# An Automatic Generation Method of Traceability Links from Requirement to Design in Business Applications

[1]Soojin Park

***Abstract***

*Requirements traceability link information is the basis for determining whether requirement change requested throughout the software development life cycle should be reflected in the system. Setting up complete requirements traceability links requires considerable effort. However, the commensurate benefits can be obtained in later development or further maintenance phases. For this reason, setting up and managing requirements traceability links in the software development phase are tasks that cause considerable resistance to developers. This study proposes a method for generating requirement traceability links in business applications. The key feature of the proposed method is that the traceability link from the requirements element, which is the basis of the corresponding element to the analysis element, is automatically established at the same time the elements of the analysis model are identified. This can be a way to reduce developer effort while increasing the efficiency of the traceability model. A case study on a Course Registration System demonstrates the feasibility of applying the proposed requirements traceability management method to actual software development.*

***Keywords:*** *Requirement traceability management, Class responsibility assignment, Business application system, Change request management, Software maintenance*

## I. Introduction

Difficulties in software development are derived from invisibility, one of the essences of the software itself [1]. This characteristic of a software product that cannot be seen with the naked eye or touched wit h the hand causes uncertainty for both developers and users. This uncertainty leads to frequent changes i n requirements. According to the survey results in [2], it is evident that around 40 to 90% of the total sys tem development cost is allocated to system changes due to shifts in requirements. Apart from requireme nts changes stemming from the invisibility of software, change requests in requirements due to shifts in t he business landscape are also frequent, particularly within business application system domains. The du ration needed to incorporate changes prompted by these diverse factors into the system is a noteworthy a ttribute of evaluating the quality of a software system [3].

Fig. 1 presents an overview of the process, from initiating a requirement change request to integrating it within the system. A user submits a change request (CR) to alter requirements A to A'. The Change Co ntrol Board (CCB) assesses the submitted CR and determines whether to incorporate it into the system. I f the CCB accepts the CR, an order to adapt the requirements is issued to respective team members, who then embed the new requirements into the system based on their designated roles. During this process, t he development team encounters the following issues:

**Issue 1.** Analysis of the change impact and determining whether or not to reflect the request for chang e in requirements: Is it feasible to implement the submitted request for changes in requirements within the current system?

**Issue 2.** Tracing the requirement change: Which elements require modification to align with the newl y proposed requirements as they are implemented into the system?

[1]*Graduate School of Management of Technology, Sogang University, Seoul, Korea (psjdream@sogang.ac.kr)*

Received: Sept. 15, 2023, Revised: Sept. 26, 2023, Accepted: Sept. 27, 2023

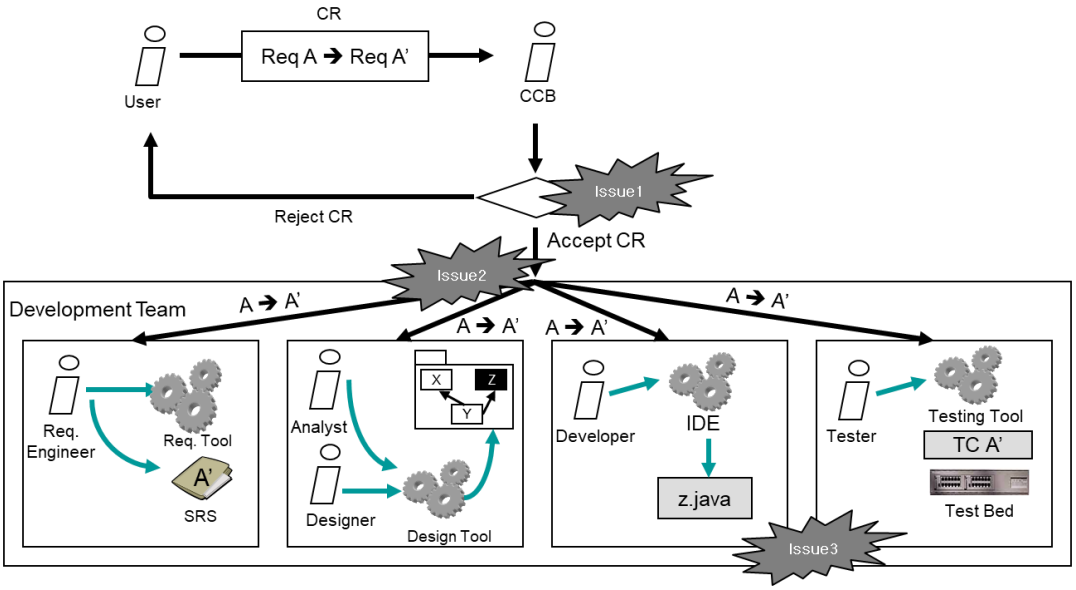



**Figure 1.** Problematic software change request management process

**Issue 3.** Sustaining consistency across the changed artifacts: Does consistency persist among altered s oftware artifacts following the assimilation of new requirements into the system?

The pivotal solution for addressing these challenging issues is establishing a well-defined requirement s traceability (TR) model. Requirements traceability denotes the inherent capability of a methodology en abling changes in any software development artifact, including requirements and specification implemen tation, to be systematically tracked within the relevant sections of other artifacts [4]. The core of require ments change management lies in assessing the cascading impact on the components that constitute the r emainder of the system upon modifying a particular requirement. This necessitates clearly and distinctly identifying the TR link between each element, from detailed specifications to implementation code. Rec ently, various techniques in the AI field have been applied to the problem of automatic configuration of TR links. Still, there are hidden problems behind improving the recall value suggested by such technique s. As long as the recall does not reach the value 1.0, meaning complete coverage in the TR link setup, th ere is a possibility of losing the link from the requirements artifact to the design artifact, corresponding t o the starting point of the TR link connection. In the requirements traceability setup problem, the positio n of the missing TR links is important in addition to the numerical recall value.

