# Change Impact Analysis with a Goal-Driven Traceability-Based Approach

Wen-Tin Lee,* Whan-Yo Deng,† Jonathan Lee,‡ Shin-Jie Lee§
*Software Engineering Lab., Department of Computer Science and Information Engineering, National Central University, Jhongli, Taiwan 320*

Recently, the growing popularity of requirements engineering attracts an increasing attention on requirements traceability and change impact analysis, which also imposes a great demand for a systematic approach in developing software systems to handling traceability relations and requirements changes in an automatic manner. In this work, a goal-driven requirements traceability approach is proposed to develop and manage requirements changes along three dimensions: (1) to develop software and manage requirements based on the goal-driven use case (GDUC) approach, (2) to establish and maintain the traceability relation with a design structure matrix (DSM) to derive the traceability tree, and (3) to analyze requirements change impacts through the partitioning of DSM into blocks to serve as a basis for calculating use case points. The proposed approach is illustrated by a benchmark problem domain of a meeting scheduler system. © 2010 Wiley Periodicals, Inc.

## 1. INTRODUCTION

A major challenge in requirements management is that creeping requirements, namely, changes in current requirements are not controlled or analyzed, affect all downstream deliverables.[1] Consequently, projects with such creeping requirements are likely to fail. Recent progress in requirements traceability has demonstrated its applicability to performing impact analysis of requirements changes and to ensuring all source requirements being fully addressed.[2]

Analyzing change impacts with requirements traceability usually encounter three main problems:[3] (1) the traceability relations to be maintained are often imprecise and out of date, (2) the establishment of traceability relations among requirements and work products is not integrated with the development process, and (3) the manual impact analysis is tedious and time consuming.

*Author to whom all correspondence should be addressed: e-mail: wtlee@selab.csie.ncu.edu.tw.
†e-mails: deng@selab.csie.ncu.edu.tw.
‡e-mail: yjlee@selab.csie.ncu.edu.tw.
§e-mail: jielee@selab.csie.ncu.edu.tw.

INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, VOL. 25, 878–908 (2010)
 Published online in Wiley InterScience (www.interscience.wiley.com). • DOI 10.1002/int.20443

Several investigations have been conducted to address the requirements traceability problems, either by generating and maintaining traceability relations,[4–9] or by defining and adapting traceability models.[10–12] Inspired by the requirements traceability reference model proposed by Ramesh and Jarke,[10] and as a continuous endeavor of our previous work on goal-driven requirements engineering to analyze the interactions among requirements,[13–17] this work presents a goal-driven approach with two key features to establishing the trace relations of goals and use cases:

- to identify the three types of links proposed in the reference model: evolution, dependency, and satisfaction, to serve as a basis for formulating the trace relations among goals and use cases; and
- to establish and maintain the traceability relation with design structure matrix (DSM),[18–22] and to utilize the DSM partition mechanism to perform change impact analysis.

The meeting scheduler problem,[23] a widely used benchmark problem domain in requirements engineering community, is adopted throughout this work as an example to illustrate the proposed approach. In the sequel, we give an outline of our goal-driven use case model as a background information in Section 2.1, detail the main features of the proposed approach in Section 3, compare related work on requirements traceability and change impact analysis in Section 4, and finally summarize the benefits of the fusion of goal-driven approach and DSM in Section 5.

## 2. BACKGROUND WORK

In this session, we introduce background information on work that have significant impacts on this work, especially, researches on goal-driven use case[13,14] and design structure matrix.[18,20–22]

### 2.1. Goal-Driven Use Case Method

A brief summary of our goal-driven use case model[13] to construct a use case model with goals is outlined below for the readers' reference. To identify goals from domain descriptions and system requirements, we propose a faceted classification scheme so that each goal can be classified with three facets: *competence*, *view*, and *content*. The *competence* describes whether a requirement is satisfied completely or only to a degree. A *rigid* type of goal describes a minimum requirement that a target system must satisfy utterly. A *soft* goal describes properties or concerns that stakeholder care about for a target system and can be satisfied to a degree. The *view* facet concerns whether a goal is *actor specific* or *system specific*. Actor-specific goals are actors objectives in using a system; system-specific goals are requirements on services that the system provides. A goal can be further distinguished based on its *content* and can be either related to a systems functional aspects or associated with the systems *nonfunctional* aspect.

A goal can be achieved, optimized, or maintained by its associated use case. The use cases of the meeting scheduler system are established for actors *meeting initiator* and *meeting participant* (see Figure 1). For the actor meeting initiator, the use case *Plan a meeting* covers the scenario for an initiator to achieve an original goal

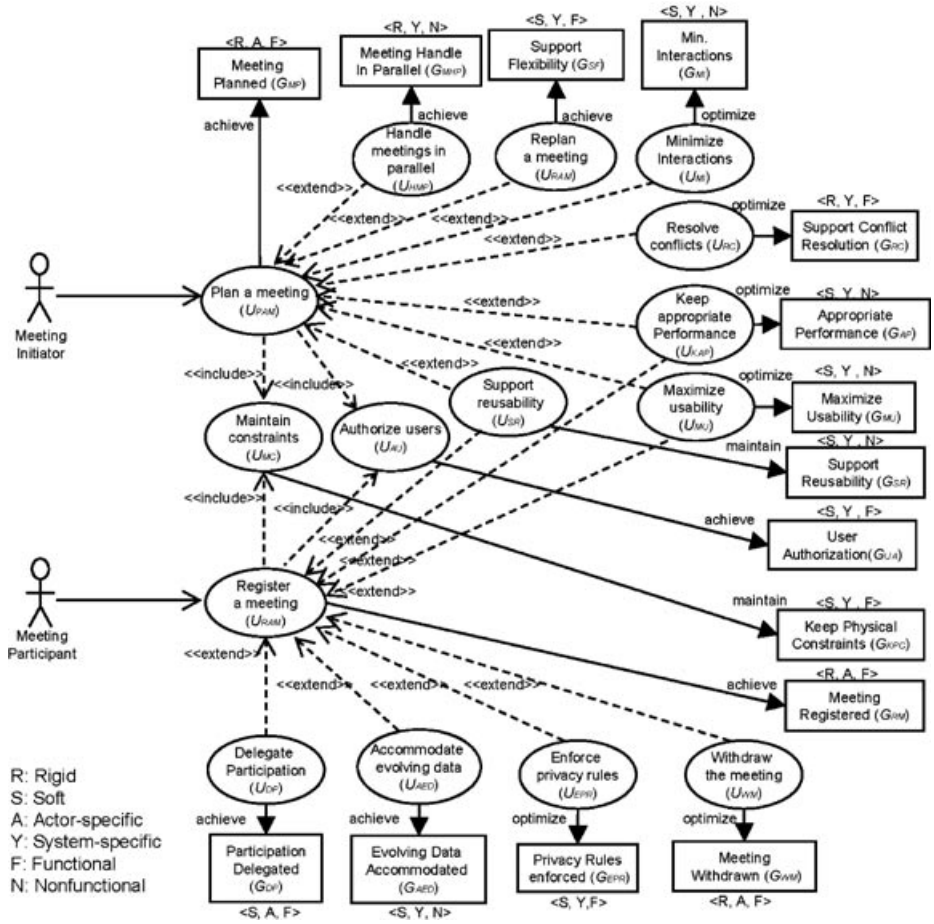



**Figure 1.** Use case model for meeting scheduler system template.

*MeetingPlanned*, which is a goal of rigid, actor specific, and functional. Meanwhile, in the case of the actor *meeting participant*, the use case *Register a meeting* covers the case for a participant to achieve an original goal *MeetingRegistered*, which is a goal that is rigid, actor specific, and functional.

To achieve a system-specific goal, an extension use case may be created. Referring to our example, the original use case *Plan a meeting* describes the process to create a meeting from the view of the actor *initiator*. The extension use cases *Handle meetings* in parallel and *Resolve conflicts* extends it to take all initiators into account, that is, to achieve the system-specific goals *MeetingHandleInParallel* and *SupportConflictResolution*. The extension use cases *Accommodate evolving data* and *Enforce privacy rules* extend the original use case *Register a meeting* to achieve the soft, system-specific, and functional goals *EvolvingDataAccommondated* and *PrivacyRulesEnforced*, respectively.

To achieve a nonfunctional goal, an extension use case serves as a constraint to qualify its original use case. In our example, several constraints (may be rigid or soft) on a meeting are considered as extension use cases to extend or constrain the behavior of the original use case *Plan a meeting*, which is a direct course to create a meeting, including *Minimize Interactions*, *Keep Appropriate Performance*, *Support Reusability*, and *Maximize Usability*. To enhance reusability, the use case models are further elaborated by extracting the common fragments among various use cases into an included use case by using "include" relationships. For example, the use cases *Maintain constraints* and *Authorize users* are included by the use case *Plan a meeting* and use case *Register a meeting*.

### 2.2. Design Structure Matrix

The design structure matrix (DSM) developed by Steward[18] is a square matrix with identical row and column labels to identify the dependencies between tasks and to sequence the engineering design processes. DSM is a complexity management tool to design and optimize complex systems, project tasks, and organization structures. There are three basic configurations in a DSM: parallel, sequential, and coupled, to describe the relationships among the system elements (see Figure 2).

The parallel configuration is a configuration of no interaction between the two elements A and B, and the DSM entries between these two elements contain no marking. The sequential configuration shows that if element A sends information to or influences the behavior of element B, then the (Column A, Row B) entry contains a mark. The coupled configuration indicates that if two elements A and B require information from each other or influence each other, then each entry of (A,B) and (B,A) in the DSM contains a mark.

In Ref. 20, Browning reviewed four DSM applications to demonstrate their usefulness for product and process development, project planning and management, system engineering, and organization design. The four DSM applications, including component-based, team-based, activity-based, and parameter-based DSM, are categorized into Static DSM and Time-based DSM.

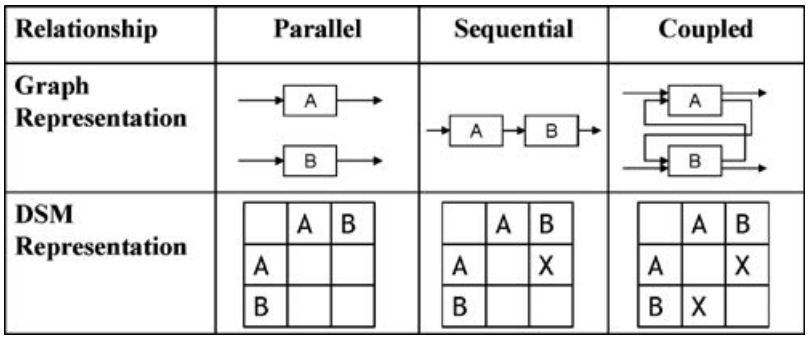



**Figure 2.** DSM configurations.

- *Static DSM*: representing system elements existing at the same time, including component-based and team-based DSM.

    1. *Component-based or Architecture DSM*: A system architecture can be modeled in terms of the relations among its components/elements. The potential reintegration of the elements can be further analyzed via clustering.
    2. *Team-based or Organization DSM*: An organization can be decomposed into teams, and modeled as a system by documenting the interactions between the teams. The integration analysis can be applied to cluster teams into metateams and to minimize the interactions among clusters.

