IEICE TRANS. INF. & SYST., VOL.E97–D, NO.4 APRIL 2014

PAPER Special Section on Data Engineering and Information Management

Visualization System for Monitoring Bug Update Information

Yasufumi TAKAMA[†a)], *Member and* **Takeshi KUROSAWA**[†*], *Nonmember*

SUMMARY This paper proposes a visualization system for supporting the task of monitoring bug update information. Recent growth of the Web has brought us various kinds of text stream data, such as bulletin board systems (BBS), blogs, and social networking services (SNS). Bug update information managed by bug tracking systems (BTS) is also a kind of text stream data. As such text stream data continuously generates new data, it is difficult for users to watch it all the time. Therefore, the task of monitoring text stream data inevitably involves breaks of monitoring, which would cause users to lose the context of monitoring. In order to support such a monitoring task involving breaks, the proposed system employs several visualization techniques. The dynamic relationship between bugs is visualized with animation, and a function of highlighting updated bugs as well as that of replaying a part of the last monitoring time is also proposed in order to help a user grasping the context of monitoring. The result of experiment with test participants shows that highlighting and replay functions can reduce frequency of checking data and monitoring time.

key words: *information visualization, Web mining, monitoring support, bug tracking system*

1. Introduction

Recent growth of the Web has brought us various kinds of text stream data, which includes bulletin board systems (BBS), blogs, and social networking services (SNS), etc. Such text stream data become important Web resources, from which we can obtain valuable information. Because of dynamic nature of text stream data, how to treat continuously arriving data is key issue for utilizing it.

It is often observed in our daily life including work, study, and private time that we have to check text stream data. For example, we often check Twitter and Facebook periodically in order to examine whether new / interesting information arrive or not. This paper calls the task of checking text stream data as monitoring task. As it is difficult for users to always check continually generated data, a monitoring task inevitably involves breaks, which makes monitoring a difficult task. That is, when users want to check online news, BBS, or Twitter while having a coffee break at office, they have to check data received during their primary jobs in a relatively short time. Furthermore, monitoring new data usually requires context information, e.g., which items were of interest in previous monitoring time, and the reason why those items attracted the users' attention. Therefore, way of presenting data with context information is important in order to support monitoring task.

Information visualization technologies are known to be effective for supporting human to access data. Various kinds of information visualization systems have been developed, and those main targets have been exploratory data analysis (EDA) [1], [2] and information retrieval [3]. On the other hand, to the best of our knowledge, less effort has been made for supporting monitoring task with information visualization technologies. Although peripheral displays [4]–[7] relate with monitoring task as will be noted in Sect. 2.1, those studies have not paid attention to the existence of breaks during monitoring task.

This paper proposes a concept of monitoring support for utilizing text stream data. Among various kinds of text stream data available on the Web, this paper focuses on bug update information, and proposes an information visualization system for supporting the task of monitoring it. Bug update information is managed by a bug tracking system (BTS), which continuously sends emails reporting bug update information to developers. Therefore, it is typical text stream data. Furthermore, in the case of open source software (OSS) such as Linux distributions, it includes a large collection of software applications, which are managed by different BTSs. Therefore, a developer of a certain software applications has to access multiple BTS for collecting bug update information relating with his/her work.

From the viewpoint of monitoring support, users of BTS, i.e., developers of OSS in this paper, have an important characteristic: various developers from all over the world, including employees in companies, researchers in universities / research institutions, and volunteers, take part in the development. Therefore, the development goes ahead while one is away from the development. Such a characteristic of OSS development make monitoring task more difficult. Therefore, bug update information of OSS is suitable as the target data for studying the concept of text stream monitoring support.

To support monitoring task of bug update information in OSS development, this paper proposes a visualization system that displays bug update information obtained from multiple BTSs. The proposed system visualizes the dynamic relationship between bugs with animation. To help a user grasping the context of monitoring, the system is equipped with a function for highlighting updated bugs as well as that of replaying a part of the last monitoring time. The effectiveness of the system is evaluated through an experiment

Manuscript received July 4, 2013.
Manuscript revised October 23, 2013.
[†]The authors are with the Graduate School of System Design, Tokyo Metropolitan University, Hino-shi, 191–0065 Japan.
[*]Presently, with Mitsue-Links, Co., Ltd.
a) E-mail: ytakama@sd.tmu.ac.jp
DOI: 10.1587/transinf.E97.D.654

with test participants. The experiment supposes the situation where they have to take a break during monitoring. The result shows most of participants could find interesting bugs after a break. In particular, it is shown that the highlight function has statistically significant effect on the efficiency of achieving the task.

This paper is an extended version of the paper published in the proceedings of an international conference [8], by adding detailed discussion on the concept of monitoring support for text stream data, analysis of user behaviors using the proposed system, etc. This paper has two contributions. First, monitoring task is proposed as a new task to be supported by information visualization. It is expected that new task will motivate many researchers to study new information visualization techniques and systems. Second, the system proposed in this paper can support OSS developers.

The organization of the paper is as follows. Section 2 introduces related work including topic detection and tracking from text stream data, peripheral displays, and bug tracking systems. Section 3 describes the proposed system, which is followed by Sect. 4 that shows experimental results.

