2010 Asia Pacific Software Engineering Conference

A Software Tag Generation System to Realize Software Traceability

Shinya Yamada, Masataka Ugumori, Shinji Kusumoto
Graduate School of Information Science and Technology, Osaka University
1-5 Yamadaoka, Suita, Osaka 565-0871, Japan
Email: {s-yamada, mugumori, kusumoto}@ist.osaka-u.ac.jp

***Abstract*—This paper describes a system that supports to generate the software tag which makes software development visible to software purchaser (users). A software tag is a partial set of empirical data about a software development project shared between the purchaser and developer. The purchaser uses the software tag to evaluate the software project, allowing them to recognize the quality level of the processes and products involved. In order to implement the mechanism to use the software tag effectively, it is necessary to support generating the software tag. We have implemented a system named *CollectTag* that supports to collect data and generate the software tag. We conducted a case study to evaluate the usefulness of CollectTag and generated software tag. The results show that using CollectTag requires low cost to generate the software tag.**

***Keywords*-Project management, measurement, quality, empirical software engineering**

I. Introduction

Software systems are becoming huge and complex, and our everyday life heavily depends on such software systems. In such social context, there are various issues related to software systems. For example, stoppage of a service in the society's infrastructure(ex. financial systems, transportation systems) due to software faults causes huge social loss.

One of the major concerns of software purchasers (users) in Japan is the quality of the software systems. Japanese society generally demands high-quality software systems with low fault rates and high operability levels. On the other hand, many software purchasers in Japan are not knowledgeable about the nature of software. It is reported that only 40% of Japanese major companies employ a full-time Chief Information Officer (CIO) and that only 20% of all CIOs are confident of their knowledge about information technologies[12].

Without a sufficient understanding of software quality and software projects, many companies try to purchase software systems from software developers (vendors). This produces a very risky situation. For example, purchasers cannot specify system requirements very well, and they do not oversee the project properly. Such situations often lead to project failures. It is reported that only 31.1% of software projects are recognized as "successful projects" in Japan[13]. To confront these issues, there is strong demand to provide transparency of software projects to the software purchaser and improve communications between purchaser and developer.

A new concept "Software Traceability" has been proposed to solve the issues [6][15]. That is, we apply the concept of traceability in food distribution to the software development process and traceability information, indicating "when, where, by whom and how" the software has been developed, is attached as a "software tag" to the actual software product. The Software Tag is a new scheme to provide information feedback about the project from the developer to the purchaser. It establishes transparency of the software development project by allowing purchasers to view and analyze the elements of the tag. The Software Traceability and Accountability for Global Software Engineering (StagE) project [6][15] is a government-supported project that pursues standardization and promotion of the software tag scheme. In this project, we have defined the detailed structure of the software tag.

There remain several challenges to implement the software traceability based on the software tag. One of them is to develop technologies to generate/produce the software tag from software development projects. This paper proposes a software tag generation system. In developing the system, we focus on the following requirements: (1) it has general versatility, that is, it can be applied to the wide variety projects, (2) the cost to generate the software tag should be low, that is, it can collect necessary data from the project and generate the software tag at low cost and (3) it generates a convenience software tag, that is, the generated tag can be easily used for analysis of software projects. Based on the requirements, we have implemented a prototype system named *CollectTag*. CollectTag allows the user to define concrete software tag definitions for his/her own project. Also, it provides automatic measurement mechanisms to collect metrics for specific elements of the software tag. Finally, it outputs the software tag in the XML format. We have applied CollectTag to the actual software development project data and evaluated the usefulness of the system from the viewpoint of cost to generate the software tag. As the results, we found that by using CollectTag, generating the software tag is not high-cost.

Section II briefly describes the software traceability and the software tag. Next, Section III proposes a software tag generation system, CollectTag and Section IV evaluates

1530-1362/10 $26.00 © 2010 IEEE
DOI 10.1109/APSEC.2010.55

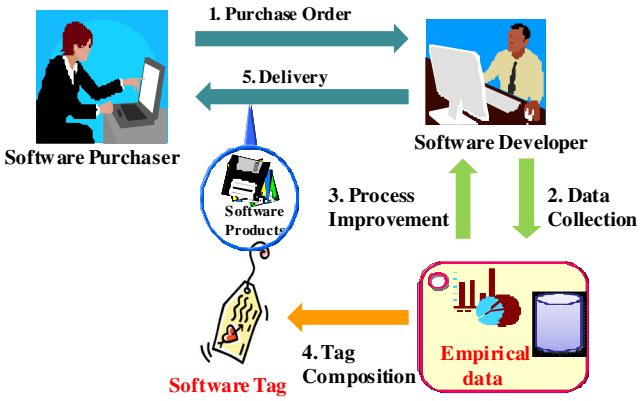

Figure 1. Overview of the software tag use

the usefulness of the system through a case study. Finally, Section V concludes this paper.

II. Software Traceability based on the Software Tag

A. Overview of the Software Tag Use

A software tag is a packaged data set about a software project. It is currently composed of 41 characteristic elements of project data and progress data (See Section II-B). Figure 1 shows an overview of the software tag use.

Step1: A software purchaser orders development of a software system. The purchaser includes both the final products and the software tag in their requirements.