- *Time-based DSM*: representing system elements in a flow through time, including activity-based and parameter-based DSM.

    1. *Activity-based or Schedule DSM*: A process can be modeled through its constituted activities by documenting the information flow among the activities. The iterations/feedbacks in the processes can then be minimized by analyzing the DSM using sequence analysis methods, such as partitioning, tearing, banding, simulation, and eigenvalue analysis.
    2. *Parameter-based DSM*: The low-level activities, design variables, system parameters can be modeled by documenting the interrelationships between the parameters. The sequencing analysis methods can be utilized to reduce process duration and enhance design quality.

DSM employs several analysis methods to optimize complex systems and project tasks, such as partitioning, clustering, and simulation.[21,22]

## 3. GOAL-DRIVEN TRACEABILITY-BASED APPROACH TO CHANGE IMPACT ANALYSIS

There are four main features involved in this work to establish the traceability relations among goals and use cases and to evaluate the change impacts (see Figure 3 for an overview):

1. *G2U* and *U2U* relation identification: Goal to use case (*G2U*) evolution links and use case to use case (*U2U*) dependency links are identified and maintained in the DSM.
2. *U2G* and *G2G* relation evaluation: Users are engaged to identify the satisfaction links related to use case to goal (*U2G*), and the goal-to-goal (*G2G*) dependency links are then established automatically in the DSM based on graph theory.
3. DSM partitioning and traceability tree derivation: The DSM, with four submatrices: *G2U*, *U2U*, *U2G*, and *G2G*, is partitioned into blocks to derive the traceability tree from the DSM.
4. Change impact analysis: When user proposes changes during software evolution, change impacts are analyzed to find affected requirements as well as affected use cases and their corresponding use case points.[24]

### 3.1. G2U and U2G Relation Identification

Although DSM is a powerful complexity management tool to design and optimize the complex system with various DSM techniques, it is still limited in systematically identifying and capturing the interactions/relations among the system

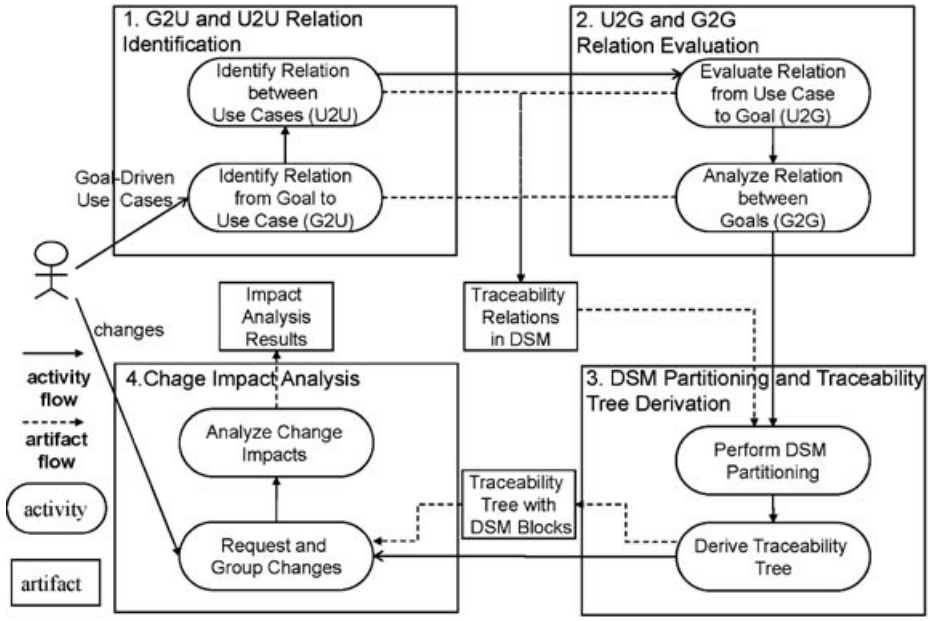



**Figure 3.** Overview of GART

elements. This work presents a systematic way to evaluate the relations between goals and use cases. A DSM is divided into four submatrices: *G2U*, *U2U*, *U2G*, and *G2G* matrices, to capture the traceability links (see Figure 4). Detail discussions are as follows.

### *3.1.1. Identify Relation from Goal to Use Case*

To capture the links between goals and use cases, the three link types of traceability relations: evolution, dependency and satisfaction, between goals and use cases are required to be elaborated. Traceability link is deemed as an impact relation to reflect its applicability to performing impact analysis of the requirement changes. The impact relation can be applied to work products as a result of performing a process, such as goals, use cases, designs, test cases, etc. An *impact relation* from a work product $x$ to a work product $y$ indicates that by changing work product $x$, work product $y$ may be affected, which is formally defined below.

DEFINITION 1. *Let* $\boldsymbol{R}$ *be the impact relation on a set of work products* $\boldsymbol{W}$. *For every* $x, y \in \boldsymbol{W}$, $x\ \boldsymbol{R}\ y$ *if and only if change* $x$ *may affect* $y$. $\boldsymbol{R}$ *is transitive because if* $x$ *impact* $y$ *and* $y$ *impact* $z$ *then it follows that* $x$ *impact* $z$ *for every* $x, y, z \in \boldsymbol{W}$.

Figure 5 illustrates the three types of traceability links between goals and use cases. In Figure 5a, a goal evolves to a use case, namely, a goal has an evolution link

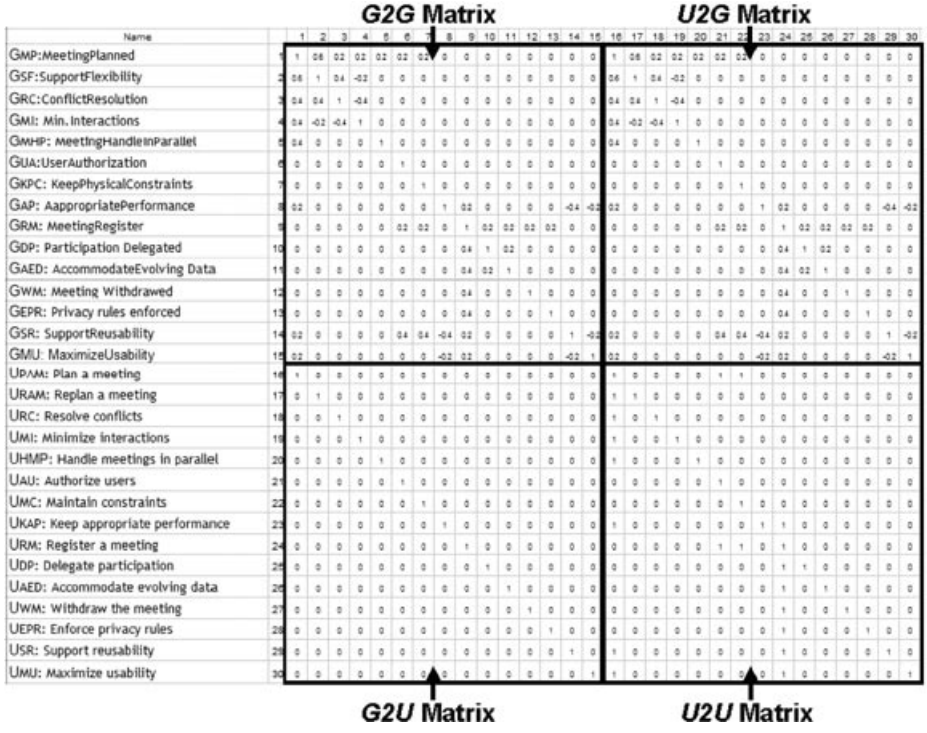



**Figure 4.** Traceability links between goals and use cases in DSM.

to its derived use case, by changing the goal, its derived use case may be affected. Therefore, we can view evolution link as a kind of impact relation. In Figure 5b, a goal/use case depends on another goal/use case, that is, a goal/use case has a dependency link to its dependent goal/use case, changing the goal/use case may affect its dependent goal(s)/use case(s). Thus, we can treat dependency link as a kind of impact relation. In Figure 5c, a use case satisfies its related goals to some

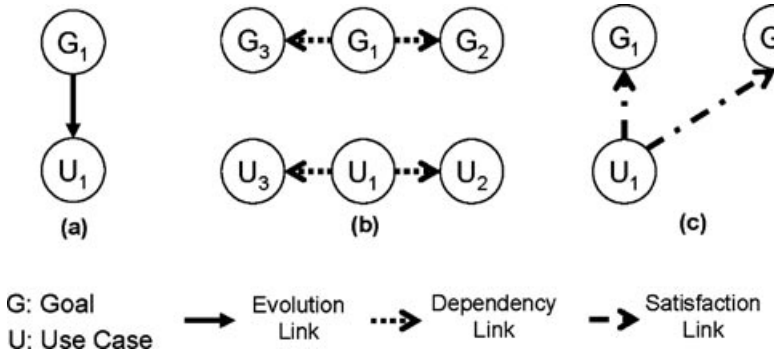



**Figure 5.** Link types between goal and use case.

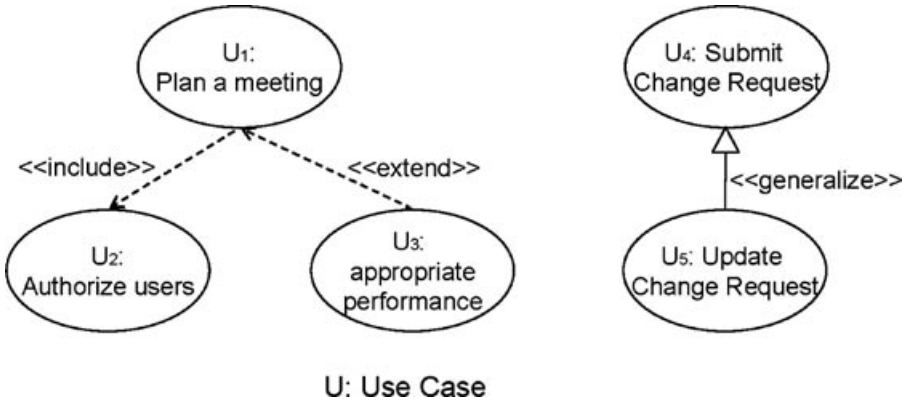



**Figure 6.** Use case to use case dependency link.

degree, i.e. a use case has a satisfaction link to its related goal, changing the use case may affect its related goal(s). The definition of these three types of traceability links is formally given below.

DEFINITION 2. *Let evolution link, satisfaction link, and dependency link* $\in \boldsymbol{R}$ *be a kind of impact relation. The relations between goal and use case are defined by*

1. *an impact relation from a goal to a use case is an evolution link.*
2. *an impact relation between goals/use cases is a dependency link.*
3. *an impact relation from a use case to a goal is a satisfaction link.*

We begin with identifying the evolution links from goals to use cases after the goals and use cases have been modeled. Since each goal is evolved to its associated use case, the goal to use case evolution links is one-to-one relations and is kept in the *G2U* submatrix of the DSM (see *G2U* matrix in Figure 4). The $(G_i, U_i)$ entries (for $i = 1, \ldots, n$) in DSM are marked as "1" to indicate the evolution links between them.