2. Related Work

2.1 Peripheral Displays and Monitoring Support Systems

As monitoring of text stream data is a difficult task, visualization is expected to be useful for supporting it. However, most of information visualization systems aim to support EDA, which is an approach for analyzing data space from various viewpoints without having predetermined hypothesis. Whereas EDA is supposed to be a primary job for a user, monitoring task is supposed to be done between intervals of such primary jobs.

Peripheral displays (ambient displays) have been studied for providing important but non-emergent information without disturbing user's primary jobs [4]–[7]. Although this concept is expected to be useful for monitoring task, main target of existing systems is data with simple structure, such as stock price [6], weather information [6], [7], and electricity consumption [4].

Monitoring support for software development progress has been studied [9]–[11]. Jazz [9] visualizes information about other developers, such as online /offline and their progress, in an integrated development environment. Augur [10] visualizes the record of file modification for monitoring the progress of collaborative works. Ripley et al. [11] have employed 3D views for visualizing progress of development from the viewpoints of developers and files. However, these systems do not consider breaks during monitoring task.

The code_swarm [12] visualizes software project histories as non-interactive video. Although its non-interactive nature would be suitable for monitoring support, it is not designed for monitoring current progress of the development of software.

2.2 Bug Tracking Systems for Open Source Software

A BTS is a software application that helps programmers keep track of reported software bugs. It is used by developers and users of software to report bugs and to discuss its cause and how to fix it. When an update is occurred for a bug, the update information is sent to those involved in it via e-mail.

A BTS assigns a unique ID to each bug, and manages various kinds of information about bugs such as importance, involved persons, and status. A status of a bug indicates its progress toward fix. Although different set of statuses is used by different BTSs, this paper treats the following 5 statuses.

- Unconfirmed: Reported but not confirmed yet.
- New: Confirmed to be present.
- In Progress: Developer is working on it.
- Fixed
- Rejected: Unfixable, already reported, out of support, etc.

When a bug is reported for the first time, its status is regarded as "Unconfirmed." After the existence of the reported bug is confirmed, its status becomes "New." When a developer to fix the bug is assigned, its status becomes "In Progress." The "Fixed" is assigned to the bug when it is fixed. In the case of a minor bug, its status can be directly changed from "New" to "Fixed." If the bug cannot or will not be fixed for some reasons, it is rejected.

The importance of a bug is determined based on its impact and priority in fixing works. As will be noted in Sect. 3.2, this paper considers three levels of importance: high, medium, and low.

As a bug is updated dynamically, affects other bugs in complex manner, and involves many developers, tracking bug update information is not a trivial task. On the other hand, visualization techniques are known to be helpful for users to grasp large-scale, complex data space. From this viewpoint, several studies have applied visualization techniques to data managed by BTS [1], [13]. The Bug's Life [13] employs two kinds of visualizations, one of which is for providing overview and another is for detailed information about each bug. The metaphor of a watch is employed for visualizing detailed information about a bug. Tesseract [1] visualizes data from version management system and BTS in integrated manner. It consists of several panels, which include the panel for showing the activity change of a project, that for file information, and for information about developers and bugs. When a user operates on one of those panels, views of other panels are automatically updated based on the relationship among objects (bugs, developers, files, etc.).

Bug Maps [14] provides time series of defects in the classes of object-oriented systems with a set of interactive visualization techniques. Target data is collected from version control platforms such as CVS and SVN in addition

to BTSs. Knab et al. [2] have proposed interactive visualization approach for supporting analysis of data in issue tracking repository, which contains problems reports from a multi-year, multi-site software development project in the consumer electronics domain. It integrates multiple views such as pie chart for process step length and polymetric views for effort measures.

However, these systems do not suppose a monitoring task, in the sense that breaks inevitably occurred in the task are not considered explicitly.

3. Visualization for Monitoring Bug Update Information

3.1 Outline of Monitoring System

As noted in Sect. 1, this paper proposes a concept of monitoring support for utilizing text stream data. As one of its applications, an information visualization system targeting bug update information is also proposed. One of important characteristics of the monitoring task is that it inevitably involves breaks. That is, a user has to check text stream data within a short time and go back to their primary jobs. Furthermore, context information tends to be lost during breaks, which makes it difficult to interpret target data.

Two types of visualization techniques are mainly employed in the paper: animation for visualizing dynamic change of bug status and highlight for providing context information.

Animation is employed for visualizing the status of bugs and relationship with others. Using animation can help a user grasp target data within a short time. Another reason why we employ animation is because it would be suitable for secondary jobs like monitoring. By using animation, the initiative of interaction is given to a system. As a user can start monitoring by just watching a screen with less operations, it is expected to be suitable for a secondary job that should be done while taking work break. Details are described in Sect. 3.2.

Highlighting technique is employed for visualizing context information. By combining the replay of part of the last monitoring period, a user can recognize how the state of bugs changed during a break. Details are described in Sect. 3.3.

The proposed system consists of a data collection module, a data management module, and a monitoring interface. The data collection module continuously receives emails about bug update information from multiple BTSs, and extracts required information from contents of received emails. Extracted data are sent to the data management module, which organizes and stores bug update information. The monitoring interface receives the data to be visualized from the data management module. The data collection and management modules are implemented with Python. The interface is implemented as a Web application using Processing, HTML, CSS, and JavaScript.