Step2: During software development, various kinds of empirical data are created and generated. For example, requirements documents, software design documents, source code, test cases, issue tracking logs, manual documents, review logs, and quality analysis records may be produced. These are collected and archived. Note that we collect not only the final data at the end, but also interim snapshot data during development.

Step3: The collected data is analyzed for process improvement of the development organization, as is the usual process improvement scheme for software development organizations.

Step4: The collected data is used to construct the software tag. Parts of the empirical data are selected and abstracted into the software tag format.

Step5: The software tag is delivered to the software purchaser periodically during the development and/or finally at the end of the development together with the final software product. The software purchaser evaluates the software development by viewing and analyzing the tag, and accepts the delivered software product.

If a controversy such as a question about the quality of the product occurs between the software purchaser and the developer, the delivered software tag and (if necessary) the empirical data are analyzed, providing a basis for exploring a resolution to the controversy.

The software tag is a key to improving transparency of software projects. By examining the software tag, the software purchaser can identify and understand the development process, which has been mostly hidden from the purchaser. The purchaser can evaluate the quality of the processes and products of the project. For the software developer, the software tag is useful to prove that they have conducted the proper activities in the software project. Also, it can be used to trace the quality of the activities of sub-contractors and sub-sub-contractors (such contracting chains are very popular in Japan).

This scheme can be very useful for offshore and global development, because transparency and traceability of software development can be established with a fairly low overhead for the developers. Standardizing the software tag will help to establish a minimum baseline for project quality, and to improve negotiations over software development contracts. Evaluation of software products and projects based on the objective empirical data contained in the software tag will lead to more healthy use of software in society

B. Software Tag Standard

We have defined the elements of the software tag as shown in Table I, named Software Tag Standard 1.0. It is composed of 41 tag elements, which are categorized into project information and progress information. The project information depicts the overall sketch of the project with various basic pieces of information. The progress information provides qualitative and quantitative indices of project achievement with various measures of the development phases. The tag standard provides more precise explanations and example metrics for each tag element which are not presented here.

We divided project information into five categories described below, and settled tag elements for each category (see Table I).

- basic information of the software project (Basic Information)
- information of the system developed by the project (System Information)
- information of development framework applied to the project (Development Information)
- information of relationships between target project and other projects (Project Organization)
- other information (Others)

To settle progress information, we referred to ISO/IEC 12207 Software Life Cycle Processes and activities[7], and included process, quality and effort information described below into it.

- information of requirements, design, programming and test for the software (Requirements, Design, Programming, and Test)

Table I
SOFTWARE TAG STANDARD 1.0

Classification	Category	No.	Tag Element	Explanation
Project Information	Basic Information	1	Project Name	Unique name of project
		2	Organization	Information of development organization
		3	Project Information	Information needed to identify the project characteristics
		4	Customer information	Information identifying the purchaser or owner
	System Information	5	System Configuration	Information identifying system configuration to label the type of system
		6	System Scale	Development system scale
	Development Information	7	Development Approach	Development process type or techniques
		8	Organizational Structure	Structure of development organization
		9	Project Duration	Information of development length
	Project Organization	10	Super-Project Information	Name of super project which creates this project
		11	Sub-Project Information	Name of sub projects which is created by this project
	Other	12	Special Notes	Other necessary or useful data for interpreting or analyzing tag data
Progress Information	Requirements	13	User Hearing Information	Information of user-requirements hearing
		14	Scale	Amount of requirements
		15	Revisions	Amount of changed requirement
	Design	16	Scale	Amount of design products
		17	Revisions	Amount of changed design
		18	Design Coverage by Requirements	Implementation ratio of design for requirements
	Programming	19	Scale	Amount of programming products
		20	Revisions	Amount of changed programs
		21	Complexity	Complexity of programs

Classification	Category	No.	Tag Element	Explanation
Progress Information	Test	22	Scale	Amount of testing
		23	Revisions	Amount of changed test
		24	Density	Ratio of test to system size
		25	Progress Status	Test progress to plan
	Quality	26	Review Status	Quantity information of review
		27	Review Density	Ratio of review to system size
		28	Review Effectiveness	Ration of found defects to amount of review
		29	Defect Count	Number of defects found by test
		30	Fixed Defect Count	Number of fixed defects
		31	Defect Density	Ratio of defects to system size
		32	Defect Detection Rate	Ratio of detected defects to consumed test
		33	Static Check Results	Report of static checker
	Development Cost	34	Overall Cost	Development and maintenance cost
		35	Productivity	Ratio of amount of products to overall cost
	Schedule and Management	36	Process Management	Information on management of development process
		37	Purchaser-Developer Meeting Status	Amount of user-vendor communication
		38	Total Risk Item Count	Number of risk items in the development
		39	Risk Item Existence Period	Time length between a risk item creation and deletion
	Other Products	40	Scale	Amount of product metrics not listed above
		41	Revisions	Amount of change in products not listed above

- information of quality assurance activities on the project (Quality)
- information of development effort on the project (Development Cost)
- information of project plan and management on the project (Schedule and Management)
- other information for the products attached to the software (Other Products)

It is not mandatory to use all 41 elements in the software tag in all cases. The purchaser and the developer can negotiate and select elements to use. Also, they can discuss and determine the details of the metrics. For example, #19, Scale of Programming, might be agreed to be measured by lines of code without comments. Based on the software tag standard, the purchaser and the developer should decide followings before using the software tag.