Referring to our example, the goal to use case evolution links—a one-to-one relation, of the meeting scheduler system are identified in *G2U* matrix in Figure 4. The evolution link from goal $G_{MP}$ to use case $U_{PAM}$ is identified since use case $U_{PAM}$ is discovered with respect to goal $G_{MP}$. The $(G_{MP}, U_{PAM})$ entry in *G2U* matrix is marked as "1" to indicate the evolution link from goal $G_{MP}$ to use case $U_{PAM}$. As a result, *G2U* Matrix in Figure 4 is a diagonal matrix since goals $\{G_{MP}, \ldots, G_{MU}\}$ have an evolution link to use case $\{U_{PAM}, \ldots, U_{MU}\}$, respectively.

### *3.1.2. Identify Relation between Use Cases*

The use case dependency links are illustrated in Figure 6. Use case $U_1$ includes use case $U_2$ and is extended by use case $U_3$. Use case $U_5$ generalizes use case $U_4$. U1 depends on $U_2$ through the "include" relation between $U_1$ and $U_2$, which implies that a change in $U_2$ may influence $U_1$. $U_3$ depends on $U_1$ through the "extend"

relation between $U_1$ and $U_3$, that is, a change in $U_1$ may influence $U_3$. Owing to the semantics of the "generalize" relation, $U_5$ depends on $U_4$, since changing $U_4$ may influence $U_5$ but not being influenced by any changes in $U_5$.

When *include*, *extend* or *generalize* relation occurs between use cases, the corresponding DSM entries are marked as "1" based on the dependency links between them in the *U2U* submatrix of the DSM (see *U2U* matrix in Figure 4). The use case to use case dependency links extracted from the use case model of meeting scheduler system is identified in *U2U* matrix in Figure 4. Use cases $U_{RAM}$, $U_{RC}$, $U_{MI}$, $U_{FMT}$, and $U_{KAP}$ depend on $U_{PAM}$ through the "extend" relations between them, that is, a change in $U_{PAM}$ may influence $U_{RAM}$, $U_{RC}$, $U_{MI}$, $U_{FMT}$, and $U_{KAP}$. To indicate the dependency links between these use cases, the entries ($U_{PAM}$, $U_{RAM}$), ($U_{PAM}$, $U_{RC}$), ($U_{PAM}$, $U_{MI}$), ($U_{PAM}$, $U_{FMT}$), and ($U_{PAM}$, $U_{KAP}$) are marked as "1" in the *U2U* matrix.

Use cases $U_{PAM}$ and $U_{RM}$ depend on $U_{MC}$ through the "include" relations between them, namely, a change in $U_{MC}$ may influence $U_{PAM}$ and $U_{RM}$. The corresponding DSM entries ($U_{MC}$, $U_{PAM}$) and ($U_{MC}$, $U_{RM}$) in *U2U* matrix are marked as "1" based on the dependency links between them. The diagonal entries in *U2U* matrix are marked as "1" to specify that each use case has a dependency link to itself.

### 3.2. U2G and G2G Relation Evaluation

Traceability link strength and direction are crucial factors in structuring requirements traceability. The satisfaction degree of *U2G* satisfaction links are evaluated and resulted in *U2G* matrix in the "Evaluate Relations from Use Case to Goal" section. The *G2G* dependency links between goals are analyzed to produce *G2G* matrix in the "Analyze Relations between Goals" section.

#### *3.2.1. Evaluate Relation from Use Case to Goal*

Goal and use case evaluation process involves measuring the satisfaction degree of *U2G* satisfaction links, which are maintained in the *U2G* submatrix of the DSM (see *U2G* matrix in Figure 4). In GDUC,[13] each goal $G_i$, where $\{G_i | \ 1 \leq i \leq n$, $n$ is the total number of goals$\}$, is achieved/optimized/maintained by its directly associated use case $U_m$, where $\{U_m | \ 1 \leq m \leq n, n$ is the total number of use cases$\}$. In addition to the directly achieved/optimized/maintained relationships, we further examine the relationships between goals and use cases caused by side effects. The side effects to a goal $G_i$, where $\{G_i | \ 1 \leq i \leq n\}$, are analyzed by considering the effects of a use case $U_m$ to the goal $G_i$, where $\{U_m | \ 1 \leq m \leq n, \ i \neq m\}$. By investigating all the effects, including the side effects among goals and use cases, the relationships between goals and use cases—the satisfaction degree, can be determined. In the evaluation, the satisfaction degree ($S_{\text{degree}}$) of a goal is rated from $-5$ to 5 to represent the satisfaction degree to the goal while performing the use case. The score can be assigned by domain experts based on a rating table (see Table I as suggested in Ref. 25). In Table I, five means the goal can be fully satisfied by the use case; $-5$ means the goal will be fully denied by the use case; and 0 means

**Table I.** Define ratings of satisfaction links.

| Score | Explanation |
|---|---|
| 5 | The goal is fully satisfied after the use case is performed |
| 3 | The goal is largely satisfied after the use case is performed |
| 1 | The goal is partially satisfied after the use case is performed |
| 0 | The goal is not affected after the use cases is performed |
| −1 | The goal is partially denied after the use case is performed |
| −3 | The goal is largely denied after the use case is performed |
| −5 | The goal is fully denied after the use case is performed |
| 2,4, −2, −4 | Represent the degrees between scores listed above |

the use case does not have any effect on the goal. Detail explanation of the rating of satisfaction degree can be found in Table I.

We further normalize the satisfaction degree to $S_{\text{degree}}/5$ ($-1 \leq S_{\text{degree}}/5 \leq 1$) to obtain the link strength of *U2G* satisfaction link. A fuzzy threshold *T1* is introduced to provide the flexibility that the satisfaction link can be filtered out if $|S_{\text{degree}}/5| < T1$. The default value of fuzzy threshold *T1* is 0 to keep all the satisfaction links identified by users.

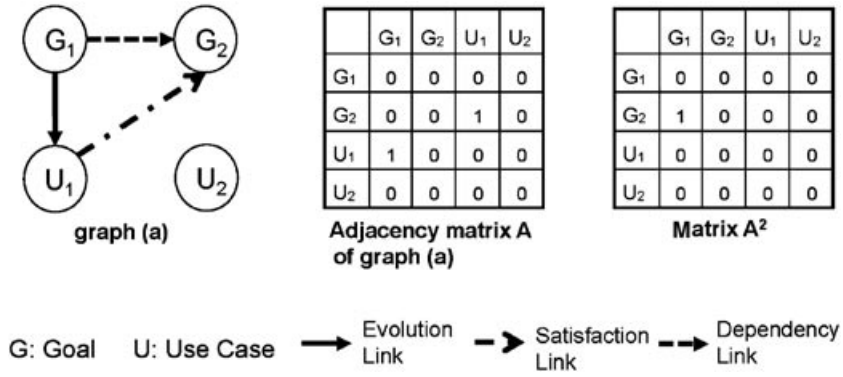
To analyze the satisfaction links from use cases to goals, we examine the relationships among goals and use cases in a pairwise manner by means of Table I, which results in the satisfaction links in *U2G* matrix in Figure 4. For example, the effect of performing use case $U_{PAM}$ is evaluated with respect to all goals in the system. $G_{MP}$ is rated as 5 since performing $U_{PAM}$ can fully satisfy the goal. $G_{SF}$ is rated as 3 to indicate that performing $U_{PAM}$ can largely satisfy the goal. The effect of performing use case $U_{RC}$ with respect to goal $G_{MI}$ is rated as $-2$ to show that performing $U_{RC}$ can partially to largely deny the goal. The link strength of satisfaction links ($U_{PAM}$, $G_{MP}$), ($U_{PAM}$, $G_{SF}$), and ($U_{RC}$, $G_{MI}$) are normalized to 1, 0.6, and $-0.4$ in *U2G* matrix, respectively.

### *3.2.2. Analyze Relation between Goals*

To derive the dependency links between goals, we formulate goals and use cases as a vertex set and traceability links as edges between goals and use cases in a graph. The *G2U*, *U2U*, and *U2G* links identified in previous sections can be captured in the adjacency matrix of goals and use cases. Figure 7 illustrates how the concept works behind this formulation.

In Figure 7, goal $G_1$ is evolved to use case $U_1$, which satisfies goal $G_2$ to some degree. The adjacency matrix $A$ of graph (a) is identified and the entries ($G_1$, $U_1$) and ($U_1$, $G_2$) are marked as 1 to indicate the evolution and satisfaction links, respectively. The entry ($G_1$, $G_2$) in $A^2$, the square of matrix $A$, is 1, which means that there exists an edge sequence of length 2 from $G_1$ to $G_2$. This edge sequence is a path through $G_1 \rightarrow U_1 \rightarrow G_2$, an evolution link from $G_1 \rightarrow U_1$ and a satisfaction link from $U_1 \rightarrow G_2$, which implies there exists a dependency link from $G_1$ to $G_2$, according to the Definitions 1 and 2.

Theorem 1 summarizes the formulation to derive goal-to-goal (*G2G*) dependency links based on graph theory (see Ref. 26 for a reference.)




| | G1 | G2 | U1 | U2 |
|---|---|---|---|---|
| G1 | 0 | 0 | 0 | 0 |
| G2 | 0 | 0 | 1 | 0 |
| U1 | 1 | 0 | 0 | 0 |
| U2 | 0 | 0 | 0 | 0 |

| | G1 | G2 | U1 | U2 |
|---|---|---|---|---|
| G1 | 0 | 0 | 0 | 0 |
| G2 | 1 | 0 | 0 | 0 |
| U1 | 0 | 0 | 0 | 0 |
| U2 | 0 | 0 | 0 | 0 |

**Figure 7.** Graph and its adjacency matrix.

THEOREM 1. *Let $T$ be a graph with vertex set $\{G_1, \ldots, G_n\}$, $\{U_1, \ldots, U_n\}$ and adjacency matrix $A$ with G2U, U2U and U2G traceability links ($n$ = number of goals/use cases).*

*If the $(i, j)$-entry of $A^2 > 0$ where $i, j = 1 \ldots n$, then $G_i$ has a dependency link to $G_j$.*

*Proof.* The $(i, j)$-entry of $A^2$ is the number of edge sequences of length 2 from $G_i$ to $G_j$ with a path $G_i \rightarrow U_i \rightarrow G_j$. Suppose the $(i, j)$-entry of $A^2 > 0$, that is, there exists at least one edge sequence through the path $G_i \rightarrow U_i \rightarrow G_j$. Therefore, $G_i$ has an evolution link to $U_i$ and $U_i$ has a satisfaction link to $G_j$. According to Definitions 1 and 2, evolution link and satisfaction link are a kind of impact relation that is transitive, $G_i$ has an impact relation to $G_j$. From Definition 2, the impact relation between goals is a dependency link. Thus, $G_i$ has a dependency link to $G_j$. ■

Referring to our meeting scheduler system, we obtain a *G2G* matrix in Figure 4 with the dependency links between goals. For example, the values 0.6 and 0.4 of the entries ($G_{MP}$, $G_{SF}$) and ($G_{MP}$ , $G_{RC}$) in *G2G* matrix in Figure 4 are added from the same entries in the $A^2$ matrix created by applying Theorem 1, which means that there exist two dependency links from $G_{MP}$ to $G_{SF}$ and from $G_{MP}$ to $G_{RC}$, respectively. The two dependency links are generated through the paths $G_{MP} \rightarrow U_{PAM} \rightarrow G_{SF}$ and $G_{MP} \rightarrow U_{PAM} \rightarrow G_{RC}$. This indicates that to change $G_{MP}$ may affect $G_{SF}$ and $G_{RC}$, and therefore, $G_{SF}$ and $G_{RC}$ depend on $G_{MP}$.

