'The dimensions of maintenance', *Proceedings of 2nd IEEE International Conference on Software Engineering*, pp. 492–497.

Weiser, M. D. (1984) 'Program slicing', *IEEE Transactions on Software Engineering*, **SE-10**(4), 352–357.

Wild, C., Maly, K. and Liu, L. (1991) 'Decision-based software development', *Software Maintenance: Research and Practice*, **3**(99), 17–43.

Yau, S. S. and Chang, S. C. (1984). 'Estimating logical stability in software maintenance', *IEEE Computer Society Computer Software and Applications Conference*, pp. 109–119.

Yau, S. S. and Collofello, J. S. (1980). 'Some stability measures for software maintenance', *IEEE Transactions on Software Engineering*, **6**(6), 545–552.

Yau, S. S. and Collofello, J. S. (1985) 'Design stability measures for software maintenance', *IEEE Transactions on Software Engineering*, **SE-11**(9), 849–856.

Yau, S. S., Collofello J. S. and MacGregor, T. (1978) 'Ripple effect analysis of software maintenance', *Proceedings IEEE COMPSAC*, pp. 60–65.