This study proposes a method to (1) create a requirements traceability model while minimizing the los s of the top-level trace links from requirements artifacts to design artifacts, (2) extend the requirements tr aceability model to design phase artifacts, and (3) measure the impact degree of each requirement chang e. This study uses the automatic analysis pattern generation technique using the CRA (Class Responsibil ity Assignment) pattern language proposed in previous research [5]. Applying the CRA pattern language from requirements to create an analysis model automatically establishes a trace link between the analysi s entity and the requirement entity that serves as its rationale. As a result, the TR link for all entities in th e analysis model created by applying the CRA pattern is automatically set. What differentiates the propo sed approach is that it synchronizes the timing of creating the artifact and establishing the TR link with t he artifact. It means there is no orphan artifact node in the analysis model, and the top-ranked links of a r equirement traceability model are secured. According to the application of the CRA pattern, the TR link automatic setup process is demonstrated through the *Course Registration* System example.

The rest of the paper is organized as follows: Section 2 presents related works on the software require ments traceability management approaches. Section 3 gives an overview of the presented requirements tr aceability management process. Section 4 introduces each traceability node and links in the proposed tra ceability model, and each element is demonstrated with a case study using the *Course Registration Syste m*. Section 5 discusses the conclusion of this study with future work plan.

## II. Related Work

Current approaches to facilitating software requirements traceability management can be grouped into three categories: manual-based traceability management, semi-automated traceability management, and fully automatic traceability management. First, the manual-based requirements traceability management method finds support in numerous commercial traceability tools, including RETH [6], DOORS [7], RTM [8], RequisitePro [9], and others. Despite developers utilizing a specific tool, it does not alleviate the necessity to independently decipher the TR links between artifacts and establish them between different types of artifacts from different development phases through the tool. The manual set-up of TR links can lead to potential errors like omitting requirements TR links or establishing incorrect links. Moreover, setting up TR links is intricate and still demands a substantial investment of time and resources. Consequently, due to these inherent drawbacks, even with a requirements management tool, its practical utilization tends to be limited in actual software development endeavors.

Second, the semi-automated approach to requirements traceability management involves traceability management tools automatically aiding the creation of certain TR links. This technique relies on essential data derived from previous developers' TR link setup history [10][11][12] for establishing the fundamental TR links. The initial task of identifying these TR links still rests with the developer. Consequently, issues akin to inherited errors from initial links by developers, intricacies in establishing TR links, and the considerable time and cost with manual link setup, which were predicaments of manual traceability management, remain unresolved.

The third approach is to support a tool that autonomously facilitates establishing requirements TR links. Several traditional techniques encompassing information retrieval (IR) [13][14][15], traceability establishment rules [16][17], mathematical axioms [18], specialized types of integrators [19], and others are employed to create these TR links. Recently, Recently, machine learning-based approaches have been introduced as a solution for requirements traceability recovery [20][21][22]. [20] proposed a method of learning a machine learning classifier that can discover a new TR link using a priori knowledge and update the existing traceability. As a result of performance comparison with seven popular technologies among existing IR technologies, it showed an advantage in terms of F-score. On the other hand, [21] proposed a framework called TVace BERT (T-BERT), a modification of BERT, for generating TR links between source code and artifacts written in natural language. It showed better results than the VSM model, which is analyzed as the performance improvement obtained using the pre-trained language model and transfer learning. [22] judged that there was a limit to traceability recovery only by applying learning techniques, and proposed a framework called SPLINT that integrates hybrid textual similarity measures and semi-supervised learning strategies. The integrated application of the two techniques proved that the imbalance and sparsity problems in the data related to requirement traceability were solved to some extent.

Except for the manual TR link setup approach, the semi-automatic and fully automatic approaches to requirements traceability management predominantly rely on historical a priori data when establishing traceability relationships. The utilization of past data aids in establishing TR links among present software development products. However, given that past decisions cannot be unequivocally projected onto the future, ensuring 100% adherence, the comprehensive assurance of requirements traceability remains fundamentally uncertain. The amount of benefit stemming from upholding accurate requirements for traceability relationships hinges on the extent of completeness in traceability establishment. As a result, the potential omission of traceability relationships, inherently implied within existing research, diminishes the benefit the demanded developers' endeavors toward requirements tracing management.

## III. Automatic Generation of Requirement Traceability Link Based on Class Responsibility Assignment Pattern Language Application

Existing automatic requirements traceability approaches encompass diverse models, data, and techniques, leading to variations in the resulting precision and recall values of the TR links set through their application. Nonetheless, a shared and fundamental Issue emerges from the discrepancy between establishing the TR links and the software artifact creation time. Following the creation of artifacts, specifically after registering them within the common storage, a TR link connecting these artifacts comes into setup. While the accuracy of the TR link can be enhanced by utilizing knowledge inherent in historical data via v

arious techniques or models, achieving a recall of 1—indicating perfect completeness in the TR link setup—remains elusive.

In the configuration of TR links, the recall value carries relatively more significance than the precision value. If an incorrect TR link is established, particularly a trace link connecting requirements and analysis artifacts at an early stage, losing this link can trigger a ripple effect, leading to the forfeiture of trace links to numerous artifacts that necessitate ongoing traceability in subsequent development phases. For instance, even with a recall of 0.999, the misplacement of a trace link accounting for 0.001, especially if it pertains to a link between early-stage products proximate to the root of the traceability tree, can engender the loss of traceability across all products after the misplaced trace link.

To resolve this problem, this study establishes several strategies for designing the requirements traceability management method.

1. Storing software artifacts within a shared repository mandates the establishment of a trace-from link (which explains why the artifact was identified based on another artifact). Artifacts generated by developers locally may not necessitate traceability setup; however, when considering the addition of an artifact—tested within a developer's local area—to the common repository, supplementary information is required to delineate which artifact from the prior stage served as the foundation for the new artifact. Consequently, these requirements prevent the generation of artifacts corresponding to orphan nodes by aligning traceability establishment timing with artifact creation. The ultimate aim of this strategy is to achieve a recall value of 1.0 for the requirements traceability setup, which means there is no missing TR link.
2. The means to mitigate the effort needed for specifying a trace-from link is necessary. The first strategy above might encounter opposition from developers, as it necessitates trace-from link setup for all artifacts—an obligation that was absent before. Consequently, an avenue to minimize the developer's exertion required for TR link setup should be explored.