Figure 1 shows a screenshot of the monitoring interface. The largest panel in the screenshot is a visualization panel, in which relationship among bugs are visualized. The detail of this panel is described in Sect. 3.2. The right-hand panel shows the detailed information about the bug selected in the visualization panel. The lower panel of the left-hand is a control panel, which is used for various operations, such as start / pause of monitoring, selection of dataset to be monitored, and setting of filtering conditions. The visualization panel is implemented as a Java applet with Processing, and other parts are implemented with JavaScript.

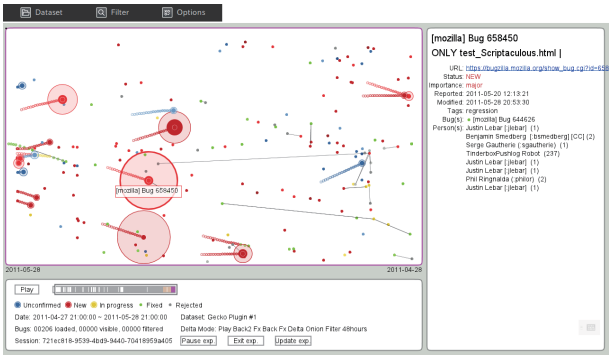

Fig. 1 Screenshot of the proposed system.

3.2 Visualization of Bug Status

The status of a bug changes as noted in Sect. 2.2, which leads to dynamic change of relationship among bugs. When monitoring bug update information, it is important to grasp such dynamic changes. The visualization panel visualizes the status of bugs and those relationships with animation.

Let the time period for monitoring (called monitoring period hereinafter) be $[t_{start}, t_{end}]$, i.e., bug update information reported between this time period is visualized. Usually, t_{start} is set to the ending time of the last monitoring task, and t_{end} is set to the current time. Let $t_{monitor}$ ($\in [t_{start}, t_{end}]$) be the time at which the status of bugs and those relationships are visualized. When monitoring is started, the visualization panel visualizes the relationship among bugs with animation, by changing $t_{monitor}$ from t_{start} toward t_{end}. When visualizing bug status at $t_{monitor}$, bugs newly reported at that time are appeared on the left edge of the visualization panel. The statuses of existing bugs are also updated. Details are described below.

According to the discussion in Sect. 2.2, this paper focuses on the following three factors, which are important for monitoring bug update information.

1. Status of a bug including update frequency and progress toward fix.
2. Persons involved in a bug.
3. Relationship among bugs.

Regarding (1), each bug is represented as a node in the visualization panel. The color of a node indicates the status of the corresponding bug. Figure 2 shows the color assignment, which is determined based on existing study [13] and

Status	High importance	Medium importance	Low importance
Unconfirmed	#729fcf	#3465a4	#204a87
New	#ef2929	#cc0000	#a40000
In Progress	#fce94f	#edd400	#c4a000
Fixed	#8ae234	#73d216	#4e9a06
Rejected	#babdb6	#888a85	#555753

Fig. 2 Color assignment to bug status.

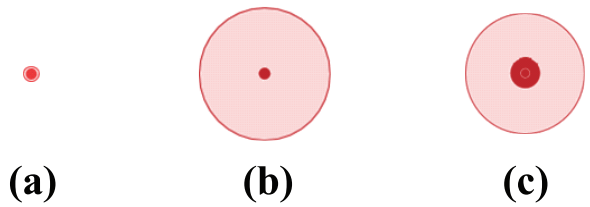

Fig. 3 Bi-cylindrical representation of nodes.

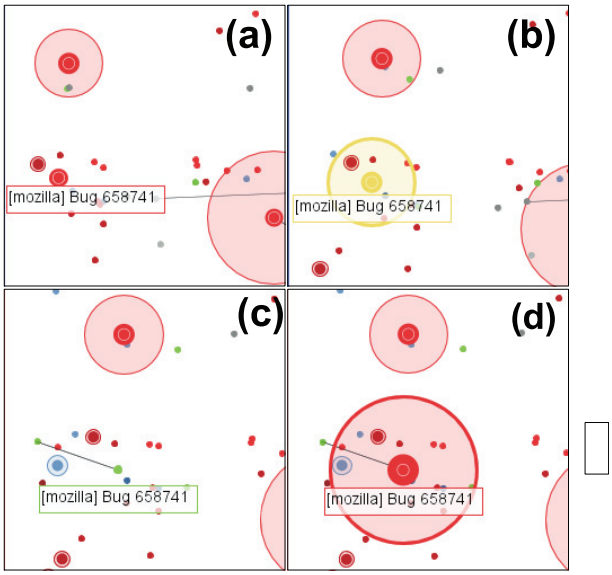

Fig. 4 Status change of a bug.

the color assignment commonly used for describing OSS development information. The importance of a bug is represented by its node's brightness: more important bug is represented as a brighter node.

