Formal Notations and Terminology for Users' Feedback and Its Specialization for Interactive Fault Localization

Gergő Balogh and Péter Soha
Department of Software Engineering, University of Szeged, Szeged, Hungary

Keywords: Users' Feedback, Terminology, Source Code Analysis, Fault Localization.

Abstract: The knowledge of users is utilized in several aspects during software engineering-related tasks and research, such as UX and usability testing, code review sessions, and interactive debugging tasks, to name a few. Anyone who wishes to work with such a system has to face two challenges. They have to evaluate the proposed process and implement or integrate it into their workflow. This paper aims to aid these endeavors by providing a lingua franca for the stakeholders to define and express the expected users' feedback and the reaction they give to them. Our goal is not to evaluate all users' feedback-related results or to solve their testability and applicability. Nevertheless, our findings will support the resolution of these issues. We provide a formal terminology, which allows the stakeholders to specify their feedback instances, the items, and the actions.

1 INTRODUCTION

1.1 Ground Terms

Before we can discuss any issues, challenges, or achievements, we have to define the very basics of our terminology. Because in our opinion these terms are the part of the common knowledge, moreover essential to understand the further parts, we emphasize them at the beginning op this paper. The central concept in our paper is *feedback*, which is information about reactions to a product, a person's performance of a task, among others, which is used as a basis for improvement. The feedback can be collected with a *feedback system*, for example, the customers' experience with the product they use. In these systems, there are two kinds of active participants. They are the *user* and the *stakeholder*. For example, a user can be a developer who gives a co-worker a code review (i.e., feedback). The stakeholder uses this information to improve the subject.

Feedback. The transmission of evaluative or corrective information about an action, event, or process to the original or controlling source.

Feedback System. A physical or theoretical system that can collect, evaluate, and process a particular type of feedback.

User. An actor who provides feedback about the subject of the feedback system.

Stakeholder. Any other participants related to the feedback system besides the user. A person with an interest or concern in the outcome or the utilization of the feedback system.

1.2 Motivation

The knowledge of users is utilized in several aspects during software engineering-related tasks and research. End-users could express their opinions about a software system, which developers could use to enhance the UX of that system. A comprehensive analysis of similar methods is presented in the surveys by Hassenzahl et al. (Hassenzahl and Tractinsky, 2006) and Law et al. (Law et al., 2009). Programmers give feedback about the code quality (readability, maintainability, etc.) to each other so that they can speed up maintenance tasks, for example, Thongtanunam, P. et al. (Thongtanunam et al., 2015). McIntosh et al. (McIntosh et al., 2016) presented several techniques to improve the quality of the code. A researcher could integrate developers' knowledge into various methodologies, like in the case of interactive fault localization (iFL, (Horváth et al., 2020; Gong et al., 2012)). To aid the interactivity, various tools have been presented, such as VIDA (Hao et al., 2009)

Balogh, G. and Soha, P.
Formal Notations and Terminology for Users' Feedback and Its Specialization for Interactive Fault Localization.
DOI: 10.5220/0011317800003266
In *Proceedings of the 17th International Conference on Software Technologies (ICSOFT 2022)*, pages 226-233
ISBN: 978-989-758-588-3; ISSN: 2184-2833

by Hao et al. or STAD (Korel and Laski, 1988) by Korel and Laski.

Currently, each stakeholder uses their notations and terminologies to describe the feedback given by the users, the items they are associated with, and the action taken in response to them. We did not find any common terminologies for these feedback-based systems. At the same time, any researcher or developer who wishes to work with these systems is going to face two challenges. They will have to evaluate the proposed process and integrate it into their workflow. These issues are more apparent when someone has to transfer findings from one medium to another. For example, a semi-formal or informal description of the feedback system is sufficient when presented in a scientific paper. However, it could be vague when stakeholders want to utilize it as part of real-life workflows (e.g.: as implemented features in software systems or executable action during code reviews).