Implementing the first strategy above within an actual development project is not quite difficult. A comparatively simple solution could entail guiding the user to include the name of the artifact, which is the destination of the trace-from relationship alongside the file when committing to git involving a particular file. Besides the simple solution, developing a dedicated tool should be supported to visualize the traceability tree or conduct change impact analyses based on the trace-from relationship—this paper's focal point of discussion centers around the execution of the second strategy. In our previous study [5], the analysis model is automatically generated from the requirements model using the Class Responsibility Assignment (CRA) pattern language. The method presented in this paper simultaneously establishes TR links from requirements to analysis when the analysis model is generated. Given that the CRA pattern language's application domain is business applications, the scope of application for the automatic generation model of requirement TR links proposed in this paper is likewise confined to systems situated within the realm of business applications.

Within the spectrum of TR link configurations among artifacts spanning diverse phases, the significance of automating the establishment of TR links during the analysis phases within the requirement phase is notable. This is particularly attributed to the fact that tracing in the requirements-analysis stage entails a considerably heuristic task, distinct from the subsequent traceability that extends to design-implementation-testing stages, necessitating a substantial commitment from developers. It means the potential benefits of automated TR link setup are relatively more substantial. From other aspects, the TR links between the requirement and analysis phases encompass links between nodes nearest to the root node within the traceability tree structure. Given that the scope of traceability relationships lost when a single link is absent is the widest, the importance of the automatic establishment of precise links is comparatively great.

### 3.1. Automatic Generation Process of Requirement Traceability Link Based on CRA Pattern Application

Fig.2 shows a process of automatically generating analysis elements through pattern instantiation using the CRA pattern language and automatically generating trace-from link between the automatically generated analysis elements and the requirement elements corresponding to the rationale describing.

(1) The system analyst refers to the use case specification and answers each item of the pre-designed question set suggested by [5].

(2) The atomic CRUD (Create/Read/Update/Delete) operation type is defined, encompassing the specific sequence for each operation type and the parameter values linked to the sequence for each chosen CRUD operation. During this phase, aside from the response values for each query provided by the system analyst, the pre-identified key abstraction model, including important entity classes and relationships among them, outlining the interconnection between them, become pivotal factors in determining a distinct CRA pattern. A selection of two to six CRA patterns is made to construct a sequence diagram that embodies a single business service flow. Adhering to the pre-arranged sequence of the chosen CRA pattern, it is interwoven into the skeletal structure of a sequence diagram that corresponds to a single service flow.

(3) The answer value for each question outlined by the system analyst in Step 1 also serves as a criterion for selecting the CRA pattern; a portion of these answers corresponds to a parameter value encompassed within the chosen CRA pattern. Answer values are allocated to each parameter featured within the object name or message name at the top of the sequence. This results from the interweaving of CRA patterns in step 2, the instantiation of the flow sequence diagram. The newly generated classes and operations during the instantiation of a sequence diagram are automatically added to the analysis model.

(4) Whenever elements are added to the analysis model due to CRA pattern instantiation, the relationship (trace-from link) between the added analysis element itself and the requirements element that is the basis for its identification is automatically added to the requirements traceability model. Traceability-related information can be manually added when creating a key abstraction model used as an input for pattern instantiation. The TR links for the analysis elements identified after pattern instantiation can also be added.

To run the entire process described in Fig. 2, the support of an automated tool is required. The Analysis Model Skeleton Generator mentioned in Fig.2 is that. With the tool, developers select a proper word to answer a question from use case specifications written in Microsoft Word. Then, a proper set of the CRA pattern will be automatically selected, and each word developers select will substitute each pattern variable. Consequently, a UML authoring tool will automatically create the sequence diagram. At the same time that the Analysis Model Skeleton Generator automatically creates an analysis model, it can create a TR link from the element that became the rationale for the analysis model creation in the requirements specification to the created analysis element. As a result, it means that the time of the creation of the analysis model and the time of the TR link between the requirements-analysis stage corresponding to the point where most TR links are missing can be matched. To this end, a well-defined requirements traceability model must first be defined. Next, we will examine the main structure of the requirements traceability model proposed in this study.

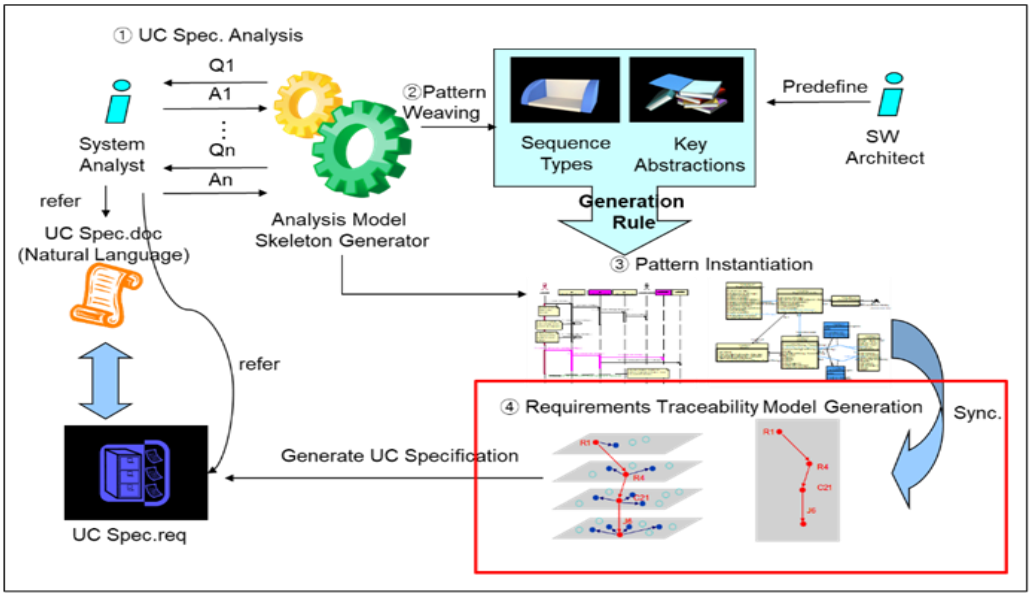



**Figure 2.** Process overview of automatic generation of requirement traceability link based on class responsibility assignment pattern application