A node is bi-cylindrical as shown in Fig. 3. It consists of 3 layers: core, an inner circle, and an outer circle. The layer between outer and inner circle is colored with faint red, and that between inner circle and core is colored with red. Although core is also colored with red, a blank is inserted between core and inner circle. The difference between radius of inner circle and core represents the progress toward fix, and the difference between radius of outer and inner circles represents its update frequency. Figure 3 shows nodes of different status. The difference between radius of inner circle and core is almost zero in Fig. 3 (a) and (b), while it is relatively large in (c). Regarding difference between outer and inner circle, the difference in Fig. 3 (a) is small, while it is large in Fig. 3 (b) and (c). Therefore, a node shown in Fig. 3 (a) indicates the corresponding bug is not updated frequently, and thus little progress. A bug corresponding to the node of Fig. 3 (b) is being updated frequently but little progress. On the other hand, the node of Fig. 3 (c) indicates the bug is also being updated frequently and going smoothly toward fix.

Figure 4 shows an example of the status change of a bug (ID:658741). Starting from "New" status (Fig. 4 (a)), its status transits to "In Progress" (Fig. 4 (b)) and "Fixed" (Fig. 4 (c)). However, its status is back to "New" as the bug fix was incomplete (Fig. 4 (d)).

To visualize the persons involved in a bug, this paper employs a swarm-like animation. According to the Gestalt principle of common fate, objects moved in a synchronized fashion are perceived as related by a human. The proposed method lets nodes move toward similar directions if those involved persons are similar.

When visualizing at $t_{monitor}$, a node newly appeared at that time is assigned to left edge of the visualization panel, and move to the right as time progresses. The position on vertical axis are determined so that nodes appeared at the same time cannot be overlapped each other. The angle θ_i^b, a bug b_i's moving direction, is determined as Eq. (1), where f_{ij} is the frequency of the person u_j being involved in b_i. The θ_j^u is the moving direction assigned to u_j, which is determined by applying SHA-256 (Secure Hash Algorithm 256-bit) to some unique ID of u_j, such as email address.

The velocity of b_i (v_i) is determined as Eq. (2).

$$\theta_i^b = \frac{\sum_j f_{ij}\theta_j^u}{\sum_j f_{ij}} \tag{1}$$

$$v_i = \frac{v_x(s_i)}{\cos\theta_i^b} \tag{2}$$

In Eq. (2), $v_x(s_i)$ is horizontal velocity, which is determined by s_i (status of b_i). A bug with status of "Unconfirmed," "New," or "In progress" moves slow, and rejected bug moves fast. A fixed bug has medium speed. This setting allows unimportant bugs go outside the visualization panel quickly.

Figure 5 illustrates how nodes form a swarm. When the same persons are involved in several bugs, those bugs have the same directions (Fig. 5 (a)). As time progresses, when another person, of which direction is indicated as an arrow pointing to upper right in Fig. 5 (b), is involved in 2 of those bugs, the direction becomes different from that of the remaining bugs (Fig. 5 (c)). Relationship between bugs is represented as an edge between those in the visualization panel. In this paper, an edge is drawn between nodes having some relationship, such as dependence and block relationship. The information about related bugs is obtained from BTS. However, it is difficult to find several groups of related bugs if related bugs are distributed over the panel, even though those are connected by edges. In order to make related bugs form a cluster, the direction and velocity of a node

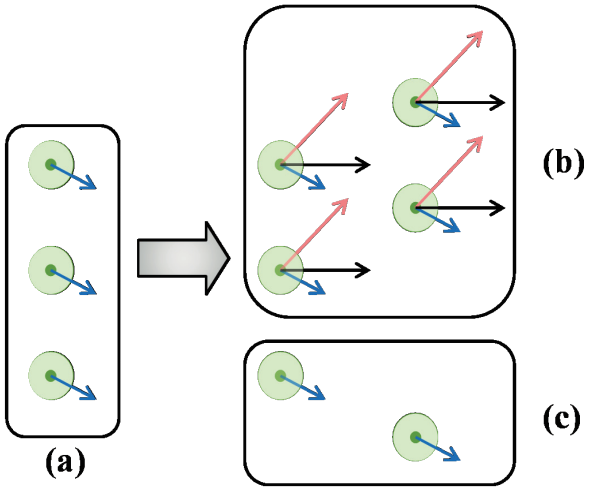

Fig. 5 Moving direction of nodes.

is modified as follows. From a set of bugs connected with edges (denoted as S), the following two bugs are selected.

- $b_p \cdots$ A bug, of which the corresponding node is located at the rightmost on the visualization panel in S.
- $b_v \cdots$ A bug having the most serious status in S.

The order of serious status is "In Progress" (the most serious), "New," "Unconfirmed," "Rejected," and "Fixed," which is determined from developer's viewpoint. If more than one bug have the most serious status, the bug (node) having the biggest inner circle among those is selected as b_v.

Let the position of a bug b_i on the visualization panel be x_i. If the distance between nodes of b_p and b_i is equal to or greater than the predefined threshold, the direction and velocity of b_i are modified by Eq. (3) and Eq. (4), respectively. If the distance is less than the threshold, b_i has the same direction and velocity as b_v. In Eq. (4), $\beta > 1$ is the coefficient (it is set to 2.0 in the experiment in Sect. 4).

$$\theta_i^b = \arctan(x_p - x_i) \tag{3}$$

$$v_i = \beta v_v \tag{4}$$

Figure 6 shows the formation of connected nodes. Five nodes are connected in the figure, among them blue node corresponds to b_p, and red node corresponds to b_v. While connected nodes were initially distributed over visualization panel (Fig. 6 (a)) because those appeared at different times, those are gradually forming a cluster on the panel (Fig. 6 (b), (c)). Showing related nodes forming a cluster can attract user's attention, which is expected to help users recognize the group of related bugs and its evolution.