- tag elements to be used
- metrics used for tag elements
- measurement targets of metrics (e.g. whole system, sub systems, or files)
- frequency of measurement
- timing of offering the software tag to the purchaser (e.g. every week, every month, or the end of respective process)

In this standard, we have included various kinds of information that are considered important to the purchasers. The overall structure should be simple for the purchaser to understand, so we have tried to keep it as simple as possible. Also, we have tried to keep in mind the balance of the tag elements. This standard does not include tag elements that are computable from other tag elements. There are a number of standards and reports such as SWEBOK, CMMI, ISO/IEC 15939, and reports by the Software Engineering Center in Japan (SEC) which can help interpret the tag elements. The definition process was based on discussions with industry and academic collaborators.

Through the discussion, we recognized that appropriate metrics (tag elements) set and calculation methods of them are different for organizations or projects. Therefore, we made tag elements selective, and on the tag standard, calculation methods of corresponding metrics of tag elements were not included but examples of the metrics are included.

In order effectively to use the software tag in the actual software projects, it is necessary to support generating and analyzing the software tag. In the StagE projects, we have been developing several supporting tools. In the following Sections, we describe the supporting tool for generating the software tag.

III. SUPPORTING SOFTWARE TAG DATA GENERATION

In order to realize the software traceability based on the software tag, first, it is necessary to develop a support mechanism to effectively generate/construct the software tag to various software development projects. This Section describes the requirements for software tag data generation system, our correspondence to them and a software tag data generation system named *CollectTag*.

A. Requirements

We have the following requirements to the software tag generation system:

- Project-independent software tag generation: As described in Section II-B, the purchaser and the developer negotiate and select elements of the software tag to use. Also, the corresponding metrics to the elements are determined by the purchaser and the developer. So, it is necessary to provide a versatile mechanism that can be applicable to various projects.
- Low-cost software tag generation: It is very important issue to reduce the cost of software development projects. In the software development project under a severe budget constraint, it would appear that it's difficult to assign resources to prepare the software tag. Thus, it is necessary to provide an economical support to collect data and prepare the software tag.
- Output a convenient software tag: In order to confirm the content of the tag, providing analysis/visualization tools are useful for the purchaser. Also, to archive, exchange and reuse the software tag, a standard data format is needed. It is necessary to define the format and the tag generation system should provide a feature to output tags that meet the standard.

B. Features of CollectTag

In order to meet the requirements in Section III-A, we realize the following features to the proposed system:
(1) The user can define each element and metric in the software tag for his/her own project.

In order to flexibly generate the software tag, we implement a feature that the user can define the contents of his/her project. Here, we call the definition of the tag-data *tag-data-definition* and call the data collected *tag-data*. That is, the tag-data-definition is defined to each of metrics for each tag element collected in the project. It includes the name, the explanation, measurement frequency and so on (See Section III-D1). On the other hand, the tag data is created for each metric when the user inputs the actual value and each tag includes the value and the created date and time. So, we can grasp the transition of the value for each metric.

Figure 2 shows the process of the tag data definition.

At first, after the user (the developer) and the purchaser negotiate and select the elements in the tag and define the metrics for each element, the user inputs them into the system and makes the tag-data-definition. Then, the system provides the input form based on the definition to the user. According to the input form, the user inputs the data (measurement results) to each element. Finally, the inputted tag data are stored into the tag data repository.

This feature provides the extendable tag data collection. By preparing the frequently-used tag-data-definitions in advance, the system just loads the definition and provides the input form based on the definition. If the user needs

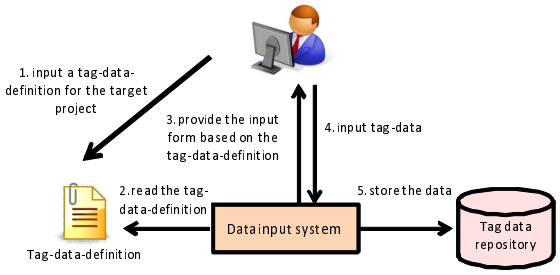

Figure 2. Process of tag data definition

other tag elements and metrics, he/she can add them to the frequently-used tag-data-definition and make the original tag-data-definition for his/her project.

Currently, CollectTag provides the pre-registered tag-data-definitions for the following elements: No.14 Requirements-Scale, No.15 Requirements-Revisions, No.16 Design-Scale, No. 17 Design-Revisions, No. 19 Programming-Scale, No. 20 Programming-Revisions, No. 21 Programming-Complexity, No. 24 Test-Density, No. 27 Quality-Review Density, No. 29 Quality-Defect Count, No. 30 Quality-Fixed Defect Count, No. 31 Quality-Defect Density and No. 32 Quality-Defect Detection Rate.
(2) The system provides automatic measurement for specific elements.

In order to provide the software tag generation at low cost, the system provides automatic measurement for some elements under typical software development. In the software tag, there are several elements related to software products (requirements specifications, design specifications, source codes, bug reports and so on). It needs extra efforts to get metrics from the products. Recently, several software tools are frequently used in the software development. For example, version management systems, such as Subversion and CVS, are well-known and commonly used. Software products and the update history information are stored in the product repository by version management systems. Next, some metrics are measured from the specifications and source codes. It's not realistic to prepare automatic measurement features for any specifications and source codes. So, we focus on the UML models and source code written in Java language since they are popular methodology in recent software developments.