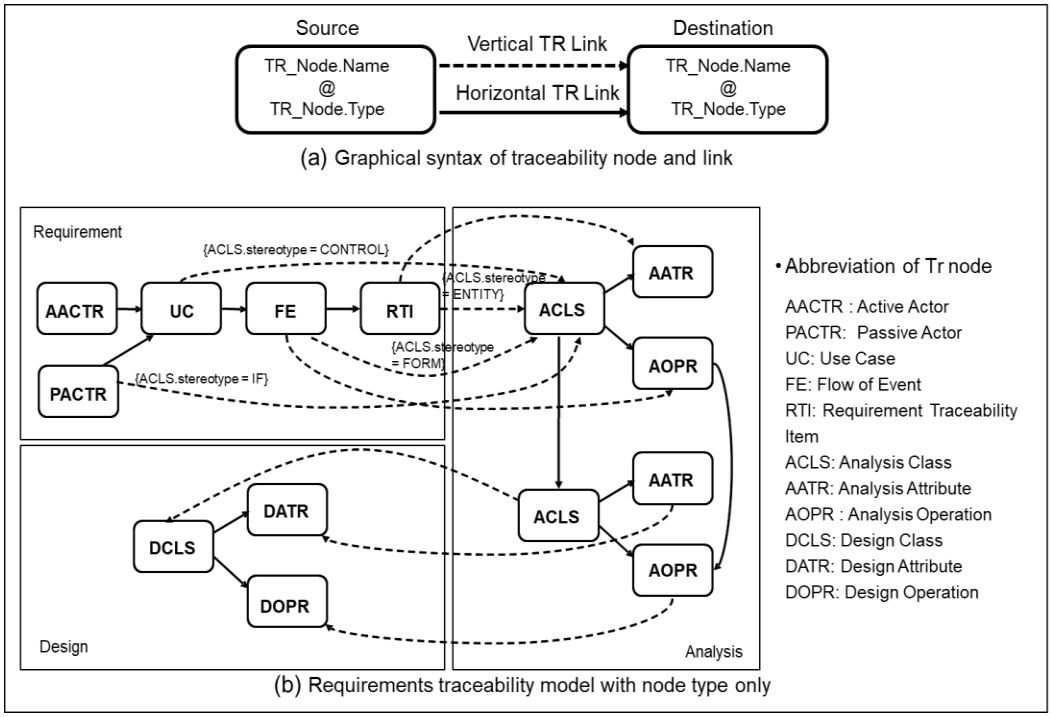



**Figure 3.** Requirements traceability model: graphical syntax for trace nodes and trace links

### 3.2. Traceability Model from Requirements to Design Elements

The graphical syntax of the model expressing requirements traceability is shown in Fig. As shown in 3(a), each traceability node displays the name and type of artifacts the node represents as TR_Node.Name@TR_Node.Type in a rounded rectangle. An arrow with a direction from the source node to the destination node in Fig.3(a) indicates the trace-to link. A horizontal TR link is a trace-to link between artifacts of the same development phase marked with a solid line. On the other hand, the trace-to link between artifacts created in different development phases is a horizontal TR link marked with a dotted line.

Fig. 3(b) describes the types of traceability nodes constituting the requirement traceability model proposed in this study and the TR link connection relationship between the types. Artifacts in the stages of implementation and testing that need to be traced from design artifacts can be automatically secured if a design tool that supports forward engineering is used. Therefore, this study defines the trace-to link between artifact nodes from the requirement phase to the design phase.

## IV. Case Study with Course Registration System

In this section, the feasibility of the proposed model is evaluated by showing stepwise application of each traceability node and the TR link settings between nodes in Fig.3 together with the case registration system example.

### 4.1. ACTR (Actor: AACTR/PACTR) and UC (Use Case) Node

When a use case model is selected as a requirements model, the components of the use case model are use cases and actors. Actors are divided into active actors, who send messages that drive use cases to the system, and passive actors, driven by receiving specific messages from the system while performing use cases. In the use-case diagram, the actor who shoots the arrow of the communication-association relationship as a use-case is the active actor, and the actor who receives the arrow is the passive actor. Fig. 4 (a) and 4(b) show the traceability model of the requirements phase corresponding to some fragments of the use case diagram related to the register for courses use case. The actors and use cases of the use case model correspond 1:1 to the traceability nodes UC, AACTR, and PAACTR nodes, respectively. The poi

nt to note here is that the direction of the relationship on the use case diagram and the direction of the tra ce-to link in the traceability model do not match each other. The relationship on the use case diagram ind icates the direction of communication invocation. Still, the direction of the link in the traceability model indicates the direction of trace-to. In other words, because the use case is influenced by who (what) the p assive actor is, the trace-to link directs the PACTR node, the node corresponding to the passive actor, to the UC node.

### 4.2. FE (Flow of Event) Node

The FE (Flow of Event) node is a traceability node belonging to the requirements phase and correspon ds to each flow on the use case specification. Accordingly, a trace-to link is established from the UC nod e mapped to the use case in which the corresponding FE is identified. The name of the node is the same as the name of the corresponding event flow. If there is only one main flow without other sub flows in th e basic flow in the use case specification, it is named *"use case name.BasicFlow".* Fig. 4(c) and 4(d) sho w the relationship in which the FE node corresponding to the flow called *Create a Schedule* is affected b y the UC node called *Register for Courses*, which was already identified in Fig.4 (b) In the case of the ex ample use case, the basic flow contains two or more subflows. In this case, the name of the correspondin g flow is used as the name of the FE node.

### 4.3. ACLS (Analysis Class) Node

In the case of the ACLS node representing the analysis class, the stereotype of each class is additionally displayed as a tagged value on the vertical trace-to link, as depicted in Fig.5 (a). Classes identified in the Analysis Phase can be identified by dividing into three stereotypes: <<Control>>, <<Boundary>>, and < <Entity>> according to the guidelines of Unified Process [23]. <<Boundary>> In the analysis class, whi ch is a stereotype, the Form class, which mainly serves as a graphical user interface, and the target syste m to be interfaced in the future Interface class that implements the interface protocol with the passive act or, which becomes the system, belongs to two classes. Therefore, the requirements element that is the rat ionale for identifying the Form class and the Interface class is different. Since the guideline is to identify one Form class per flow of event on the use case specification, a vertical TR link is established from the FE (flow of event) node of the requirements phase accordingly. On the other hand, the Interface class es tablishes a vertical TR link from the PACTR (passive actor) node of the requirements phase.