Figure 4 presents the DSM with *G2U*, *U2U*, *U2G*, and *G2G* submatrices to show all the traceability links between goals and use cases. The entries in the *G2U* matrix containing the evolution links from goal to use case are kept in the $(i,j)$-entry (where $i = 16, \ldots, 30$, and $j = 1, \ldots, 15$) of the DSM. The entries in the *U2U* matrix, which contains the dependency links between use cases, are kept in the $(i,j)$-entry (where $i = 16, \ldots, 30$, and $j = 16, \ldots, 30$). The satisfaction degree $S_{\text{degree}}$ of *U2G* satisfaction links are established and normalized in the $(i,j)$-entry

(for $i = 1, \ldots, 15$, $j = 16, \ldots, 30$). The entries in the *G2G* matrix containing the dependency links between goals are kept in the $(i,j)$-entry (where $i = 1, \ldots, 15$, and $j = 1, \ldots, 15$).

### 3.3. DSM Partitioning and Traceability Tree Derivation

DSM partitioning is adopted here to group the goals and use cases into blocks to assist project managers to manage the requirements and plan the successive project tasks. The traceability tree can be derived from the DSM partition blocks to facilitate impact analysis in the case of any requirement change occurs.

#### *3.3.1. Perform DSM Partitioning*

According to Steward,[27] DSM partitioning is an algorithmic process of finding the blocks and ordering them such that all the predecessors of a block appear somewhere before that block. A block is the largest set of interdependent system elements involved in the iteration cycle. In our work—GART, DSM partitioning is to group the coupled system elements (goals and use cases) into blocks that can help project managers plan the successive project tasks and analyze the impacts of requirements changes.

Figure 8 shows the DSM partition result of our meeting schedule system, in which the traceability links are grouped into five blocks. Each block includes all the coupled goals and use cases that bidirectionally influence each other owing to the

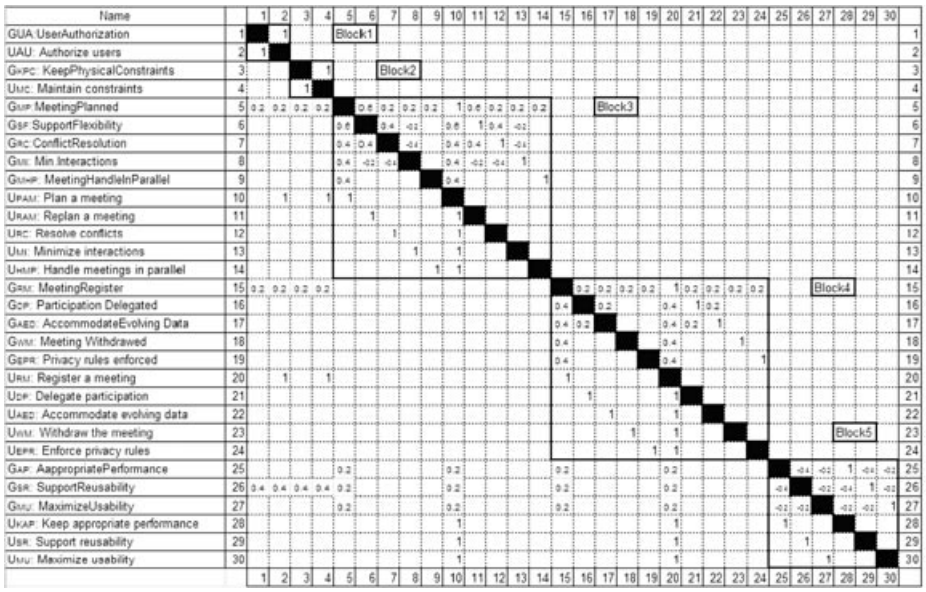



**Figure 8.** DSM partition result of meeting scheduler system.

loop relations between these couples. That is, if an element in one block changes, all the elements in the same block may be affected.

Block 1, including goal $G_{DRH}$ and $U_{AU}$, has links with blocks 3, 4, and 5, which means to change goal $G_{DRH}$ or use case $U_{AU}$ in block 1 may affect the other goals and use cases in blocks 3, 4, and 5. Because use case $U_{AU}$ is included by use cases $U_{PAM}$ and $U_{RM}$, by changing $U_{AU}$ may affect use cases $U_{PAM}$ and $U_{RM}$, as well as those use cases that extend $U_{PAM}$ and $U_{RM}$, and goals satisfied by those use cases.

Block 2, including goal $G_{KPC}$ and use case $U_{MC}$, has the same links as goal $G_{DRH}$ and $U_{AU}$ in block 1 to blocks 3, 4, and 5. Since use case $U_{AU}$ is also included by use cases $U_{PAM}$ and $U_{RM}$, by changing $U_{MC}$ may affect the same goals and use cases as changing the use case $U_{AU}$ in block 1.

The goals and use cases in blocks 3 and 4 can be viewed as goals and use cases for actor *meeting initiator* and *meeting participant*, respectively. There is no link between the elements in block 3 and block 4, hence changes in block 3 will not affect elements in block 4, and not being affected by any changes in block 3. Each block can be divided into four submatrices (see block 4 in Figure 8), which shows the *G2U*, *U2U*, *U2G*, and *G2G* relations between goals and use cases in each block.

Block 5 includes goals $G_{AP}$, $G_{SR}$, and $G_{MU}$, which are soft, system specific, and nonfunctional, and use cases $U_{KAP}$, $U_{SR}$, and $U_{MU}$, which are extension use cases with respect to use cases $U_{PAM}$ and $U_{RM}$. Changes in these goals and use cases in block 5 will not affect goals and use cases in blocks 1–4 since there is no link from the elements in block 5 to the elements in blocks from 1 to 4.

### *3.3.2. Derive Traceability Tree*

Representing the system elements in a tree structure facilitates impact analysis of changes. From the DSM partition result, we can represent the traceability trees in terms of blocks or system elements. Figure 9 represents the DSM blocks in tree structure, which provides a higher level view to understand the relations among DSM blocks. Blocks 1 and 2 have no traceability relations between each other but have the traceability relation to blocks 3, 4, and 5, respectively. Blocks 3 and 4 have the traceability relations to block 5 but have no relation between each other. Block 5 has no traceability relation to other blocks.

Figure 10 shows the traceability tree *T1* whose root is goal $G_{DRH}$ and contains the use case $U_{AU}$ in block 1 and related goals and use cases in blocks 3, 4, and 5. From the tree structure, it is easy to identify the relations between elements in blocks. If the goal $G_{DRH}$ changes, the related goals and use cases can be traversed and regarded as affected work products. $G_{DRH}$ traverses the use case *UAU* in block 1 and related goals $G_{MP}$, $G_{MHP}$, $G_{MI}$, $G_{RC}$, and $G_{SF}$ ,which further discover the use cases $U_{PAM}$, $U_{HMP}$, $U_{MI}$, $U_{RC}$, and $U_{RAM}$ in block 3. In block 4, $G_{DRH}$ traverses the related goals $G_{EPR}$, $G_{WM}$, $G_{AED}$, $G_{DP}$, and $G_{RM}$, which further discover the use cases $U_{EPR}$, $U_{WM}$, $U_{AED}$, $U_{DP}$, and $U_{RM}$. In block 5, $G_{MP}$ also traverses the related goals $G_{MU}$, $G_{SR}$, and $G_{AP}$, which further discover the use cases $U_{MU}$, $U_{SR}$, and $U_{KAP}$. DSM partition blocks together with traceability trees provide a visual way for project managers to organize project tasks, facilitate team communication, and perform change impact analysis.