Our system automatically measures some metrics, related to the selected elements in the software tag, from the data in the product repository. Also, bug tracking systems, such as Gnats[5] and KAGEMAI[10], are often used in the software developments. Bug tracking systems store the information about software bugs (faults) and so it's possible to get some metrics, related to the quality elements in the software tag, from the bug data repository.

As a result, we focus on the software development where the following software tools are used:

- Version management system: Subversion, CVS
- Bug tracking system:Gnuts, KAGEMAI
- Modeling tool: UML design tools that output UML diagrams into the XMI files.

Table II summarizes the elements, metrics for each element and sources for the metrics, that our system provides the automatic measurement.

(3) Output the software tag as XML format

XML (Extensible Markup Language) is a set of rules for encoding documents electronically. Many systems contain data in incompatible formats. Exchanging data between such systems is a time-consuming work. XML is a generic data storage format and it is often used to define data format to exchange information between such systems. So, we settled the draft of the standard software tag format which is based on XML format, called standard software tag format data.

C. Utilization process of CollectTag

Here, we explain the process of generating the software tag based on CollectTag. The process consists of the following five steps from the beginning to the end of the project.

Step 1:The user creates the new software tag project.

Step 2:The user makes the tag-data-definition for the project.

Step 3:The user inputs the necessary data to the input form of the project. (some data are automatically inputted (see Section III-B)).

Step 4:Repeat Step 3 at predetermined intervals until the end of the project. (The intervals are determined by the negotiation between the purchaser and the developer.)

Step 5:The final software tag is generated.

The brief overview of Steps 3 and 4 is shown in Figure 3. Figure 3 represents the flow of the project in chronological order. After the project starts, the user inputs/collects the data by using CollectTag. At predetermined intervals (in Figure 3, the interval is 3 days), the user repeats the data input. In Figure 3, on March 27th, the first data input was conducted and on March 30th (after 3 days), the second was done.

Collected tag data are stored in the tag data repository. After completion of the project or as necessary, CollectTag outputs the software tag in the XML format.

D. Structure of CollectTag

Figure 4 shows the system structure of CollectTag. It mainly consists of three units: *Tag data definition unit*, *Tag data input unit* and *Tag data output unit*.

The user inputs the tag-data-definition that defines the set of elements and the corresponding metrics of the software tag through the Tag data definition unit. The Tag data input unit generates the input form according to the tag-data-definition and provides the form to the user. Then, after the user inputs the data on the form, the tag data is stored into

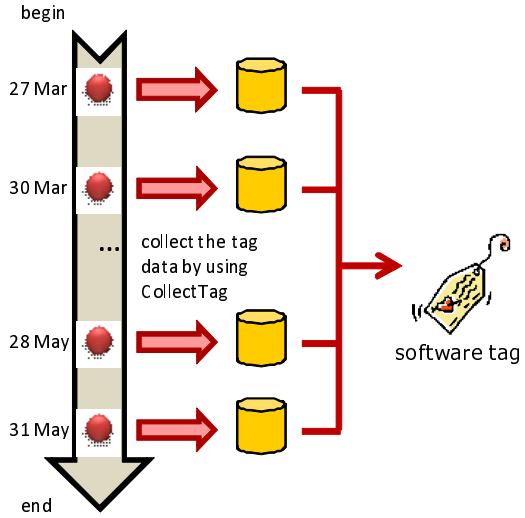

Figure 3. Overview of the tag data collection

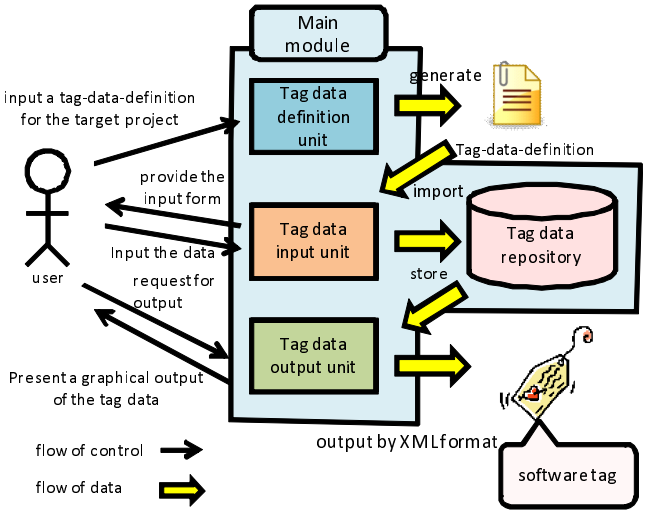

Figure 4. System structure of CollectTag

the Tag data repository. If the user requests the output, the Tag data output unit shows the tag data in graphical/tabular form to the user or translates and outputs the data into XML format.

In the following, we explain the details of each unit and the data items managed in the unit.