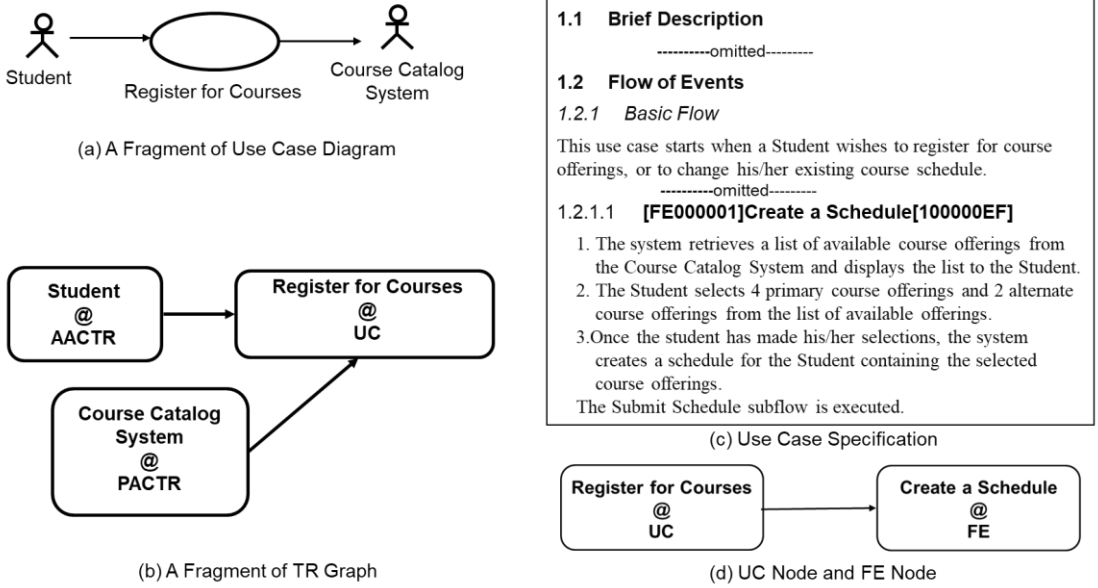
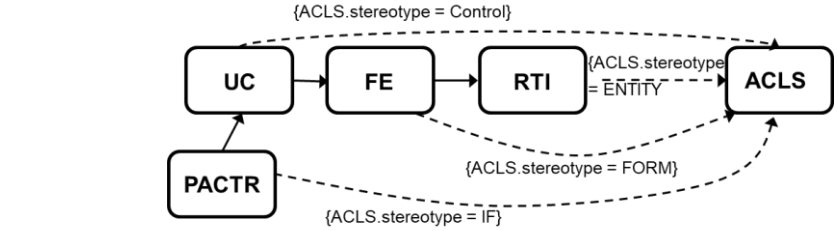



**Figure 4.** An example of ACTR and UC node identification for the *Course Registration System*

In Unified Process, the <<Control>> class guides to assign one per each use case. Therefore, a vertic al trace-to link from the UC node, which is a node of the requirements phase, to the ACLS node corresp onding to the <<Control>> class is established. Since the <<Entity>> class is a class that contains infor mation to be managed by the system, the reason for its identification must be found among the nouns des cribed in the use case specification. Although it is not a special element of the requirements model, word s included in the use case specification that are the basis for extracting the analysis model's classes, attrib utes, and operations elements are identified as Requirement Trace Item (RTI) nodes. The vertical trace-t o link from the RTI node to ACLS (corresponding to the Analysis <<Entity>> class) is set.

Fig. 5(b) is the specification part of the Create a Schedule flow of the *Register for Courses* use case. I n applying the CRA pattern among the use case specifications written in MS Word, if the system analyst designates a word as an answer to a question, a kind of markup is set before and after the designated wo rd. This markup sets hyperlinks between the requirements and analysis elements derived from it. A tool t hat supports automatic generation of a traceability model currently under development stores such hyper link information through integration with MS Word. In the previous examples of traceability node identi fication, the role of the markup is the same as that. Fig. 5(c) diagrams the vertical trace-to link between t he analysis class (denoted by ACLS node) identified from the use case specification and the requirement s element node, which is the source of extraction of each analysis class on the specification. It can be co nfirmed that the trace-to link between each node is connected by stereotype in the examples in Fig. 5(c) according to the designed traceability model in Fig. 5 (a). Fig. 5 (c) shows only the TR link between the requirements node and the analysis node derived from it. The vertical TR link between the artifact nodes of the two phases and the horizontal TR link between ACLS, the output of the same analysis phase, is o mitted to lessen the complexity in the figure. The horizontal TR links between ACLS are established sim ultaneously when the relationship among analysis classes is generated, the same as the vertical TR links.

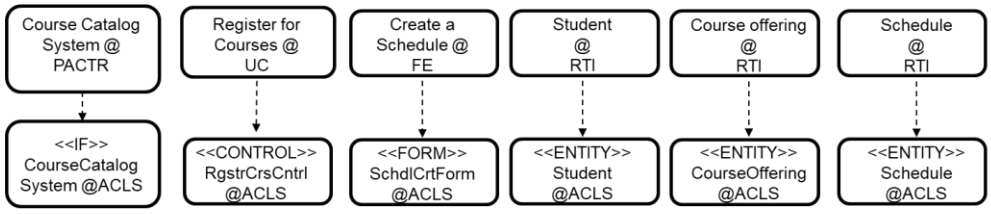



(a) Requirements Traceabilty Model

**1. [UC000001]Register for Courses[100000CU]**
**1.1 Brief Description**
----------omitted---------
**1.2 Flow of Events**
*1.2.1 Basic Flow*
This use case starts when a Student wishes to register for course offerings, or to change his/her existing course schedule.
----------omitted---------
1.2.1.1 **[FE000001]Create a Schedule[100000EF]**

1.The system retrieves a list of available course offerings from the **[PACTR000001]Course Catalog System[100000RTCAP]** and displays the list to the **[RTI000001]Student[100000ITR].**
2.The Student selects 4 primary **[RTI000002]course offering[200000ITR]s** and 2 alternate course offerings from the list of available offerings.
3.Once the student has made his/her selections, the system creates a **[RTI000003]schedule[300000ITR]** for the Student containing the selected course offerings.
The Submit Schedule subflow is executed.