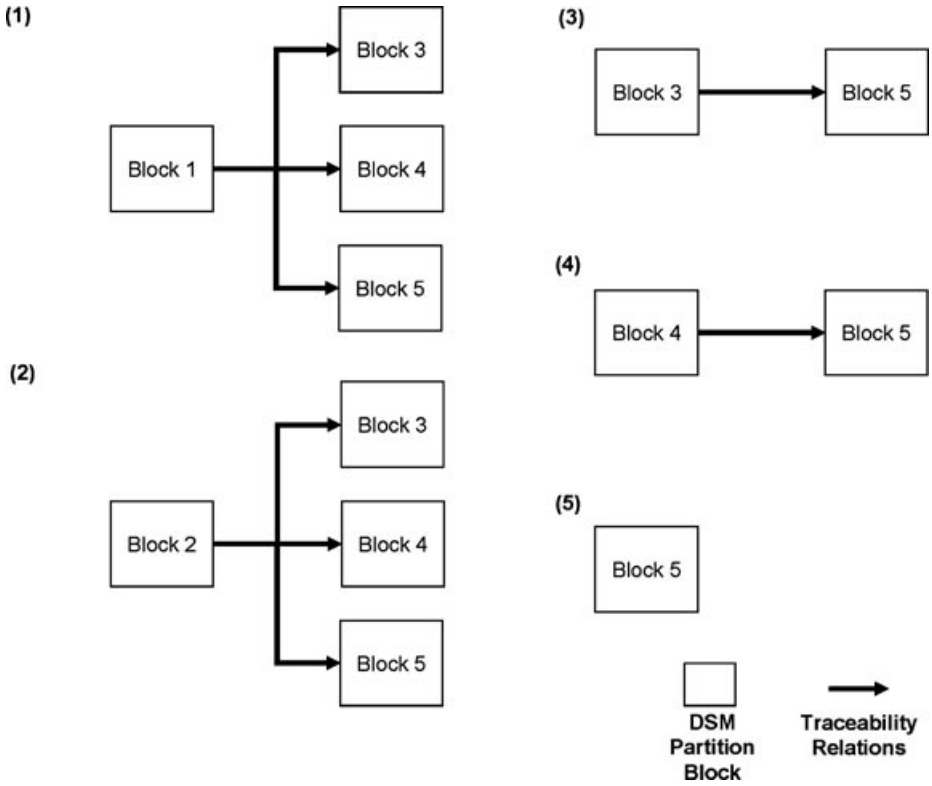



**Figure 9.** Tree representation of DSM blocks.

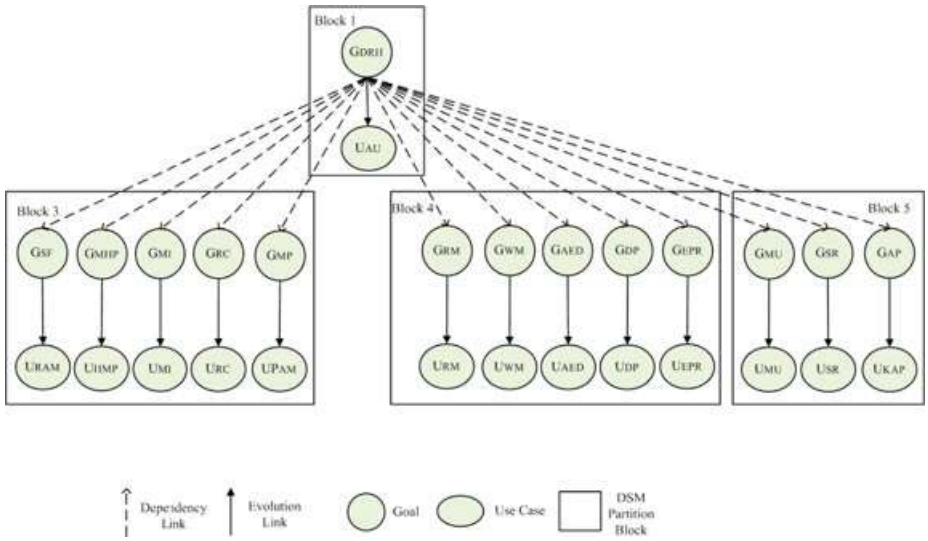



**Figure 10.** Traceability tree T1 of goals and use cases.

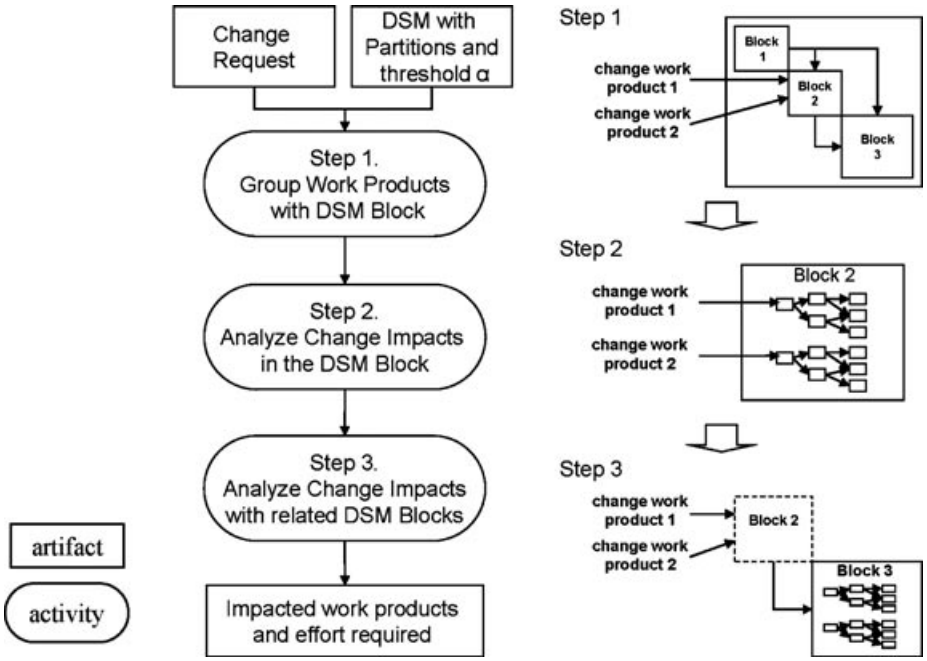



**Figure 11.** Change impact analysis.

### 3.4. Change Impact Analysis

To successfully manage requirements change, change impacts should be correctly analyzed to determine what would be modified for the changes and to avoid the unforeseen ripple effects that frequently result in failures of a project.

When a user requests for a change, the traceability relations between work products can be utilized to analyze the affected work products to implement the changes. The proposed change impact analysis approach to analyzing the affected work products and the effort required to make the changes (see Figure 11).

- Step 1, the traversed change request with change items and the changed work products are grouped based on the DSM blocks where they reside.
- Step 2, the effect of the grouped change work products in the DSM blocks are analyzed to trace the impacted work products of each change work product and get the impact value by summed the results of the multiplication of each impacted use case's use case points and the normalization value of the evolution, dependency and satisfaction links between goals and use cases in a rippled way.
- Step 3, the effects of the changes to related DSM blocks are analyzed, and the total affected work products, system size and effort required to make the changes are generated by utilizing use case point analysis method.

In order to show the practicability of our method with different conditions of requirements change, three changes, *Change A*, *B* and *C* are proposed and illustrated below.

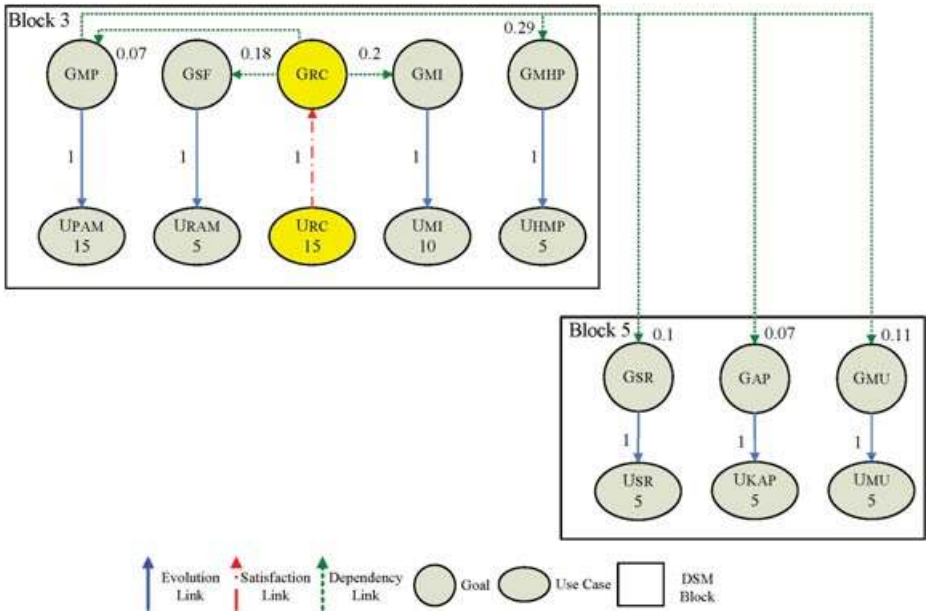



**Figure 12.** Traversal result to change A.

### *3.4.1. Change A: Modify an Existing Requirement*

We take *Change A: Modify an existing requirement* $G_{RC}$ and $U_{RC}$ to support conflicts resolution with additional knowledge, as an example in meeting scheduler system to illustrate how change impact analysis together with use case point analysis can be applied to analyze change impacts.

Figure 12 shows the ripple traversal results of *Change A*. Starting with $G_{RC}$ in Block 3, there are three dependency links from $G_{RC}$ to $G_{SF}$ , $G_{MI}$ and $G_{MP}$, which will lead us to $\{U_{RAM}, U_{MI}$ and $U_{PAM}\}$, respectively. By following the links originating from $G_{MP}$, $G_{MHP}$ and $U_{HMP}$ can be further reached in block 3. Again, by tracing the dependency links from $G_{MP}$ to block 5, three sets of nodes can be found: goals $\{G_{SR}, G_{AP}, G_{MU}\}$, and use cases $\{U_{SR}, U_{KAP}, U_{MU}\}$. Figure 12 also shows two important information: (1) the result after normalizing the dependency links between goals, for example 0.18 between $G_{RC}$ and $G_{SF}$; and (2) the use case points of each affected use case, such as 15 points for $U_{PAM}$.

Figure 13 presents the detail specification of use case $U_{RC}$. The original use case transactions, T1–T8, describe the scenario to solve the date conflict problem. The new added use case transactions, T9–T12, describe the scenario to solve the problem when no allowable location available. We define the change impact ratio (CIR) of a changed use case as

$$\text{CIR} = \frac{\text{Transaction Added} + \text{Modified}}{\text{Base Use Case Transaction}} \tag{1}$$

| **Use Case Name** | Use Case $U_{RC}$: Resolve Conflicts |
|---|---|
| **Actors** | Meeting Initiator and Meeting Participant |
| **Preconditions** | A strong conflict occurs while generating a meeting, that is, a date within the date range belongs to at least one exclusive set or **all proposed locations can't meet the equipment requirement**. |
| **Post-conditions** | Support conflict resolution according to resolution policies stated by the client. |
| **Basic Flows** | The system will notify the initiator, and ask him:<br>• to notify a participant to remove a date from his exclusive set:<br>T1. The system indicates the participant whose exclusive set includes a date with highest flexibility level.<br>T2. The initiator notifies the participant to removes the date from his exclusive set. After the participant has removed the date, the initiator then re-starts a new round of meeting scheduling<br>• to propose a participant with low importance level to withdraw from the meeting:<br>T3. The system indicates the participant with the lowest important level.<br>T4. The initiator notifies the participant to withdraw from the meeting, and re-starts a new round of meeting scheduling.<br>• to extend a date range:<br>T5. The system asks the initiator to input a new date range.<br>T6. The initiator inputs a new date range, and asks the participants to input their new exclusive sets and preference sets. After all participants have input the date, the initiator re-starts a new round of meeting scheduling.<br>• to propose some participant to add some new dates to their preference set.<br>T7. The system indicates the participant whose preference set includes dates with highest flexibility level.<br>T8. The initiator notifies the participant to add some new dates to their preference set. After the participant has added some new dates, the initiator then re-starts a new round of meeting scheduling<br>**If all proposed location can't meet the equipment requirement (i.e., rooms of the locations can't support all equipment that participants needs) while making a meeting schedule.**<br>• **The initiator propose other locations**<br>**T9. The system informs the initiator that no allowable location available.**<br>**T10. The initiator may inputs some new locations and then re-starts a new round of meeting scheduling.**<br>• **The initiator ask participant to remove equipment requirements**<br>**T11. The initiator notifies the participant to removes equipment requirements.**<br>**T12. After the participant has removed the equipment requirements, the initiator then re-starts a new round of meeting scheduling.** |

**Figure 13.** Use case specification of $U_{RC}$.

Referring to Change A, we can obtain its CIR as $4/8 = 0.5$.

The impact metric V of a change with the changed use cases $U_i$ is defined in Equation 2, where $UCP_j$ are the use case points of impacted use cases $U_j$ of $U_i$ in the same block, and $Links_{ij}$ are all the traversed links from $U_i$ to $U_j$. $UCP_k$ are the use case points of impacted use cases $U_k$ of $U_i$ in other blocks, and $Links_{ik}$ are all

the traversed links from $U_i$ to $U_k$. The fuzzy threshold $T2$ is introduced to provide the flexibility that the traceability links from $U_i$ to $U_k$ can be filtered out if

$$\prod \text{Links}_{ik}(\text{normalized value of Links}_{ik}) < \text{T2}.$$

$$V = \text{CIR} \times \left( \text{UCP}_i + \sum_{j=1}^{n} \prod_{\text{Links}_{ij}} (\text{normalized value of Links}_{ij}) \times \text{UCP}_j + \sum_{k=1}^{m} \prod_{\text{Links}_{ik}} (\text{normalized value of Links}_{ik}) \times \text{UCP}_k \right) \quad (2)$$

where $n$ is the number of impacted use cases in the same block of $U_i$, $m$ is the number of impacted use cases in other blocks, and $\prod_{\text{Links}_{ik}}(\text{normalized value of Links}_{ik}) \geq T2 = 0.002$.