2 CONTRIBUTIONS

Our main contributions are the following:

- We provide the first version of a formal terminology, which could be utilized to define and describe the various feedback-related systems and methodologies (Section 3 and Section 4).
- Several examples illustrate the usage of the proposed terminology, in which we describe already published fault localization feedback systems with the notations from our terminology (Section 5).

3 FORMALIZATION OF USERS' FEEDBACK

The usage of user's feedback as mentioned above has several common properties and components. All of them define some subject items on which the users express their opinion; then this feedback is utilized to decide which action should be executed as the next step of the process. The following section will formalize these three main constituents of feedback systems: the items, the feedback, and the effects.

In the case of usability testing, the items are the features or functions of the system under test. During code review sessions, developers will judge the quality of various code chunks. Finally, the iFL algorithm would collect opinions about source code elements of various granularity, like methods.

The methodologies and algorithms may group some of these items and opinions based on their common properties or required actions. A negative code review usually causes the creation of a new issue about the problem. A low UX score could trigger further feature improvements. Finally, the annotation of the faulty code element(s) will terminate the fault localization process.

3.1 Subject Items

Since the primary subjects in our field are software systems, we define the current version of the terminology to support the analysis of those. Let us denote the set of code elements or other components of the system with S. Developers or users could give feedback to express their thoughts and feelings about any $s \in S$ subject.

Researchers and other stakeholders often group those items during analysis. This practice, besides reducing the cost of the process, also simplifies it by making the suspected or required connections implicit. In our terminology, we will use the $\phi : (s \in S) \mapsto (l \in L)$ function to assign labels to subjects. L is the set of all labels, while $\phi(s) \in L$ is the label of s. This labeling will be used to simplify the handling of similar subjects and hence the specifications of the feedback system. Please note that, if the designed feedback system requires it, stakeholders can choose to use the trivial labeling (every subject assigned to its own label), or compound labels, like sets, to express grouping based on multiple aspects. For example, in the case of iFL, a compound label could express the feature- and structure-related grouping of the methods.

3.2 The Feedback

There are three available modes to collect users' feedback. They either have to choose from a set of predefined options, express their thoughts freely, or use both modes simultaneously. During the postprocessing phases of the free mode, stakeholders will assign various options to the freely expressed thoughts to make them more manageable; hence in any of the three modes, a set can be specified that collects the accepted opinions of the feedback system. In our terminology, an option is an atomic description of the thoughts and feelings about a subject item or a set of items. We use the phrase "opinion" in a broader sense; hence it can encode sentiments, voluntary and involuntary reactions, or any other information originating from the users.

Based on the previously described properties of feedback-systems, we define an instance of feedback as a function $f : (l \in L' \subseteq L) \mapsto (o \in O' \subseteq O)$, which will assign opinions (o) to labels (l) of subject items. A feedback system should specify the sets of opinions (O) and feedback instances (F). A feedback instance may not assign any opinion to some labels or use all the opinions. We denote this with L' and O' subsets.

Similar to subject items, stakeholders tend to group feedback instances to speed up the subsequent steps of the process and to express suspected or expected connections. We use the term feedback-type ($\mathcal{F}$) to denote these groupings. Informally, a feedback type is a set of preset feedback instances.

$$\begin{aligned} &\mathcal{F} := \{f_1, f_2 \dots\} && \text{a feedback-type} \\ &f_i = \left\{(l_1^{(i)}, o_1^{(i)}), (l_2^{(i)}, o_2^{(i)}), \dots\right\} && \text{feedback instances} \\ &f_i \in F \end{aligned}$$

The concept of feedback types provides a new layer of abstraction over the feedback instances. Stakeholders may use the trivial case by simply naming the feedback instances accepted by the designed feedback system. In these situations, all feedback-types will contain a single feedback instance.

The above definition of set-based feedback types allows for the expression of the relation between various types utilizing set theory tools and notations. For example, consider the common feedback instances and types of scientific paper reviews. There are four different feedback possible: “strong accept”, “weak accept”, “weak reject”, and “strong reject”. We could define four distinct feedback-types (the trivial case), but we could also specify two types according to whether the feedback instance accepts or rejects the paper. Furthermore, it is also possible to express the reviewer’s confidence by introducing two more feedback types for strong and weak feedback instances.