(b) Use Case specification for *Register for Courses*




(c) Trace-to relationship from requirements phase nodes to ACLS nodes

**Figure 5.** An example of ACLS node identification for the *Course Registration System*

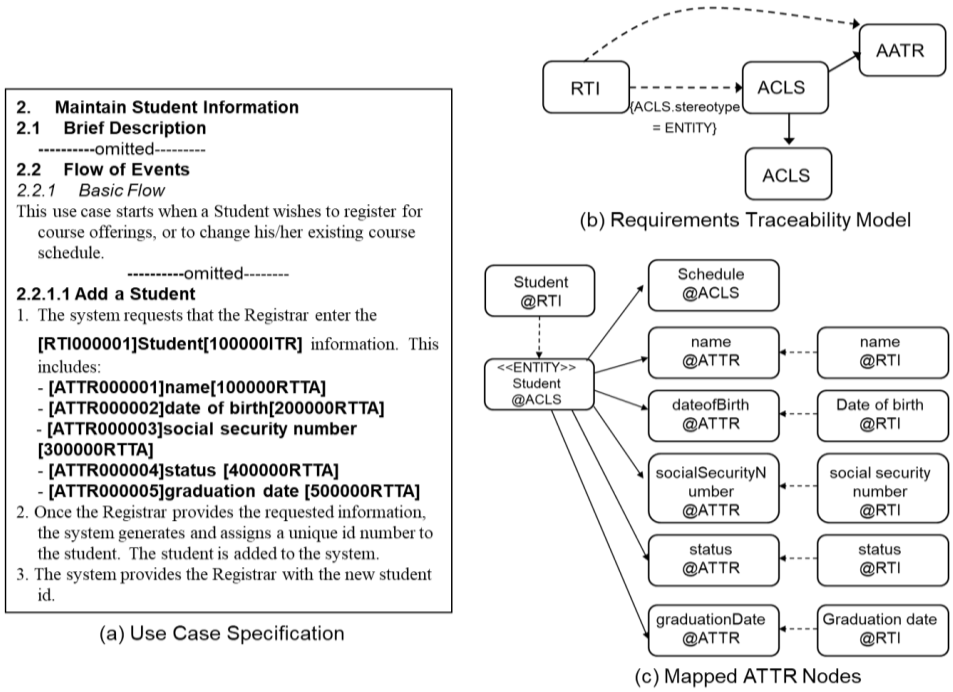



**Figure 6.** An example of ATTR node identification for the *Course Registration System*

### 4.4. ATTR(Analysis Attribute) Node

AATR node is an abbreviation of the Analysis Attribute node and is a traceability node that indicates attributes of the analysis class. As depicted in Fig. 6(a), since it is mainly extracted from words corresponding to nouns among use case specifications, a vertical trace-to link is set from RTI (Requirement Traceability Item) among node types of the requirements phase. Among the node types of the same analysis phase, a horizontal trace-to link must be maintained from the ACLS node, which is a node type that indicates an analysis class, including AATR. According to the trace-to link setting rule between these traceability types, an example of ATTR nodes traced in the Add a Student flow of the Maintain Student Information use case specification in Fig. 6(c) is shown.

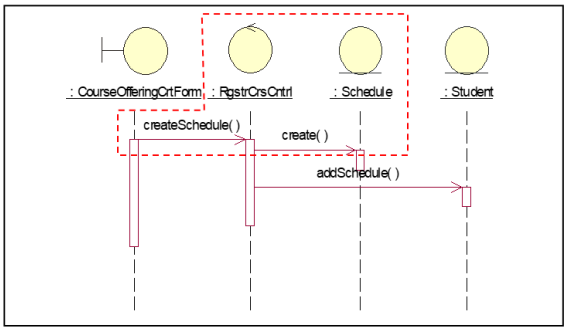
### 4.5. AOPR (Analysis Operation) Node

An AOPR node is a traceability node that refers to an operation belonging to an analysis class. Therefore, it has an ACLS node corresponding to the class to which the corresponding operation belongs and a horizontal trace to link. The requirement tracking unit of this tracking model is a function service, and a function service corresponds to one event flow in the use case specification. Since an operation is the behavior of the minimum unit system identified in the process of realizing a flow of event, the tracking node in the requirements stage to be used as the basis for AOPR identification becomes the FE node. As shown in Fig.7, one flow called Register for Courses is realized as a sequence diagram like Fig.7(b) as the result of applying the CRA pattern language. Fig.7(c) shows the *create@AOPR* node, which refers to the operation create ( ), an AOPR-type traceability node, by highlighting only the TR links connected to this node and the nodes connected by the TR link.

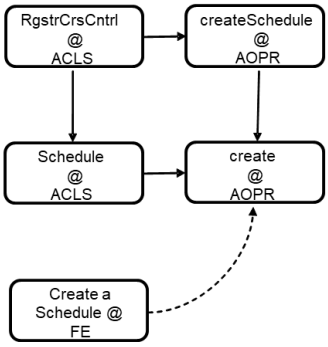
create ( ) is an operation belonging to the Schedule class. Therefore, *Schedule@ACLS* and trace-to link are established. Looking at the sequence of Fig.7(b), it can be seen that the *Schedule.creat( )* operation is called on the focus of control of *createSchedule( )*, an operation of *RgstrCrsCntrl*, a <<Control>> class. That means *Schedule.creat( )* is called in the implementation part of *RgstrCrsCntrl.createSchedule ( )*. These operations are called on the service flow called Create a Schedule. Therefore, a trace-to link from Create a *Schedule@FE* node to *create@AOPR* node is required. Of course, this trace link is also set as *createSchedule@AOPR*, but Fig.7(c) focuses only on the TR links related to *create@AOPR*, so it is not displayed. The pieces of the traceability model represented in Fig. 7(c) are identified simultaneously as the part marked with a dashed box in the sequence diagram of Fig. 7(b) is identified.

1.2.1.1 **[FE000001]Create a Schedule[100000EF]**
1. The system retrieves ----------omitted----------
2. The Student selects 4 primary [RTI000002]course offering[200000ITR]s ----------omitted----------
3. Once the student has made his/her selections, the system creates a [RTI000003]schedule[300000ITR] for the Student containing the selected course offerings.