1) Tag-data-definition: The tag-data-definition has the information about each of metrics for each tag element. Table III shows the constitution of the tag-data-definition. It consists of the following 5 items:*tagIdentifier*, *name*, *description*, *unit* and *frequency*. *tagIdentifier* is an identifier of the tag-data, *name* is the name of metric for the tag element, *description* is detailed information of the metric, *unit* is the unit of the metric, and *frequency* means the

Table II
ELEMENTS MEASURED AUTOMATICALLY BY COLLECTTAG

Classification	No.	Tag Element	Metrics	Input
Requirements	14	Scale[Transition]	Number of use case diagram Number of use case Number of actor	Modeling file based on XMI
	15	Revisions[Transition]	Number of added, deleted use case diagram Number of added, deleted use case Number of added, deleted actor	
Design	16	Scale[Transition]	Number of class diagram Number of sequence diagram Number of state machine diagram Number of activity diagram	
	17	Revisions[Transition]	Number of added, deleted class diagram Number of added, deleted sequence diagram Number of added, deleted state machine diagram Number of added, deleted activity diagram	
Programming	19	Scale[Transition]	Lines of code	Subversion or CVS repository
	20	Revisions[Transition]	Lines of added, deleted code Number of revisions	
	21	Complexity	CK metrics[1]	
Quality	29	Defect count[Transition]	Number of defects found	Log files of Gnuts or KAGEMAI
	30	Fixed defect count[Transition]	Number of fixed defects	
	31	Defect Density	Number of defects found / Lines of code	
	32	Defect Detection Rate	Number of defects found / Consumed test	

Table III
TAG-DATA-DEFINITION

name	explanation	example
tagIdentifier	identifier	quality.revStatus.revNum
name	name of data item	number of review
description	detailed information	It means the number of review. the review is conducted every Wednesday.
unit	unit of element	number
frequency	frequency of measure	once a week

Table IV
TAG-DATA

name	explanation	example
value	actual measured value	10
tagIdentifier	identifier	quality.revStatus.revNum
id	hash value	1365431584
name	name of data item	number of review
user	measurer	Ugumori
unit	unit of element	number
object	measurement object	minutes of review meeting
date	time and date	2007/4/18 15:00
remarks	remarks	dates of review, 3/27, ...

frequency of the measurement (input of the actual value to the metric).

For example, from Table III, we can see that the quality.revStatus.revNum means the number of review and the review is planned every Wednesday and it is measured once a week.

2) Tag-data: Tag-data is created for each metric defined in the tag-data-definition, when the actual value is inputted by the user. Table IV shows the constitution of the tag-data. It consists of 9 elements, that is, one actual measured value for the tag-data (*value*) and meta data of *value* including 8 elements. *tagIdentifier*, *name* and *unit* are the same as ones in the tag-data-definition. *id* is a hash value, *user* is a measurer of the metric, *object* is a source (ex. source code, minutes) of the metric, *date* is the date when the metric is measured, *remarks* is the remarks about the metric, respectively.

For example, from Table IV, we can see that the project conducted ten times reviews from the beginning to the date (April 18, 2007), the number of the review (10 times) is measured based on the minutes of review meeting and so on. Saving the meta data enables us to check the reason or evidence of the *value* of the tag-data.

3) Tag data definition unit: Tag data definition unit has functionality to add and delete the tag-data-definition described in Section III-D1. Figure 5 shows the edit screen of tag-data-definition. The user makes the tag-data-definition for each element of the software tag collected in the target project through the screen. The tag-data-definitions made by the user are stored in the tag-data-definition directory as XML format and are used for generating the input form.

4) Tag data input unit: Structure of the Tag data input unit is shown in Figure 6. It consists of input screen unit, input form generating unit and automatic data collection unit. The automatic data collection unit reads the tag-data-definitions and generates the input form. Input screen unit provides the input form to the user and the user inputs the actual value into the form. On this occasion, the user makes the setting for the metrics measured automatically. When the input is completed, inputted data by the user are stored in the tag data repository and the settings for the metrics are delivered to the automatic data collection unit. The automatic