3.3 Visualization for Context Information

As most of OSS developers are supposed to check BTSs between intervals (work breaks) of their primary jobs, a monitoring task inevitably involves breaks. It is supposed that developers are interested in bugs, of which status has been

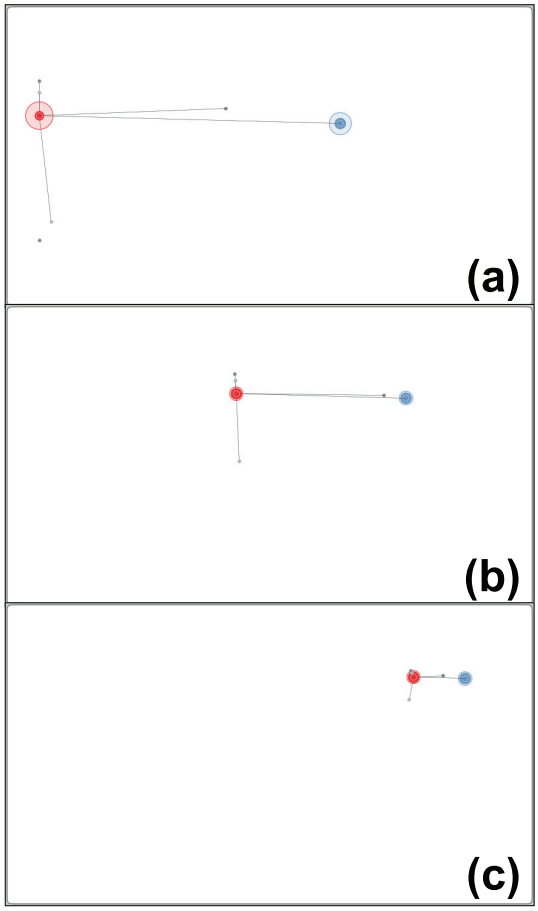

Fig. 6 Formation of connected nodes.

changed after the last monitoring period (i.e. while doing primary jobs). For example, bugs reopened several times are assumed to be serious [13]. In order to find such interesting bugs efficiently, it is important to know the difference between current situation and that of last monitoring period. In other words, keeping context beyond breaks is important for monitoring task.

From this viewpoint, this paper proposes the following functions for providing context information.

- Highlighting interesting bugs.
- Replaying animation of part of the last monitoring period.

The role of highlight function is to make interesting bugs stand out. As mentioned above, this paper defines an interesting bug as that of which status have been changed after the last monitoring period. In order to highlight such interesting bugs, this paper combines filtering function and onion skin visualization.

For highlighting interesting bugs, those without any updates after the last monitoring period are filtered out. In the visualization panel, nodes of filtered out bugs are represented with minimal size. In Fig. 7 (a), all nodes except lower-right one corresponds to filtered out bugs.

Figure 7 (b) shows onion-skin visualization, which is

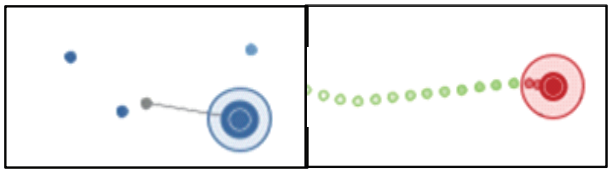

Fig. 7 Visualization of context.

used for visualizing a trajectory indicating past positions and status (color) of a bug. Nodes in trajectory part are displayed with minimal size. This is applied to bugs with one or more updates after last monitoring period. For example, it is seen from Fig. 7 (b) that moving direction has been changed twice, which means the change of involved persons. The change of color from green to red means its recent change of status from "Fixed" to "New."

As noted in Sect. 3.2, t_{start} of monitoring period is usually set to ending timestamp of the last monitoring period. Instead of that, when replaying part of the last monitoring period ($[t'_{start}, t'_{end}]$), t_{start} is set to within it. Replaying part of the last monitoring period is expected to be helpful for users to recall the situation of the last monitoring period, which would clarify the changes occurred after that. However, it would lead a user to misreading if a user confuses last and current monitoring periods. In order to avoid such confusion, border color of the visualization panel is changed between last and current monitoring periods.

4. Experiment

4.1 Experimental Settings

An experiment is conducted to evaluate the effectiveness of the proposed system for user's monitoring task. Although the proposed method employs several visualization techniques, the experiment focuses on the effects of highlighting function and replay on participants' performance. Test participants are asked to identify target bugs under a circumstance that simulates monitoring with breaks. In the experiment, target bugs to be identified by test participants are defined as those of which status are changed both in previous and target time periods. Focusing bugs, of which status changed in previous time period, means that test participants have to remind the monitoring context of the previous period.

Twelve graduate / undergraduate students in Engineering took part in the experiment with the following procedures.