(a) Use Case specification for *Create a Schedule* flow of event




(b) Automatically generated sequence diagram by applying CRA pattern language[*]




(c) Generated AOPR nodes and trace-to links

**Figure 7.** An example of AOPR node identification for the *Course Registration System*

### 4.6. Extension of the traceability model when transitioning from analysis model to design model

After the creation of the analysis model based on the requirements is completed, the baseline for the a nalysis model is set through verification. The baselined analysis model becomes an initial version of the design model. Therefore, identification of the design artifact node when baselining the analysis model an d establishing a TR link from a node belonging to the analysis model to an initial node belonging to the design model can be completed relatively simply. Copy all traceability nodes of ACLS, AATR, and AO PR types corresponding to artifacts created in the analysis phase and change only the type to DCLS, DA TR, and DOPR. Set up a vertical trace-to link from an ACLS-type node with the same name to a DCLS-t ype node. The remaining DATR and DOPR types of node identification and trace-to link are performed similarly.

The design model created as a result of the baselining of the analysis model is refined by applying a s pecific architectural pattern in the design phase or by adding library classes provided by the framework. Whenever new elements are added as a result of the refinement of the design model, they must be imme diately reflected in the traceability model. When a new node or link is created, the rules for setting TR li nks are the same as those applied to traceability nodes in the analysis phase. The only difference is that t he prefix of the type name is 'D' instead of 'A.'

### 4.7. Visualization of change impact utilizing the requirements traceability model

The requirements traceability model proposed in this study is designed in a graph structure. Therefore, when a TR link is established between the requirements phase, analysis, and design phases when a speci fic requirement is requested, it is possible to visualize how the change affects the entire software artifact immediately. Fig.8 shows an example of visualizing the impact of a change when a change occurs in a p art of the traceability model and a specific node among them. When evaluating change impact, the requir ed input is the traceability node and trace distance where the change request occurred. Trace distance is a feature that specifies the degree to which the range of change impact is to be set, depending on the natur e of the change request. Nodes that can be reached from one node through one trace link are nodes withi n the range of trace distance 1. The number of trace links that must be passed through to reach another n ode from one node becomes the trace distance value. As described in the previous subsection, the trace li nk is divided into a horizontal TR link, a TR link between artifact nodes belonging to the same develop ment phase, and a vertical TR link, a TR link between connected artifact nodes belonging to different de velopment phases.

The traceability graph at the left of Fig. 8 shows a problem in identifying the active actor, Student (Stu dent@AACTR). However, this is a rare case, and it is a case in which all requirements related to the acto

r, Student, need to be re-examined. This situation can usually arise early in the requirements engineering phase. Therefore, it is necessary to identify related nodes targeting only the traceability nodes belonging to the requirements phase. At this time, when trying to obtain a graph showing the change impact, the v ertical trace distance can be set to 0 to limit the range only to related nodes created in the development p hase where the node where the change occurred was created. In the case of horizontal trace distance, dep ending on the severity of the node where the change occurred, the developer assigns one if it is possible t o determine that only the very close nodes are affected and assigns a larger number as the wavelength of the impact is wider.

Another example of change impact analysis using traceability graph is illustrated at the right of Fig.8. The content of the Create Schedule flow, one of the flows of events described on the Register for Course s use case specification, corresponds to one example of traceability graphs that can be created when a ch ange request occurs in a flow of event. Suppose the sequence of a service flow or the operation called on the sequence needs to be changed. In that case, the ripple effect is cascaded along the vertical TR link th at goes through the requirements → analysis → design phases. Therefore, as shown in Fig.8, the change impact can be visually confirmed by arbitrarily setting the horizontal trace distance to 0 and the vertical t race distance.

In addition to visualizing the range of impact of a change in a specific node, the number of nodes affe cted by a change in a specific node and the number of total artifact nodes in one system can be counted. Consequently, the impact of the change request can be quantified as a percentage. By quantifying the exi sting qualitative change impact analysis, it is possible to present an objective rationale for determining th e acceptance of change requests raised through various sources.

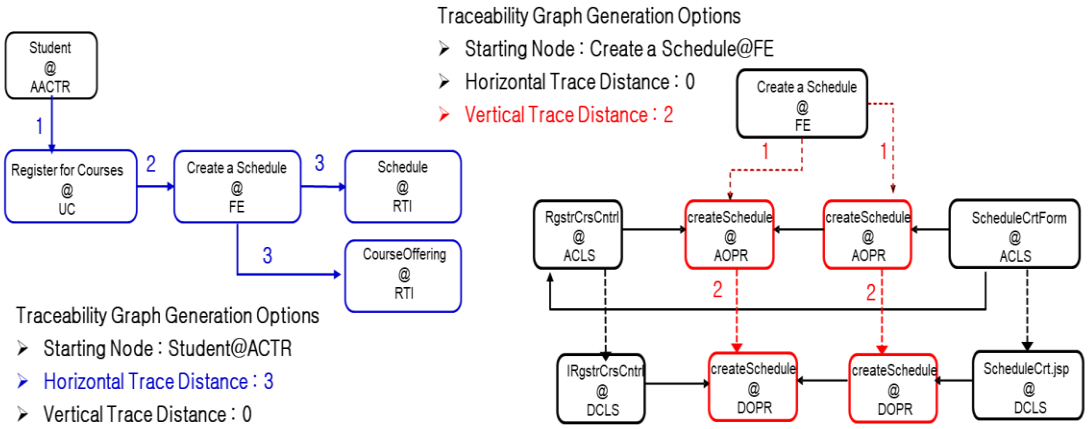



**Figure 8.** Change impact analysis using traceability graphs

## V. Conclusions and Future Work

Requirement change requests during software system development and maintenance are not problems developers can suppress. To conduct systematic requirements change management to respond to unavoid able efficiently and, in some cases, frequently occurring requirement change requests, it is necessary to s olve three issues as described in the introduction section above (1) analysis of the change impact and det ermining whether or not to reflect the request for change in requirements, (2) tracing the requirement cha nge, and (3) sustaining consistency across the changed artifacts. In this study, to resolve these issues, am ong the artifacts generated in each stage from requirements to design, the artifacts to be maintained for tr aceability were identified, and the trace-to link between each artifact was designed for the requirements t raceability model. From the requirement phase to the design phase, the creation timing of each TR link a nd destination node, the newly identified artifact itself, is synchronized by applying CRA pattern langua ge. The feasibility of the proposed requirement traceability model was shown through an application cas e of the Course Registration System belonging to the business application domain category. Through the

case study, each artifact node and the TR link needed for each node were explained with tangible examp le fragments. Change impact analysis visualized with a traceability graph in response to a specific chang e request was also demonstrated with the example case.