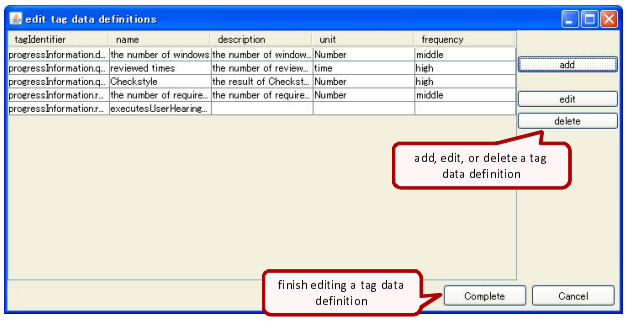

Figure 5. Edit screen of tag-data-definition

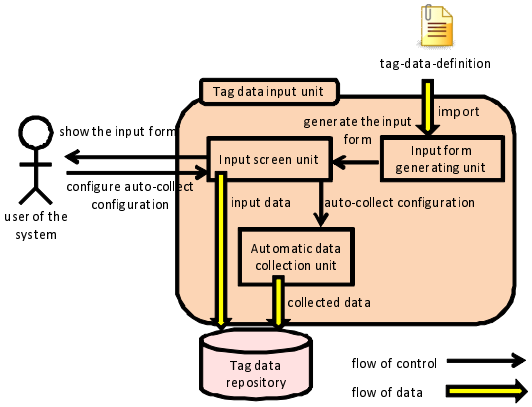

Figure 6. Structure of Tag data input unit

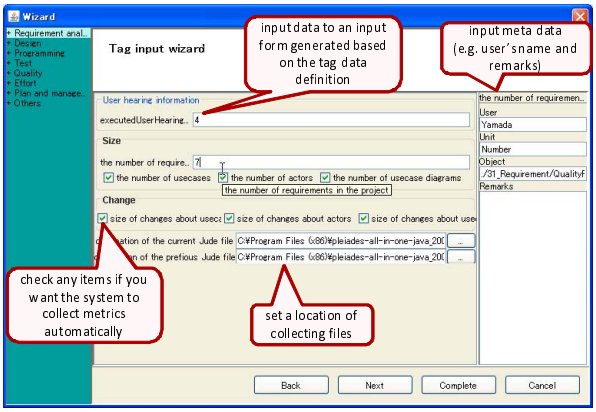

Figure 7. Input screen

data collection unit[1] measures the value data in accordance with the settings and the results are set in the form.

Figure 7 shows the input screen. The screen is automatically suggested by the Wizard and the metrics are shown in the order of the category of the software tag.

[1]We have implemented a measurement tool based on MASU[11]. It measures some metrics (ex. CK-metrics, LOC) from Java source codes.

```
<data name="the number of reviewing" source="null">
  <unit>Number</unit>↓
  <value>5</value>↓
  <date>2007/04/18 0:00:00</date>↓
  <user>Yamada</user>↓
  <obj>review document</obj>↓
  <remarks />↓
</data>←
```

Figure 8. Software tag in XML format

5) Tag data output unit: Tag data output unit outputs the software tag to the user. It shows the content of the software tag in the form of a table and outputs it as a XML file. Figure 8 shows the part of software tag written in XML format. It can be used by some software tag analysis tools [4] developed in StagE project.

IV. Case Study

A. Purpose

In order to evaluate the applicability of CollectTag, we applied CollectTag to the data collected from an actual software development. In the evaluation, we focused on the cost of generating the software tag: manual data collection and automatic measurement by CollectTag.

B. Target project

We used the actual software development data provided by "ITSpiral"[8]. IT Spiral is a collaborative project by nine universities and four industries in Japan to develop a common curriculum for teaching software engineering. IT Spiral ordered a curriculum coordination software system to several software companies. They requested the companies to deliver the software with all the related product and process data. There exists many empirical data from requirement definition, design, implementation, test and so on.

The project took the water fall model. Basically, the project adopted the object oriented approach and mainly used the UML modeling tool, JUDE[9]. Then, the code was implemented in Java language. It took two months from the beginning of the project to the end of design. Then, it took three months from the implementation to the end of the project. So, the total amount of the duration was five months. The amount of size of the data is about 450MB. Table V shows the list of the empirical data.

C. Collected elements of the software tag

Since we applied CollectTag to the data collected from the past project, we did not negotiate the contents of the software tag with the developer. So, we selected typical elements and the corresponding metrics. As the result, we selected 18 tag elements and 53 metrics for the project shown in Table VI. Table VI shows the category, no., tag element,

Table V
EMPIRICAL DATA COLLECTED FROM THE PROJECT

Process	Empirical data
Requirements	requirements specification, business flow diagrams, Screen list, use case diagrams, actor definition documents, quality requirements specification
Analysis & Design	class diagrams, collaboration diagrams, sequence diagrams, architecture specification, entity-class design specification, control-class design specification Boundary-class design specification
Implementation	source codes, API document, product repository, records of static check
Test	test plan documents
Project management	minutes of review meeting, project management reports, change management documents

metrics, collection method and value[2] for each metric. In the column of the collection method, "auto" means that the data are collected automatically and "manual" means that the data are collected manually.

D. Process of case study

The process of the case study is as follows:

Step 1: We defined the tag-data-definitions of the software tag used for this project using CollectTag. Here, we selected the elements in the software tag shown in Table VI. As described in Section III-B, CollectTag provides the pre-defined tag-data-definitions. Here, we added the 13 tag-deta-definitions whose corresponding metrics are labeled by * in Table VI.

Step 2: Using CollectTag, we inputted the value manually and automatically according to the input wizard. Inputting the data manually, we collected the necessary value from the corresponding sources (empirical data) shown in Table V. Inputting the data automatically, at the first input, we just set the file path to the corresponding element.

Step 3: We output the software tag of the project.

One of the authors conducted the Steps 1, 2 and 3. To record the time of Steps 1 and 2, we used a stop-watch. We inputted the data totally 30 times for the 5 months project, which means the interval of input is about 3 days.

In Step 1, we only made the additional tag-data-definitions for metrics collected manually by using the tag-data-definition unit, since the tag-data-definitions corresponding to the data collected automatically have already stored in CollectTag.

We evaluated the generation cost of a software tag in terms of the amount of time to generate tag data definitions, input tag data manually, and measure the metrics automatically.

E. Result

In Step 1, the amount of time we spent to define 13 tag-data-definitions was 9 min. and 50 sec. In Step 2, the amount of time to input the tag data in 30 times was 70 min. and 47 sec., the average was 2min. and 17 sec., and the maximum was 6 min. and 4 sec. Partial results for Step 2, input time and time of automatic measurement, are shown in Table VII.

[2]The final value at the end of the project.