1. Monitoring: Checks previous time period using the proposed system.
2. Break: Types English sentences (note that all participants are Japanese).
3. Monitoring: Checks target time period and find one of target bugs using the proposed system.

Table 1 Summary of datasets.

ID	BTS	Period	# of nodes (target)
A	Mozilla	2011.7.23-8.23	232(4)
B	Mozilla	2011.4.28-5.28	201(1)
C	GNOME, Freedesktop, Fedora, Ubuntu	2011.6.26-7.26	379(5)
D	Freedesktop	2011.8.22-9.22	161(1)

Table 1 shows the used datasets, which are collected from actual BTSs. A dataset A collects bug information about Mozilla Gecko's Core (components: Canvas:2D, Canvas:WebGL, SVG, Video/Audio), B collects those about Plugins of the Core, C is about Evince, Poppler, and cairo. A dataset D collects bug information about LibreOffice (Drawing and Spreadsheet). The number of target bugs in each dataset is also shown in Table 1.

Every dataset is divided into two time periods, such as follows. In step 1, a participant can check the previous time period only once.

- Target time period: The latest 24 hours.
- Previous time period: Other than target time period.

During a break (i.e. step 2), a participant engages in English type for 5 minutes, of which purpose is to simulate the situation where a participant loses the context of monitoring because of his/her primary job.

A test participant can finish step 3 when s/he can find one of target bugs. In this step, following two settings are considered regarding functions proposed in Sect. 3.3.

- Replay of part of previous time period: with or without
- Highlight (filtering and onion skin visualization): with or without

In the condition with replay, t_{start} is set to 48 hours before target time period. Therefore, total time period to be checked when using replay is 72 hours. With the combination of those two settings, 4 conditions are used in the experiment. The combination of datasets and conditions as well as its execution order are different from participant to participant, which satisfies the following conditions.

- Each participant is asked to conduct the experiment with all of 4 datasets by different conditions.
- Each dataset is analyzed 3 times by each condition.
- Each condition is executed 3 times in each execution order (from the first to the 4th).

Table 2 shows combination of conditions and datasets ({A,B,C,D}) and the order for each participant. In the table, 'R' and 'H' indicate the replay and highlight function are applied, respectively. Although there is the possibility that the results of the experiment contains order effects, the result of the paper is preliminary, and further analysis is still necessary in future.

In step 3, a participant is allowed to check (replay) target time period (and part of previous time period in case of replay condition) any number of times until s/he can identify

Table 2 Combination / order of conditions and datasets.

Participants	1st	2nd	3rd	4th
1	A-R	B	C-RH	D-H
2	A-RH	B-H	C	D-R
3	A-H	B-RH	C-R	D
4	A	B-R	C-H	D-RH
5	D	C-R	B-H	A-RH
6	D-R	C	B-RH	A-H
7	D-RH	C-H	B	A-R
8	D-H	C-RH	B-R	A
9	D	B-R	C-H	A-RH
10	D-R	B	C-RH	A-H
11	D-RH	B-H	C	A-R
12	D-H	B-RH	C-R	A

Table 3 Frequency of checking data in step 3.

	Without Highlight		With Highlight	
Participants	No-replay	Replay	No-replay	Replay
1	4	48*	2	1
2	3	1	5	2
3	3	5	5	2
4	9	5	2	2
5	32*	6	5	1
6	4	2	1	1
7	3	2	2	4
8	4	4	3	3
9	12	7	5	1
10	5	5	9	1
11	2	10	2	1
12	8	1	2	1
Mean	5.30	4.20	3.60	1.80
S.D.	3.10	2.71	2.29	0.98

a target bug.

4.2 Experimental Results

Table 3 shows the number of times participants checked data in step 3. In every condition expect dataset A with replay, all participants could identify a target bug. The participant 1 gave up before he could find any target bug from dataset A with replay condition. It is also observed that one participant (participant 5) had difficulty in achieving task in dataset D with no effect (i.e. without replay and highlight). In Table 3, the cases where a participant failed or had difficulty in achieving task are marked as *, which shows they checks the data too many times compared with others. Regarding these two cases as outliers, we analyze the result by excluding participants 1 and 5 in the rest of this subsection.

It is seen from Table 3 that frequency of checking data tends to be small when replay and highlight function are used. Table 4 is the result of two-way ANOVA for evaluating the effect of replay and highlight function, which shows that the highlight function has a statistically-significant positive effect on reducing the frequency of checking data.

Table 5 shows time (seconds) spent on monitoring task in step 3. The cases where a participant failed or had difficulty in achieving task are marked as *. Table 6 shows the result of two-way ANOVA for evaluating the effect of replay and highlight function on monitoring time. Although no statistically-significant effect is observed, monitoring time tends to be short with highlight function. On the other hand, monitoring time with using replay tends to be long, of which the reason is that time period to be checked is 72 hours when using replay, whereas that without replay is 24 hours only. From the viewpoint of time required for viewing the data, it takes about 3.6 seconds with using replay, and about 1.2 seconds without it. However, Table 5 also shows that increase of monitoring time can be suppressed by combining replay with highlight function.

Table 4 Effect of replay and highlight on frequency of checking data. SS: sum of squares, df: degrees of freedom, MS: mean square.