Impact metric V of *Change A* can be derived as follows:

$$\text{V} = 0.5 \times (15 + 0.2 \times 1 \times 10 + 0.18 \times 1 \times 5 + 0.07 \times 1 \times 15 + 0.07 \times 0.29 \times 1 \times 5 + 0.07 \times (0.1 \times 1 \times 5 + 0.07 \times 1 \times 5 + 0.11 \times 1 \times 5)) = 9.57$$

To obtain the value of V, UCP of $U_{RC}$ is added with the result of the sum of the use case points of the normalized value of the traversed links from $G_{RC}$. For example, the impacted UCP of the use case $U_{MI}$ is added from the result of $0.2 \times 1 \times 10$, where 0.2 is the normalized value of the dependency link from $G_{RC}$ to $G_{MI}$, 1 is the value of evolution link from $G_{MI}$ to $U_{MI}$ and 10 is the UCP of use case $U_{MI}$, respectively.

The technical complexity factor (TCF) and the environment factor (EF) of the meeting scheduler system are set to 0.85 and 0.8 based on the use case point approach to deriving impacted UCP, respectively.

$$\text{Impacted Use Case Points (UCP) of } \textit{Change A} = \text{TCF} \times \text{EF} \times \text{V of } \textit{Change A} \quad (3)$$

We then calculate the impacted use case points by Equation 3 after the change as follows:

$$0.85 \times 0.8 \times 9.57 = 6.5 \text{ UCP}$$

Suppose we use a factor of 20 person-hours per UCP, then the effort required to implement *Change A* can be obtained below:

$$\text{Effort required to implement } \textit{Change A} = 6.5 \times 20 = 130 \text{ person-hours}$$

To further verify our estimation, a second estimate effort in person-months based on COCOMO2[28] is also adopted as follows:

$$\textit{Code Size} = 9.57 \text{ Unadjusted UCP} \times 0.8 \times 50 = 383 \text{ LOC}$$

Person-month required to implement *Change A*

$$PM_{NS} = A \times \text{Size}^{E} \times \prod_{i=1}^{n} \text{EM}_i$$

$$= 2.94 \times (0.383)^{1.15} \times 0.85 = 0.83 \text{ person-month}$$

$$= 132.8 \text{ person-hours}$$

The way the code size is calculated is followed by the conversion ratios given in Ref. 29. The conversion ratios from unadjusted UCP to IFPUG function points and from IFPUG function points to java statement per function point are 0.8 and 50, respectively. As an average project, the effort multipliers of meeting scheduler system are all equal to 1.0 except the ACAP is 0.85 (analyst capability is high) to comply with the setting of UCP environment factors. $A$ and $E$ are set to 2.94 and 1.15, respectively. The result to implement the change is 131.2 person-hours by taking 1 person-month = 20 person-days. As a result, the two estimates end up with a 2.8 person-hours difference.

### *3.4.2. Change B: Add a New Requirement*

We take *Change B*: *Add a new requirement $G_{EN}$ and $U_{EN}$ to notify meeting initiator and participants by e-mail*, as an example in meeting scheduler system to illustrate how to analyze change impacts of a new requirement (see Figure 14 for the traversal results of *Change B*). Since *Change B* is a new requirement, the CIR of *Change B* is 1 and the UCP of $U_{EN}$ is 5 obtained by analyzing its use case specification in Figure 15.

To obtain the value of V, UCP of $U_{EN}$ is added with the result of the sum of the use case points of the normalized value of the traversed links from $G_{EN}$. For example, the impacted UCP of the use case $U_{RAM}$ is added from the result of $0.07 \times 0.15 \times 1 \times 5$, where 0.07 is the normalized value of the dependency link from $G_{EN}$ to $G_{MP}$, 0.15 is the normalized value of the dependency link from $G_{MP}$ to $G_{SF}$, 1 is the value of evolution link from $G_{SF}$ to $U_{RAM}$, and 5 is the UCP of use case $U_{RAM}$, respectively.

Impact metric V of *Change B* can be derived as follows:

$$\text{V} = 1 \times (5 + 0.07 \times 1 \times 15 + 0.07 \times (0.15 \times 1 \times 5 + 0.12 \times 1 \times 15 + 0.13$$

$$\times 1 \times 10 + 0.15 \times 1 \times 5) + 0.07 \times (0.1 \times 1 \times 5 + 0.07 \times 1 \times 5 + 0.11$$

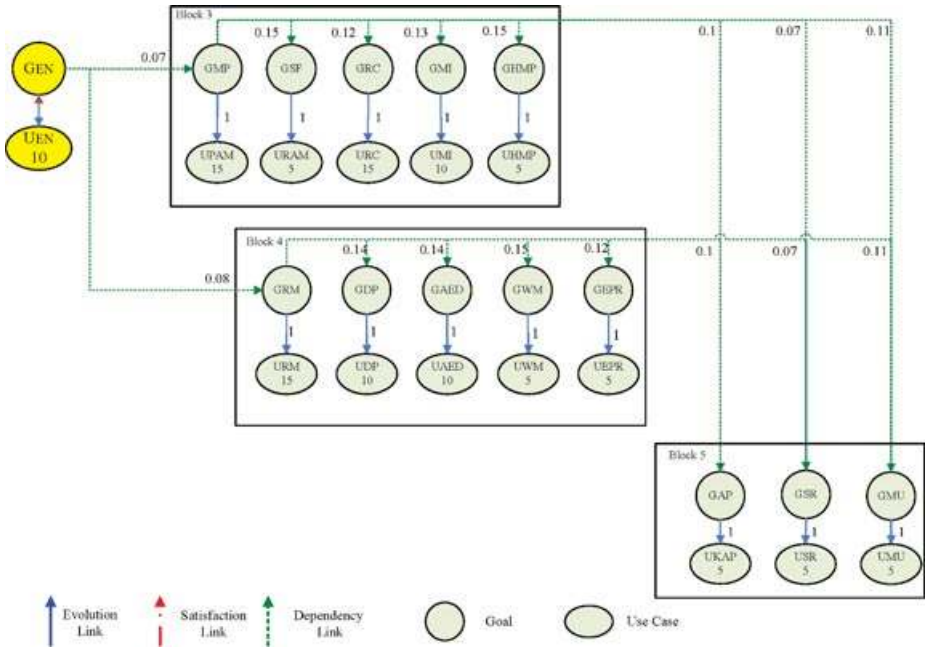



**Figure 14.** Ripple traversal to change B.

| Use Case Name | Use Case UEN: Email Notification |
|---|---|
| Actors | Meeting Initiator, Meeting Participant and Email Service |
| Preconditions | 1. The initiator finishes meeting planning<br>2. When the meeting status changes, the system notifies meeting initiator and participant automatically |
| Post-conditions | The notification mail was sent to relevant stakeholders. |
| Basic Flows | T1:<br>1. The initiator clicks submit button to notify meeting participant or when the meeting status changes<br>2. The system gets the notification content, generates the email text, and sends the notification mail to relevant stakeholder. |
| Alternative Flows | 2a. If the email content is empty or there is no receiver, system should cancel the sending process. |
| Special Requirements | 2s-1. System can encode/decode the URL so that user can login to system directly |

**Figure 15.** Ripple traversal to change B.

$$\times 1 \times 5) + 0.08 \times 1 \times 15 + 0.08 \times (0.14 \times 1 \times 10 + 0.14 \times 1 \times 10 + 0.15$$

$$\times 1 \times 5 + 0.12 \times 1 \times 5) + 0.08 \times (0.1 \times 1 \times 5 + 0.07 \times 1 \times 5 + 0.11$$

$$\times 1 \times 5)) = 8.11$$

We then calculate the impacted use case points by Equation 3 after the change as follows:

$$0.85 \times 0.8 \times 8.11 = 5.5 \text{ UCP.}$$

The effort required to implement *Change B* can be obtained by using a factor of 20 person-hours per UCP below:

$$\text{Effort required to implement } \textit{Change B}$$

$$= 5.5 \times 20 = 110 \text{ person-hours}$$

Based on COCOMO2, a second estimate effort in person-months is also adopted as follows:

$$\textit{Code Size} = 8.11 \text{ Unadjusted UCP} \times 0.8 \times 50 = 324 \text{ LOC}$$

Person-month required to implement *Change B*

$$PM_{NS} = A \times \text{Size}^{\text{E}} \times \prod_{i=1}^{n} \text{EM}_i = 2.94 \times (0.324)^{1.15} \times 0.85$$

$$= 0.68 \text{ person-month} = 108.8 \text{ person-hours}$$

The result to implement the change is 108.8 person-hours by taking 1 person-month = 20 person-days. As a result, the two estimates of *Change B* end up with only a 1.2 person-hours difference.

### *3.4.3. Change C: Modify Two Requirements*

We take *Change C: Modify two requirements $G_{AED}$ and $U_{AED}$ to support the meeting customization, and $G_{DP}$ and $U_{DP}$ to support partial meeting participation*. Figure 16 show the traversal results of Change C. Referring to Change C on the use case specifications of $U_{DP}$ and $U_{AED}$ (see Figures 17 and 18), both the CIR of $U_{DP}$ and $U_{AED}$ are 1/4 = 0.25.

To obtain the value of V, UCP of $U_{DP}$ and $U_{AED}$ are added with the result of the sum of the use case points of the normalized value of the traversed links from $G_{DP}$ and $G_{AED}$. For example, the impacted UCP of the use case URM is added from the result of 0.08 × 1 × 15, where 0.08 is the normalized value of the

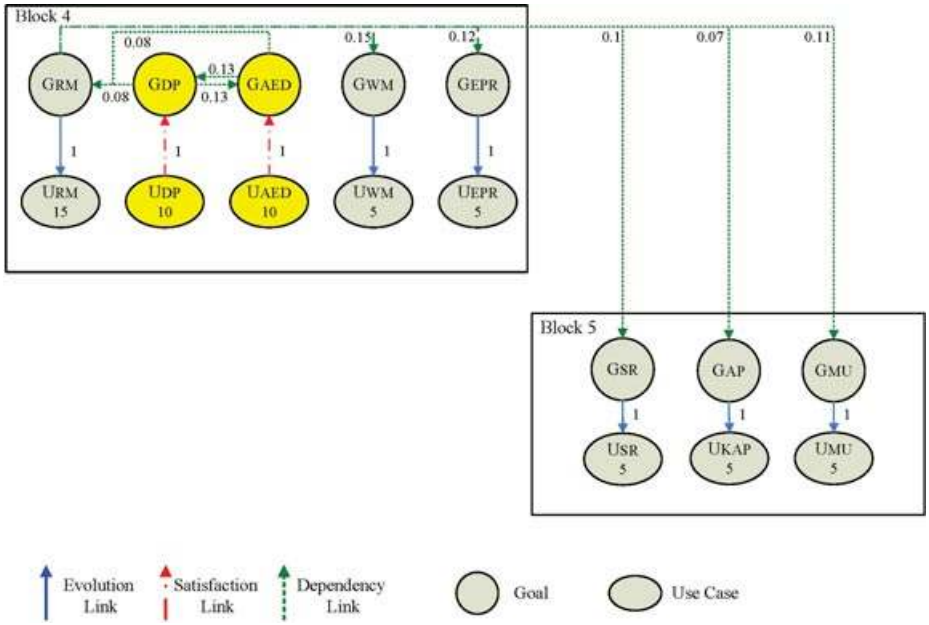



**Figure 16.** Ripple traversal to change C.

dependency link from $G_{DP}$ to $G_{RM}$, 1 is the value of evolution link from $G_{RM}$ to $U_{RM}$, and 15 is the UCP of use case $U_{RM}$, respectively. The traceability links from $G_{DP} \rightarrow G_{AED} \rightarrow G_{RM} \rightarrow G_{SR}$, $G_{AP}$ and $G_{MU}$, and from $G_{AED} \rightarrow G_{GDP} \rightarrow G_{RM} \rightarrow G_{SR}$, $G_{AP}$, and $G_{MU}$ can be filtered out by Equation 2, since the results of