Table VII
RESULTS

Date of measured	Input time	Time for automatic collection
2007/03/27	0:01:54	0:00:01
2007/03/30	0:02:54	0:00:00
2007/04/04	0:02:27	0:00:00
2007/04/06	0:01:45	0:00:01
	...	
2007/05/31	0:01:09	0:00:02
2007/12/12	0:06:04	0:00:41
2007/12/17	0:03:07	0:00:55
	...	
2008/02/20	0:02:34	0:00:52
2008/02/25	0:02:15	0:00:52
2008/02/28	0:02:01	0:00:49
Total	1:10:47	0:14:28
Average	0:02:17	0:00:28
maximum	0:06:04	0:01:18

F. Evaluation

In Step 1, in order to add 13 tag-data-definitions, we spent 9 min. and 50 sec. Though adding one tag-data-definition spent 46 sec. averagely, it was just conducted only once at first. So, compared to the time of Step 2, it was not so much cost, we consider.

Since the requirement analysis and design were conducted from March 27 to May 31, 2007, the source codes did not exist. Also, defects data were not registered to KAGEMAI. So, during the duration, automatic measurements were just for the elements of requirement and design categories and the time was about 1 sec. shown in Table VII.

On the other hand, the time for measurements became longer from Dec. 12, 2007. Especially, on Dec. 12, it spent about 6 min. Because the programming started at the time. In order to measure metrics from the source code, it is necessary to spend time to export all source codes from the Subversion repository and execute the several measurement tools. It was also necessary to set the file path to the corresponding element but the setting was just conducted only once. So, from Dec. 17, the time became shorter (about 3 min.).

The average time was 2 min. and 17 sec. Here, we consider the cost to generate the software tag in one month. Usually, in software development projects, the project manager asks the project members to submit some reports (daily or weekly) to grasp the progress situation. So, if we assume that the reports are submitted weekly and the data for the software tag are inputted in time for the weekly report submission, the total times of input data becomes 4. If the time for each software tag data input needs 2 min. and

Table VI
COLLECTED ELEMENTS OF THE SOFTWARE TAG IN THE CASE STUDY

Category	No.	Tag element	Metrics	Collection method	Value
Requirements	14	Scale[Transition]	Number of usecase diagram	auto	12
			Number of usecase	auto	25
			Number of actor	auto	7
			Number of defined-requirements*	manual	7
	15	Revisions[Transition]	Number of added usecase diagram	auto	12
			Number of added usecase	auto	37
			Number of added actor	auto	7
			Number of deleted usecase diagram	auto	0
			Number of deleted usecase	auto	12
			Number of deleted actor	auto	0
Design	16	Scale[Transition]	Number of screen*	manual	23
			Number of class diagram	auto	72
			Number of sequence diagram	auto	20
			Number of state machine diagram	auto	0
			Number of activity diagram	auto	2
	17	Revisions[Transition]	Number of added class diagram	auto	83
			Number of added sequence diagram	auto	23
			Number of added state machine diagram	auto	0
			Number of added activity diagram	auto	2
			Number of deleted class diagram	auto	11
			Number of deleted sequence diagram	auto	3
			Number of deleted state machine diagram	auto	0
			Number of deleted activity diagram	auto	0
Programming	19	Scale[Transition]	Lines of code	auto	26033
	20	Revisions[Transition]	Lines of Lines of added or deleted code	auto	92303
			Lines of added code	auto	62586
			Lines of deleted code	auto	29717
			Number of file modification	auto	2249
	21	Complexity	CK metrics.WMC (average)	auto	6.278
			CK metrics.NOC (average)	auto	0.739
			CK metrics.CBO (average)	auto	10.43
			CK metrics.RFC (average)	auto	13.00
			CK metrics.LCOM (average)	auto	10.96
			CK metrics.DIT (average)	auto	2.81
Test	22	Scale[Transition]	Number of unit test item*	manual	238
			Number of integration test item*	manual	75
			Number of system test item*	manual	508
	24	Density	Number of unit test item / Lines of code	auto	0.009
			Number of system test item / Number of usecase	auto	20.32
	25	Progress Status	Number of consumed system test item / Number of system test item	auto	1
			Number of consumed system test item*	manual	508
Quality	26	Review Status	Number of review*	manual	22
			Total review time*	manual	50.75
	27	Review Density	Number of review / Total review time	auto	0.43
	28	Review Effectiveness[Transition]	Number of defect found by review*	manual	529
	29	Defect Count[Transition]	Number of defects found by test	auto	19
	30	Fixed Defect Count[Transition]	Number of fixed defects	auto	19
	31	Defect Density	Number of defects / Lines of code	auto	0.00073
	32	Defect Detection Rate	Number of defects / Number of consumed system test item	auto	0.037
	33	Static Check Results	Number of check by *checkstyle**	manual	359
			Number of check by *findbugs**	manual	52
			Number of check by *pmd**	manual	387
			Number of check by *pmd-cpd**	manual	5

17 sec., the total time becomes 0.167 (hours) (=9 min. and 8 sec.). Considering one person-month is 160 hours (8hours $\times$ 20 days), the cost to input tag data becomes about 0.001(person-month). As the results, we consider that using CollectTag would be useful to generate the software tag at low cost.

G. Threats to Validity

There are some threats to validity in the case study.