	SS	df	MS	F-value	P-value
Subject	56.23	9	6.25		
A:Replay	21.03	1	21.03	3.426	0.0972
Error[AS]	55.23	9	6.14		
B:Highlight	42.03	1	42.03	6.078	0.0358*
Error[BS]	62.23	9	6.91		
AB	1.23	1	1.23	0.190	0.6732
Error[ABS]	58.03	9	6.45		
Total	295.98	39			

Table 5 Monitoring time (seconds) in step 3.

	Without Highlight		With Highlight	
Participants	No-replay	Replay	No-replay	Replay
1	17	544*	6	21
2	12	6	43	40
3	13	37	34	26
4	130	100	10	16
5	212*	58	20	9
6	24	25	6	17
7	12	23	9	49
8	18	42	17	31
9	45	88	17	9
10	58	20	6	39
11	11	133	12	7
12	35	6	5	4
Mean	32.3	51.8	19.2	20.5
S.D.	34.3	40.4	13.4	14.7

Table 6 Effect of replay and highlight on monitoring time (seconds). SS: sum of squares, df: degrees of freedom, MS: mean square.

	SS	df	MS	F-value	P-value
Subject	7572.63	9	841.40		
A:Replay	1113.03	1	1113.03	2.430	0.1535
Error[AS]	4122.23	9	458.03		
B:Highlight	4862.03	1	4862.03	3.069	0.1137
Error[BS]	14257.23	9	1584.14		
AB	855.63	1	855.63	1.257	0.6732
Error[ABS]	6125.63	9	680.63		
Total	38908.38	39			

4.3 Analysis of Participants' Behaviors

After the experiment, we asked the participants to answer questionnaires regarding their impression about the functions of filtering, onion skin visualization, and replay. This subsection analyzes participants' behaviors using the system based on the results of questionnaires and log data of the system.

Regarding filtering function, positive opinions were

obtained from 8 participants, such as follows.

- It makes it easy to see changes of node's status.
- It is convenient when I want to check dense zone of nodes.

On the other hand, one participant had a negative comment that it is not necessary for performing this task.

Eleven participants had positive opinions for the onion skin visualization, such as follows.

- Trajectory of a node makes it easy to watch the change of its color.
- Showing both color and direction changes is attractive, and useful for finding a node of which status is updated.

It is noted that the participant who had negative opinion on filtering function had positive opinion on the onion skin visualization. He said the onion skin visualization is useful for examining status change of bugs during target time period.

On the other hand, two participants had negative opinions that it makes the visualization panel difficult to see when onion skin visualization is applied to many nodes at the same time.

Regarding the replay, 4 participants had positive opinions, whereas 8 participants had negative opinions. An example of positive opinion is that it is useful to find a node frequently updated within previous monitoring period. Typical negative opinion, which is obtained from 5 participants, is that it is not clear when entering into target time period from previous time period. This opinion corresponds to the fact that 4 participants regarded a bug of which status has changed only in previous time period as a target bug under replay setting. This problem could be solved with minor modification, such as changing background color when entering into target time period.

It was observed that participants' behaviors during monitoring task are modeled as the following process.

1. Find candidate nodes that might be target bugs.
2. Examine candidate nodes in detail.

When participants tried to find candidate nodes in step1, they had to focus on small number of nodes from vast amount of nodes displayed on the visualization panel, by using some viewing strategies. As for the viewing strategy, some participants divided their focused area on visualization panel into several regions. For example, the participant 5 divided the visualization panel into 4 regions, and focused on one of those regions one by one. The participant 12 firstly focused on the left edge of the visualization panel, and then gradually moved his attention to the right part.

Both of filtering and onion skin visualization could also help participants' task in step1. When the status of a bug is changed, the size of the corresponding node is drastically changed by the filtering functions. Therefore participants could easily notice such nodes. Trajectory of a node visualized by onion skin visualization also makes it easy for participants to find candidate nodes. Most of participants used both or either of these functions. When they could utilize a function, they had positive opinions to it.

Interestingly, the participants seldom moved mouse cursor over nodes. The average number of unique nodes they pointed until finding a target node is less than 9 for every condition. This indicates the system has initiative of interaction, which is one of our design policy as noted in Sect. 3.1.

In step 2, the onion skin visualization was utilized by most of the participants, which corresponds to the fact that 11 participants had positive opinions on it. When the onion skin visualization could not be used, the participants had to use replay for confirming the status change of a candidate nodes, which increased both the frequency of checking data and monitoring time. Although it required additional monitoring time, replay function was useful especially when it was used together with the highlight function, which is confirmed by the fact that 10 participants could find target bugs by checking target data only once or twice as shown in Table 3.

4.4 Discussion

As noted in Sect. 3.2, the proposed system aims to support developers examining bug update information in terms of bug status, involved persons, and relationship among bugs within a short time period. Among them, the task of the experiment in this section focuses on status change of bugs. This subsection discusses other two factors. When we developed the system, we asked an OSS professional developer and an evangelist in a company to use the system and to give us comments. It was observed that when monitoring bug update information, they checked involved persons. In order to make it easy to focus on specific person, they suggested that the function for filtering nodes with specified person. Considering this comment, the filtering function of the proposed system, which is described in Sect. 3.3, can specify a person as one of filtering conditions. Conditions which can be specified as filtering condition are involved person, status, importance, dataset, keyword, and with/without update during specified period. The last condition is also used for highlighting function as noted in Sect. 3.3. Once bugs relating with specific person are found with filtering function, it is easy to track those bugs based on moving direction.