$$\prod_{\text{Links}_{ik}} (\text{normalized value of Links}_{ik}) < T2.$$

Impact metric V of Change C can be derived as follows:

$$\begin{aligned} \text{V} = {} & 0.25 \times (10 + 0.08 \times 1 \times 15 + 0.08 \times (0.15 \times 1 \times 5 + 0.12 \times 1 \times 5 + 0.1 \\ & \times 1 \times 5 + 0.07 \times 1 \times 5 + 0.11 \times 1 \times 5) + 0.13 \times 1 \times 10 + 0.13 \times 0.08 \\ & \times (1 \times 15 + 0.15 \times 1 \times 5 + 0.12 \times 1 \times 5)) + 0.25 \times (10 + 0.08 \times 1 \times 15 \\ & + 0.08 \times (0.15 \times 1 \times 5 + 0.12 \times 1 \times 5 + 0.1 \times 1 \times 5 + 0.07 \times 1 \times 5 \\ & + 0.11 \times 1 \times 5) + 0.13 \times 1 \times 10 + 0.13 \times 0.08 \times (1 \times 15 + 0.15 \times 1 \times 5 \\ & + 0.12 \times 1 \times 5)) = 6.44 \end{aligned}$$

| Use Case Name | Use Case $U_{DP}$: Delegate Participation |
|---|---|
| Description | participation through delegation - a participant may ask another person to represent him/her at the meeting; |
| Primary Actor | Meeting Participant |
| Preconditions | 1. The meeting participant can not attend the meeting and wants to delegate his participation to the other person<br>**2. The meeting participant can attend the meeting partially and delegate his participation to the other person** |
| Post-conditions | The participation has been delegated successfully |
| Basic Flows | T1: The participant selects delegation and the system displays delegation form **(The delegation form should support partial delegation that the participant can attend the meeting partially by change his attendance time)**<br>T2: The participant fills the delegation form and submits it. The system notifies the initiator about the delegation<br>T3: The initiator decides to accept the delegation and the system notifies the participant and the person being delegated |
| Alternative Flows | T4: 3a. The initiator can not accept the delegation<br>3b. The system notifies the participant to change his data |
| Special Requirements | |

**Figure 17.** Use case specification of $U_{DP}$.

We then calculate the impacted use case points by Equation 3 after the change as follows:

$$0.85 \times 0.8 \times 6.44 = 4.38 \text{ UCP}$$

Suppose we use a factor of 20 person-hours per UCP, then the effort required to implement *Change C* can be obtained below:

$$\text{Effort required to implement Change B}$$

$$= 4.38 \times 20 = 87.6 \text{ person-hours}$$

A second estimate effort in person-months based on COCOMO2 is adopted as follows:

$$\textit{Code Size} = 6.45 \times 0.8 \times 50 = 258 \text{ LOC}$$

| Use Case Name | Use Case $U_{AED}$: Accommodate Evolving Data |
|---|---|
| Description | The system should be flexible enough to accommodate evolving data - e.g., the sets of concerned participants may be varying, the address at which a participant can be reached may be varying, etc. |
| Primary Actor | Meeting participants |
| Preconditions | The participant has provided his/her preference and exclusion set |
| Post-conditions | The meeting date and location are determined |
| Basic Flows | Before meeting date & location are determined, the participant can change his data such as his preference set and exclusion set<br>T1: The system can handle explicit dependencies between meeting date and meeting location when user modify meeting date and meeting location;<br>T2: The system can handle explicit priorities among dates in preference sets when user changes the priorities.<br>T3: The system can handle variations in date formats, address formats and interface language<br>T4: The system can handle the varying sets of concerned participants and type of participant (same or different degrees of importance) ,<br>**T5: The system should provide customized solutions according to variations in type of meeting (professional or private, regular or occasional) and type of meeting location (fixed, variable).** |
| Alternative Flows | |
| Special Requirements | 1. The meeting date and location should be notified to the participants 1 week before the meeting start<br>2. After the meeting date and location are determined, the participant can not change his preference set and exclusion set. |

**Figure 18.** Use case specification of $U_{AED}$.

Person-month required to implement *Change C*

$$PM_{NS} = A \times \text{Size}^{\text{E}} \times \prod_{i=1}^{n} \text{EM}_i$$

$$= 2.94 \times (0.258)^{1.15} \times 0.85 = 0.53 \text{ person-month}$$

$$= 84.8 \text{ person-hours}$$

The result to implement the change is 84.8 person-hours by taking 1 person-month = 20 person-days. As a result, the two estimates of *Change C* end up with a 2.8 person-hours difference.

## 4. RELATED WORK

Work in a number of fields has made its mark on our research. Our approach has drawn upon several ideas from requirements traceability techniques and change impact analysis methods.

### 4.1. Requirements Traceability

Several studies[11,30–32] have shown the importance and issues of using the requirements traceability techniques to trace the requirements with other work products, such as design, source code, and test cases, in the software development life-cycle.

Gotel and Finkelstein[30] investigated and reported the requirements traceability problems and recommended several solutions. They observed a lack of "pre-RS traceability," meaning that no traceability information was available before inclusion in the requirements specification. D'omges and Pohl[12] proposed a framework of adaptable traceability environments to support the definition and usage of project-specific trace data and strategies. The organizational learning about project-specific requirements traceability should be empowered to guide stakeholder in trace capture and usage, and support continuous process improvement.

Spanoudakis et al.[4] developed a rule-based approach to automatically generate four traceability relations between requirement statement documents, use cases documents, and analysis object models of a software system by using two different traceability rules, the requirements-to-object-model and interrequirement rules.

Egyed proposed a scenario-driven approach[5] to generating and validating trace dependencies among model elements that are related to code. A test scenario or usage scenario is tested to generate observed traces for further trace analysis with other hypothesized traces. The new trace dependencies among model elements and code are then yielded and validated to solve the inconsistency and incompleteness.

To address the human source issue of requirement traceability, Gotel and Finkelsten[7] presented contribution structure to locate people with their contributed artifacts in the contribution format. The social roles, role relations, and commitments can be further inferred to form the personnel-based requirements traceability.

Cleland-Huang et al.[9] developed an event-based traceability (EBT) approach, based on event notification, to tracing and maintaining the artifacts and their related links during the software life cycle. The EBT's architecture has the following three main parts: Requirements Manager handles requirements evolution with identified change events and triggers the events by publishing an event message when a change occurs. Event Server manages subscriptions and receives event messages from the requirement manager. It then customizes event notifications into a specific update directive, and forwards it to the subscriber manager. Subscriber Manager receives and resolves event notifications and restores an artifact and related traceability links to a current state if necessary. Cleland-Huang et al.[8] also presented a goal-centric approach to manage nonfunction requirements in four steps. First, nonfunctional requirements are modeled by a soft goal interdependency graph (SIG). Second,

when the changes occur, the trace links are dynamically generated using probabilistic network model and filtered by users to remove the irrelevant ones. Third, the affected SIG elements are analyzed and evaluated to determine whether other goals are affected and still satisfied after the changes. Finally, the decision on whether to implement changes is made, and risk mitigation strategies are identified before performing the changes.

These requirements traceability work can be characterized with criteria listed below, which are then served as a basis for making the comparison for the pros and cons of the above-mentioned work (see Table II).

- *Traceability Techniques*: the traceability technique/method proposed or adopted,
- *Requirements Types*: the requirements that the approaches can handle, in particular, whether the approaches can handle nonfunctional requirements,
- *Vertical Traceability*: whether the approaches can manage the traceability relation from "a requirements to its derived requirements and allocation to functions, interface, objects, people, process and work products,"[33]
- *Horizontal Traceability*: whether the approaches handle relations across requirements or interfaces,[33] and
- *Traceability Visualization*: how the traceability relations are presented.

### 4.2. Change Impact Analysis

Managing changes to a software system is critical to the success of the system, since software undergoes change during the whole life cycle of software. Change impact analysis is defined as "identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change".[34] To successfully manage requirements changes, change impacts should be correctly analyzed to determine what should be modified for the changes and to avoid the unforeseen ripple effects that frequently result in failures.

Bohner[35] has proposed a basic software change impact analysis process to iteratively discover impacted "software life-cycle objects (SLOs)". The fan-in/out relations of the SLOs are identified in the connectivity matrix. The reachability matrix using the notion of distance indicators can then be used to indicate potential impacts to a SLO. The semantics of the relations are employed to increase the accuracy of finding the impacted elements. Three key challenges of impact analysis: huge information sources, semantics of software artifacts relationships, and work product dependency analysis methods are discussed, and the change effect on the middleware and COTS components are addressed to solve the change problem through interoperability dependency relationships.

In Ref. 36 and 37, an impact analysis method was presented to evaluate requirements change based on traceability relations between the work products. For each change, the impacted work products are traced to compute the impact metrics, the changes are then categorized into groups (compatibility class) from low to high based on the computed impact values using the compatibility relation.

To analyze the change impact on UML models, Briand et al.[38] has proposed an impact analysis approach to analyze and prioritize the impacted model elements in the UML models using the impact analysis rules and distance measure for each

**Table II.** Comparison of work on requirements traceability.

| | Our Work | Cleland-Huang et al. | Egyed et al. | Zisman et al. | Gotel and Finkelstein |
|---|---|---|---|---|---|
| Traceability Techniques | Goal-Driven and DSM | Event-Based and Probability Inference Model | Scenario-Based | Rule-Based | Contribution Structures |
| Requirements Types | GDUC can handle functional and non-functional requirements | Use Soft goal Interdependency Graph (SIG) to model non-functional requirements | Non-functional requirements can be addressed | Non-functional requirements are not explicitly addressed | Requirements types are not explicitly specified |
| Vertical Traceability | Support by evolution and satisfaction links | From goals to low-level artifacts | Requirements/model elements related to code can be traced | From requirements/ use cases to object models | Extend Artifact-based traceability with Personal-based traceability |
| Horizontal Traceability | Strength and direction of dependency link are supported | Use SIG for goals dependencies | Trace dependencies can be generated among any elements that relate to code | From requirements to requirements / requirements to use cases | Not explicitly addressed |
| Traceability Visualization | DSM blocks and traceability tree | Table and matrix view | Footprint graph is proposed | Table view | Contribution format |
| Weakness | User evaluation is still required for trace relation analysis | The recall and precision rates of the retrieved links are the major concern | Complete traceability coverage may not be provided | The recall and precision rates of the generated trace links are the major concern | The work involved to establish and utilize the contribution structures |
| Strength | Systematic way to identify trace relations and use DSM to partition and visualize the trace relations | Dynamically retrieve the links to reduce the overhead of traceability establishment | Trace links can be generated by analyzing the scenario, code and model elements | Automatic generation of traceability relations using rule inference | The traceability to human sources of requirements is provided |

change. The impact analysis of changed model elements can be automated using the OCL rules defined for each change type. In their approach, very detailed UML model information should be used and the method to perform automatic impact analysis is not clearly stated.