In this case study, we added only 13 tag-data-definitions and so the cost to define them was not so high. However, in practice, much more tag-data-definitions might be added. For example, if we collect more than 100 elements, we would take much effort to add them manually. So, it is necessary to provide as many pre-defined tag-data-definitions as possible to the user.

As described in Section IV-F, most time of the automatic measurement spent to export source codes from the repository and execute analysis tools. In this case study, the final program size was about 26033 LOC. The more the size becomes, the much time would be needed for the automatic measurement. Hence, it is necessary to evaluate the cost of automatic measurement against the large scale software. Just for information, we applied analysis tools to an open source program whose LOC is about 210 thousand and it took about 7 minutes to calculate the same metrics in this case study.

Also, if the user of CollectTag has not enough knowledge

about the project, much more time would be needed. In this case study, the subject had enough knowledge about the project data. For example, he knew where the requirements specification, minutes of review meeting, the UML model files, the results of static checking and so on. So, he took only a second to get the necessary data to input CollectTag. In that sense, the user of CollectTag must be the person who knows the detailed of the target project.

V. Conclusion

This paper describes a system, named CollectTag, which supports to generate the software tag which makes software development visible to software purchaser (users).

CollectTag has versatility in the sense of allowing the user to define software tag definitions for his/her own project. It provides automatic measurement mechanisms to collect metrics for some elements of the software tag and reduces the cost of collecting data. Finally, it outputs the software tag in the XML format and provides a convenient software tag for some tools that analyze the tag.

As a case study, we have applied CollectTag to the actual software development project data and evaluated the usefulness of it from the viewpoint of cost to generate the software tag. As the results, we found that by using CollectTag, generating the software tag needs not high-cost.

We have to improve the functionality and usability of CollectTag. At first, we are going to enrich the pre-defined tag-data-definitions. Insufficient pre-defined tag-data-definitions urges the user to input additional tag-data-definitions. Next, we have to implement the display form for the user intuitively to understand the software tag anytime as the financial statements of company evaluations or inspection results of the complete medical checkup. But, it is necessary to collect enough data to set the standard or limiting value for each metric of each tag element. This can be achieved when the software traceability becomes widely used and the software tag is commonly used in the software development projects.

Finally, the evaluation of CollectTag is essential in the context of the actual software development projects. We are going to ask some companies that support the StagE project to use CollectTag and also release CollectTag from the Web site of the StagE project. Based on the feedbacks through the evaluations, we are going to improve the applicability of CollectTag.

As the StagE project, we will focus on making international/domestic standards of the software tag. With such standardization, the software tag is expected to be used in various software industries, where we think it will strongly promote participation and understanding of software development by purchasers. Also, to reduce adaptation cost of the software tag, we will delivery software tag support tools and the software tag guidebook which explains how to use the software tag. That would accelerate incorporating the software tag scheme in the industrial practices. Moreover, we will make a template of the contract document, considering software tag and legal issues of software development.

Acknowledgment

This work is being conducted as a part of the StagE project, The Development of Next-Generation IT Infrastructure, supported by the Ministry of Education, Culture, Sports, Science and Technology. We are grateful to the members of the StagE project. Also, we would like to thank the support of Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research (C) (No.20500033)

References

[1] S. R. Chidamber and C. F. Kemerer: “A Metrics Suite for Object Oriented Design”, IEEE Trans. Software Eng., 20, 6 pp. 476-493, 1994.

[2] http://checkstyle.sourceforge.net/

[3] http://findbugs.sourceforge.net/

[4] K. Fushida, J. Takata, T. Yonemitsu, Y. Fukuchi, S. Kawaguchi, and H. Iida: “A Framework and System for Planning and Tailoring Software Development Processes with Quantitative Management,” Proc. of the 4th International Conference on Project Management (ProMAC 2008), pp.286-294, 2008.

[5] http://www.gnu.org/software/gnats/

[6] K. Inoue: “Software Tag for Traceability and Transparency of Maintenance”, Proc. of 24th IEEE International Conference on Software Maintenance, pp. 476-477, 2008.

[7] ISO/IEC 12207: Systems and software engineering - Software life cycle processes. International Organization for Standardization, 2008.

[8] M. Barker and K. Inoue: “IT SPIRAL: A Case Study in Scalable Software Engineering Education”, Proc. of the 22nd Conference on Software Engineering Education and Training, pp. 53-60, 2009.

[9] http://jude.change-vision.com/jude-web/index.html

[10] http://www.daifukuya.com/kagemai/ (in Japanese)

[11] http://masu.sourceforge.net/

[12] Nikkei Business Publications, Inc.: “Survey Report on IT Investment and CTO in Japan”, Nikkei Information Strategy, 2008(in Japanese).

[13] Nikkei Business Publications, Inc.: “Second Survey of Japanese Software Projects”, Nikkei Computer, Dec. 1, pp. 36-53, 2008(in Japanese).

[14] http://pmd.sourceforge.net/

[15] M. Tsunoda, T. Matsumura, H. Iida, K. Kubo, S. Kusumoto, K. Inoue and K. Matsumoto: “Standardizing the Software Tag in Japan for Transparency of Development”, Proc. of 11th International Conference on Product Focused Software Development and Process Improvement, No.LNCS 6156, pp.220-233, 2010.