Regarding relationship among bugs, edges were found to be informative to them. Furthermore, they also suggested the possibility of providing more information with an edge. For example, the type of relationship between bugs such as dependent and block relationship could be represented as the color or width of an edge. Although the current version of the system does not consider the type of relationship, it would be possible to extend the system to represent more information with edges. However, as providing more information could increase the cognitive load of a user [15], feasibility study should be conducted in terms of trade-off between information richness and cognitive load.

5. Conclusions

This paper proposed the concept of monitoring support for utilizing text stream data, and an information visualization system targeting OSS bug tracking systems. As the task of monitoring text stream data inevitably involves breaks, the proposed system employs the replay of part of the last monitoring period and the function of highlighting interesting bugs for providing users with context information. The experiment with test participants was conducted using data collected from actual BTSs, and the result shows the effectiveness of the proposed system. Although the system is designed for monitoring bug update information, the proposed concept of visualization for supporting monitoring tasks could be applied to various kinds of text stream data including online news, BBS, and microblogs. Our future work therefore includes development of monitoring support systems for such kinds of text stream data, by introducing additional processing such as topic extraction and other kinds of visualization techniques such as tag cloud.

Acknowledgments

This work was supported in part by a grant from National Institute of Informatics.

References

[1] A. Sarma, L. Maccherone, P. Wagstorm, and J. Herbsleb, "Tesseract: Interactive visual exploration of socio-technical relationships in software development," 31st International Conference on Software Engineering (ICSE'09), pp.22–33, 2009.
[2] P. Knab, B. Fluri, H.C. Gall, and M. Pinzger, "Interactive views for analyzing problem reports," ICSM2009, pp.527–530, 2009.
[3] M.A. Hearst and J.O. Pedersen, "Reexamining the cluster hypothesis: Scatter/gather on retrieval results," Proc. SIGIR96, pp.76–84, 1996.
[4] L. Bartram, J. Rodgers, and K. Muise, "Chasing the megawatt: Visualization for sustainable living," IEEE Comput. Graph. Appl., vol.30, no.3, pp.8–14, 2010.
[5] C. Parnin and C. Gorg, "Design guidelines for ambient software visualization in the workspace," 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, pp.18–25, 2007.
[6] C. Plaue, T. Miller, and J. Stasko, "Is a picture worth a thousand words? An evaluation of information awareness displays," Proc. Graphics Interface, pp.117–126, 2004.
[7] T. Skog, S. Ljungblad, and L.E. Holmquist, "Between aesthetics and utility: Designing ambient information visualizations," INFOVIS'03, pp.233–240, 2003.
[8] Y. Takama and T. Kurosawa, "Visualization for monitoring support and its application to bug tracking system," ISCIIA2012, no.18, 2012.
[9] S. Hupfer, L. -T. Cheng, S. Ross, and J. Patterson, "Introducing collaboration into an application development environment," CSCW'04, pp.21–24, 2004.
[10] J. Froehlich and P. Dourish, "Unifying artifacts and activities in a visual tool for distributed software development teams," ICSE'04, pp.387–396, 2004.
[11] R.M. Ripley, A. Sarma, and A. van der Hoek, "A visualization for software project awareness and evolution," VISSOFT'07, pp.137–144, 2007.
[12] M. Ogawa and K.L. Ma, "Code_swarm: A design study in organic software visualization," IEEE Trans. Vis. Comput. Graphics, vol.15, no.6, pp.1097–1104, 2009.
[13] M. D'Ambros, M. Lanza, and M. Pinzger, " "A Bug's Life" visualizing a bug database," VISSOFT'07, pp.113–120, 2007.
[14] A. Hora, N. Anquetil, S. Ducasse, M. Bhatti, C. Couto, M.T. Valente, and J. Martins, "Bug maps: A tool for the visual exploration and analysis of bugs," CSMR2012, pp.523–526, 2012.
[15] R. Mazza, "31. Memory," in Introduction to Information Visualization, pp.33–35, Springer-Verlag, 2009.

Yasufumi Takama received the B. S. Degree, M.S. Degree, and Dr. Eng. Degree from University of Tokyo, Japan in 1994, 1996, and 1999, respectively. From 1999 to 2002 he was a Research Associate at Interdisciplinary Graduate School of Science and Engineering, Tokyo Institute of Technology in Japan. Since 2005, he has been an Associate Professor at Faculty of System Design, Tokyo Metropolitan University, Japan. His current research interest includes information visualization, data mining, and Web intelligence. He is a member of IEEE, JSAI (Japanese Society of Artificial Intelligence), IPSJ (Information Processing Society of Japan).

Takeshi Kurosawa received B. S. Degree and M. S. Degree from Tokyo Metropolitan University, Japan in 2009 and 2011, respectively. Since 2011, he has been working at MitsueLinks Co., Ltd., Tokyo, Japan. His research interest includes information visualization and software development support systems.