The proposed requirements traceability management process automates the traceability node and link setup in conjunction with the artifact generation rule of the CRA pattern language proposed in previous s tudies [5] to minimize the effort of TR link setup. CRA pattern language sets only the business applicati on domain as the target scope, and all elements, from requirements to design, cannot be automatically cr eated. However, an encouraging fact regarding the effect of reducing effort by applying the CRA pattern language is that, according to the evaluation results in [5], 67% of the total number of operations includ ed in the final analysis model was automatically generated. This result means that 67% of the TR links b etween artifacts in the requirement and artifacts in the design phases, which can be critical but most freq uently lost, can be automatically configured. This experimental result is encouraging because it can redu ce a significant part of the developer's effort, and the accuracy of the established TR link is guaranteed.

There is a limit to quantitatively evaluating whether the requirements TR link setting the recall value, as for now. It is necessary to develop a tool that automatically updates the analysis/design elements adde d by the developer using the UML authoring tool to the traceability model after generating the traceabilit y model in conjunction with the CRA pattern. Currently, such a tool is being developed, and as soon as t he tool development is completed, a quantitative evaluation to prove the effectiveness of the proposed tr aceability model is planned.

## VI. References


[1] F. P. Brooks, “No Silver Bullet—Essence and Accidents of Software Engineering,” in Proc. of the IFIP Tenth World Computing Conference, Dublin, Ireland, 1986, pp. 1069-1076.

[2] B. W. Boehm, “Software and Its Impact: A Quantitative Assessment,” Datamation, Vol. 19, No. 5, pp. 48-59, May 1973.

[3] M. Madan, M. Dave, and A. Tandon, "Importance of RTM for Testing a Web-based Project,” in Proc. of the 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 2018, pp.816-818.

[4] S. J. Greenspan, and C. L. McGowan, “Structuring Software Development for Reliability,” Microelectronics and reliability, Vol. 17, No. 1, pp. 75-83, January 1978.

[5] S. Park, “A Pattern Language for Class Responsibility Assignment for Business Applications,” International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 12, No. 10, pp. 586-601, Nov. 2021.

[6] H. Kaindl, “The Missing Link in Requirements Engineering,” ACM SIGSOFT Software Engineering Notes, Vol. 18, No. 2, pp. 3, April 1992.

[7] DOORS, [Online]. Available: https://www.ibm.com/docs/en/ engineering-lifecycle-management-suite/doors.

[8] RTM, [Online]. Available: https://www.jamasoftware.com/ requirements-management-guide/requirements-traceability/how-to-create-and-use-a-requirements-traceability-matrix.

[9] RequisitePro, [Online]. Available: https://www.ibm.com/support/ pages/rational-requisitepro-714.

[10] A. Mahmoud and G. Williams, “Detecting, classifying, and tracing non-functional software requirements,” Requirements Engineering, Vol. 21, pp. 357–381, Sep. 2016.

[11] M. Rahimi and J. Cleland-Huang, “Evolving Software Trace Links between Requirements and Source Code,” in Proc. of the 2019 IEEE/ACM 10th International Symposium on Software and Systems Traceability (SST), Montreal, QC, 2019, pp. 12.

[12] R. Mordinyi and S. Biffl, “Exploring Traceability Links via Issues for Detailed Requirements Coverage Reports,” in Proc. of the 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW), Lisbon, Portugal, 2017, pp. 359-366.

[13] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, “On the Equivalence of Information Retrieval Methods for Automated Traceability Link Recovery,” in Proc. of the 2010 IEEE 18th International Conference on Program Comprehension (ICPC), Braga, Portugal, 2010, pp. 68-71.

[14] H. Hayes, A. Dekhtyar and J. Osborne, “Improving Requirements Tracing via Information Retrieval,” in Proc. of the 11th IEEE International Requirements Engineering Conference, Monterey Bay, CA, USA, 2003, pp. 138-147.

[15] N. Ali, Y. -G. Guéhéneuc, and G. Antoniol, “Trustrace: Mining Software Repositories to Improve the Accuracy of Requirement Traceability Links,” IEEE Transactions on Software Engineering, Vol. 39, No. 5, pp. 725-741, May 2013.
[16] B. Ramesh and M. Jarke, "Towards Reference Models for Requirements Traceability," in IEEE Transactions on Software Engineering, Vol. 27, No 1, pp. 58-93, Jan. 2001.
[17] G. Spanoudakis, A. Zisman, E. Pérez-Miñana, and P. Krause, “Rule-based Generation of Requirements Traceability Relations,” Journal of Systems and Software, Vol. 72, No. 2, pp. 105-127, July 2004.
[18] F. Pinheiro and J. Goguen, “An Object-Oriented Tool for Tracing Requirements, ” IEEE Software, Vol. 13, No. 2, pp.52-64, March 1996.
[19] S. A. Sherba, K. M. Anderson, and M. Faisal, “A Framework for Mapping Traceability Relationships,” in Proc. of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE 2003), Montreal, Canada, 2003, pp. 32-39.
[20] C. Mills, J. Escobar-Avila, and S. Haiduc, “Automatic Traceability Maintenance via Machine Learning Classification,” in Proc. of the 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), Madrid, Spain, 2018, pp. 369-380.
[21] J. Lin, Y. Liu, Q. Zeng, M. Jiang, and J. Cleland-Huang, “Traceability Transformed: Generating more Accurate Links with Pre-Trained BERT Models,” In Proc. of the 43rd International Conference on Software Engineering (ICSE ’21), Madrid, Spain, 2021, pp. 324–335.
[22] L. Dong, H. Zhang, W. Liu, Z. Weng and H. Kuang, "Semi-supervised pre-processing for learning-based traceability framework on real-world software projects," in Proc. of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), Singapore, 2022, pp. 570–582.
[23] I. Jacobson, G. Booch, and J. Rumbaugh, “The Unified Software Development Process,” Reading, Addison-Wesley Professional, 1999.

## Author

***Soojin Park***

2010.03~2012.02: Assistant Professor in Sogang Institute of Advanced Technology(SIAT), Sogang University
2012.03~Present: Assistant Professor in Graduate School of Management of Technology, Sogang University

Research Interests : Requirements Engineering, Software Architecture, Big Data Computing