In Ref. 39, a traceability-based approach to analyze change impacts between requirements, test cases, design (packages and classes), and code (methods) is presented. The traceability model and code parser with structure relationships in C++ are illustrated to support the top-down and button-up analysis of change impacted artifacts. However, how to establish the trace links between software artifacts is not clearly addressed and the required effort and schedule for a change are not analyzed.

Based on use case map (UCM) technique, Hassine et al.[40] use requirement dependencies with forwarded slicing algorithm to analyze ripple effect of the changes at the requirement level. The requirement dependencies include scenario dependencies and component-based dependencies. The scenario dependencies are classified in three types: functional, containment, and temporal. The component dependencies include forward and backward dependence to identify the relations between components using the scenario execution sequence.

Lock and Kotonya[41] provide an integrated approach that integrates traceability extraction methods to determine impactables affected by the change in the impact propagation structures using propagation probability. The traceability analysis then be conducted to produce a single impact propagation structure using vertical composition, lateral composition, duplication resolution, and probability decay. In their approach, how to solve the loop problem is not addressed in the analysis process.

To predict software change impact, a study had conducted to investigate the correlations between standard diagrams (UML diagrams and dataflow diagram) and the change request characteristics (type, scope, estimated, and actual effort).[42] The analysis results of three change requests are presented to show the relationship between actual implementation effort and impacted diagrams. However, it is lack the systematic analysis approaches to analyze the affected software artifacts and to predict the required effort.

In Ref. 43, a model called Architecture Rationale and Element Linkage (AREL) is proposed to model the relations between architecture elements and decisions. The Bayesian Belief Network (BBN) is used to capture the probabilistic relations between the elements in the AREL model and to predict the change impacts when the requirement changes occur.

In Ref. 44, a framework for comparison of the impact analysis (IA) approaches is proposed. It includes three parts: "IA application" to examine how the approach is used to perform IA, "IA parts" to examine the internal elements and methods used by the IA approach and "IA effectiveness" to examine how well and accuracy the approach does. The IA approaches they compared include the dependence analysis techniques among program entities. We summarize the comparison between these change impact analysis approaches with a list of criteria in Table III. The detailed descriptions of these criteria are as follows:

- Impact analysis methods: What technique the impact analysis approaches proposed or used to performed change analysis?

**Table III.** Comparison of researches on change impact analysis.

| | Our Work | Shawn A. Bohner | Simon Lock and Gerald Kotonya | L.C. Briand et al. | Jameleddine Hassine et al. | James S. ONeal et al. |
|---|---|---|---|---|---|---|
| Impact analysis methods | Impact analysis using DSM and traceability relations | Identify impacts by extending semantic in trace relations | Integrate traditional impact analysis approaches with experience based techniques | UML model-based approach (Impact analysis rules and distance measure) | Use case maps (UCM) forward slicing | Trace-based Impact Analysis Methodology (TIAM) |
| Traceability Technique | Goal-Driven and DSM | Connectivity matrix and reachability matrix with distance indicators | Impact propagation structures with probability values | Not addressed | UCM requirements dependencies at scenario and component level | Work Product Requirements Trace Model |
| Impact Analysis Results | Impacted work product tree and impact metric | Directed and indirected impacted SLOs | Impactables in Impact propagation structures | Impacted UML model elements | Impacted UCM elements | Ordered compatibility classes of requirements changes |
| Effort or impact degrees predicted | CIAM can estimate the effort to implement changes | Not addressed | Not addressed | Not addressed | Not addressed | Impact metric value for each change can be generated |
| Weakness | The use case point of each use case are needed | How to analyze the change impact is not explicitly addressed | The work involved to analyze changes by integrating various techniques | Required detailed UML model information | Impact analysis is only performed at the UCM specification level | Only requirements can be changed |
| Strength | Automatic impact analysis for potential impacted work products and effort required | Extend semantic relationship between SLOs, Extend reachability matrix with the distance measures | Various impact analysis techniques are utilized to perform impact analysis | Analyze change impact automatically by using the change taxonomy and impact analysis rules | Support early impact analysis of requirements change | The impact metric and compatibility classes of changes are analyzed |

- Traceability Technique: What the traceability technique/method they proposed or used.
- Impact Analysis Results: How the impact analysis results were presented.
- Effort or impact degrees predicted: Does the approach predict the required effort or impact degrees of the requirements changes?

## 5. CONCLUSION

This work proposes a goal-driven approach to managing requirements traceability and analyzing change impacts by fusing the goal-driven and design structure matrix (DSM) techniques. The change impact analysis method identifies potentially impacted work products and compute an impact metric based on DSM blocks. An implementation of the DSM partition and use case points has also been developed to assess the scalability of this work.

The main benefit of the proposed approach is to provide a visual way for project managers to organize project tasks, facilitate team communication, and perform change impact analysis.

## References

1. Jones C. Software change management. IEEE Comput 1996;29(4):80–82.
2. Chrissis MB, Konrad M, Shrum S. CMMI: Guidelines for process integration and product improvement, 2nd edition. Reading, MA: Addison-Wesley; 2006.
3. Rovegard P, Angelis L, Wohlin C. An empirical study on views of importance of change impact analysis issues. IEEE Trans Softw Eng 2008;34(4):516–530.
4. Spanoudakis G, Zisman A, Pe'rez-Minana E, Krause P. Rule-based generation of requirements traceability relations. J Syst Softw 2004;72(2):105–127.
5. Egyed A. A scenario-driven approach to trace dependency analysis. IEEE Trans Softw Eng 2003;29(2):116–132.
6. Antoniol G, Canfora G, Casazza G, Lucia AD, Merlo E. Recovering traceability links between code and documentation. IEEE Trans Softw Eng 2002;28(10):970–983.
7. Gotel O, Finkelstein A. Extended requirements traceability: results of an industrial case study. In: Proc 3rd IEEE Int Symp on Requirements Engineering, Jan 1993. pp 169–178.
8. Cleland-Huang J, Settimi R, BenKhadra O, Berezhanskaya E, Christina S. Goal-centric traceability for managing non-functional requirements. In: Proc Int Conf on Software Engineering, May 2005. pp 362–371.
9. Cleland-Huang J, Chang CK, Christensen M. Event-based traceability for managing evolutionary change. IEEE Trans Softw Eng 2003;29(9):796–810.
10. Ramesh B, Jarke M. Toward reference models for requirements traceability. IEEE Trans Softw Eng 2001;27(1):58–93.
11. Ramesh B, Edwards M. Issues in the development of a requirements traceability model. In: Proc Int Conf on Requirements Engineering, Jan 1993. pp 256–259.
12. D'omges R, Pohl K. Adapting traceability environments to project-specific needs. Commun ACM 1998;41(12):54–62.
13. Lee J, Xue N-L. Analyzing user requirements by use cases: a goal-driven approach. IEEE Softw 1999;16(4):92–101.
14. Lee J, Xue N-L, Kuo J-Y. Structuring requirements specifications with goals. Inf Softw Technol 2001;43(2):121–135.
15. Lee J, Hsu K-H. Modeling software architectures with goals in virtual university environment. Inf Softw Technol 2002;44(6):361–380.
16. Lee J, Fanjiang Y-Y. Modeling imprecise requirements with XML. Inf Softw Technol 2003;45(7):445–460.

17. Lee J, Kuo JY. New approach to requirements trade-off analysis for complex systems. IEEE Trans Knowl Data Eng 1998;10(4):551–562.
18. Steward DV. The design structure system: A method for managing the design of complex systems. IEEE Trans Eng Manage 1981;28:71–74.
19. Eppinger SD, Whitney DE, Smith RP, Gebala DA. A model-based method for organizing tasks in product development. Res Eng Des 1994;6:1–13.
20. Browning TR. Applying the design structure matrix to system decomposition and integration problems: A review and new directions. IEEE Trans Eng Manage 2001;48(3):292–306.
21. Browning TR, Eppinger SD. Modeling impacts of process architecture on cost and schedule risk in product development. IEEE Trans Eng Manage 2002;49(4):428–442.
22. Danilovic M, Browning TR. Managing complex product development projects with design structure matrices and domain mapping matrices. Int J Proj Manage 2007;25(3):300–314.
23. Feather MS, Fickas S, Finkelstein A, van Lamsweerde A. Requirements and specifications exemplars. Autom Softw Eng 1997;4(4):419–438.
24. Karner G. Resource estimation for objectory projects. Objectory Syst 1993.
25. Satty TL. Analytic hierarchy process. New York: McGraw-Hill International; 1980.
26. Garnier R, Taylor J. Discrete mathematics for new technology, 2nd edition. Boca Raton, FL: CRC Press, Taylor and Francis Group; 2001.
27. Steward DV. Partitioning and tearing systems of equations. J Soc Ind Appl Math: Ser B, Numer Anal 1965;2(2):345–365.
28. Boehm BW, Horowitz Clark, Reifer Brown, Madachy Chulani, R, Steece B. Software cost estimation with COCOMO II with CDROM. Singapore: Prentice Hall PTR, 2000.
29. Jones C. Estimating software costs: Bringing realism to estimating. McGraw-Hill Osborne Media, 2007.
30. Gotel OCZ, Finkelstein ACW. An analysis of the requirements traceability problem. In: Proc Int Conf on Requirements Engineering; April 1994. pp 94–101.
31. Jarke M. Requirements tracing. Commun ACM 1998;41(12):32–36.
32. Watkins R, Neal M. Why and how of requirements tracing. IEEE Softw 1994;11(4):104–106.
33. Institute SE. CMMI for development version 1.2, Technical Report CMU/SEI-2006-TR-008, 2006.
34. Bohner SA, Arnold RS. Software change impact analysis. Los Alamitos, CA: IEEE Computer Society Press; 1996.
35. Bohner SA. Software change impacts—an evolving perspective. In Proc Int Conf on Software Maintenance (ICSM'02); 2002. pp 263–272.
36. O'Neal JS. Analyzing the impact of changing software requirements: A traceability-based methodology, Ph.D. dissertation of Louisiana State University, 2003.
37. O'Neal JS, Carver DL. Analyzing the impact of changing requirements. In: Proc IEEE Int Conf on Software Maintenance; November 2001. pp 190–195.
38. Briand L, Labiche Y, OSullivan L, So'wka M. Automated impact analysis of UML models. J Syst Softw 2006;79(3):339–352.
39. Ibrahim S, Munro M. A requirements traceability to support change impact analysis. Asian J Inf Technol 2005;4(4):329–338.
40. Hassine J, Rilling J, Hewitt J. Change impact analysis for requirement evolution using use case maps. In: Proc 2005 Eighth Int Workshop on Principles of Software Evolution; 2005. pp 81–90.
41. Lock S, Kotonya G. An integrated, probabilistic framework for requirement change impact analysis. Australas J Inf Syst 1999;6(2):38–63.
42. Ackermann C, Lindvall M. Understanding change requests to predict software impact. In: Proc 30th Annual IEEE/NASA Software Engineering Workshop; April 2006. pp 66–75.
43. Tang A, Jin Y, Han J, Nicholson A. Predicting change impact in architecture design with bayesian belief networks. In: Proc 5th Working IEEE/IFIP Conf on Software Architecture; 2005. pp 67–76.
44. Arnold RS, Bohner SA. Impact analysis—towards a framework for comparison. In: Proc Conf on Software Maintenance; Sept 1993. pp 292–301.