The feedback types could be used to express quantization. In the case of iFL, researchers could define feedback instances to denote the level of confidence about whether the inspected code-elements are faulty or not (“faulty with confidence level 0”, “faulty with confidence level 1”, ...). Feedback types for low and high confidence could be specified.

3.3 The Effects of the Feedback

The feedback system will perform some actions in response to the given users’ feedback. In software engineering, feedback instances or types can be used to change the software system or any properties of industrial or research processes. We call these the effect of the feedback.

The complexity of these actions could be practically infinite. We can address these issues in two ways: the perfectionistic- and the pragmatic view. In the first case, our terminology has to encompass all the details of any possible actions anyone could execute in response to any feedback instances. In comparison, the pragmatic view will not overcrowd the terminology with details that are mostly unused in the feedback system related to software engineering. By design choice, our terminology uses the pragmatic view.

Based on our current understanding and experiences, a sufficiently complex definition of effects is the following. The effect $e(f)$ of a feedback instance f is a sequence of actions $(a_1, a_2, \dots)$, which perform the desired task if applied in the predefined order based on the labels and the assigned opinion.

$$\begin{aligned} &e(f) := (a_1(l_j, o_j), a_2(l_k, o_k), \dots) && \text{the effect} \\ &f = \left\{(l_1, o_1), (l_2, o_2), \dots\right\} && \text{feedback instance} \\ &a_i \in A && \text{an action} \end{aligned}$$

In the formula above, A denotes the set of all possible actions executed by the designed feedback system.

The sequence of actions has several special properties. An action has to be specified for each label-opinion pair of the feedback, but the specification may prescribe the “no operation” which does nothing. This restriction is necessary to preserve the unambiguity of the feedback system. An action with the same parameters could occur more than once, and the same label-opinion pair could be passed to several actions in the sequence. This property allows the expression of post- and preprocessing steps and to use actions with the persistent global state.

Our terminology allows actions to have arbitrary complexity. An action could store and retrieve auxiliary information (this could depend on the feedback instance or type), it may be ignoring its input parameters, etc.

The effect $e(\mathcal{F})$ of a feedback-type $\mathcal{F}$ is the sequence of effects and actions. These effects make it possible to give a “similar reaction” to feedback instances with the same type. To encourage stakeholders to design a clean feedback system, our terminology only permits that effects that have their instances are in the feedback type.

$$\begin{aligned} &e(\mathcal{F}) = (\alpha_1, \alpha_2, \dots) && \text{the effect} \\ &\alpha_i \in A \cup \{e(f) : f \in \mathcal{F}\} && \text{action or instances' effect} \end{aligned}$$

The effect of a feedback type could contain any actions or effects of its instances in an arbitrary order. Stakeholders could choose to include actions that ignore their attributes or supply them with any accepted label-opinion pair. The substitution of parameters with constant values for actions is only permitted in the cases of feedback types' effects.

3.3.1 Phase Structure of the Effects

The inner structure of the effects highly depends on the challenges that the feedback system wishes to be addressed. Despite this, we propose several guidelines for it. These guidelines are meant to encourage the creation of a more straightforward and easier-to-understand system. We propose that the effects of the feedback instances and types consist of three phases: a preprocessing, a core, and a postprocessing phase, in that particular order. Furthermore, we suggest that the pre- and postprocessing phases of feedback types contain only actions, while their core is a mixture of instances' effects and actions. If necessary, these phases could be partitioned into further sub-phases.

$$e(f) = (\underbrace{\operatorname{seq}A}_{\text{preprocess}}, \underbrace{\overbrace{\operatorname{seq}A}^{\text{sub-phase 1}}, \overbrace{\operatorname{seq}A}^{\text{sub-phase 2}}, \overbrace{\dots}^{\dots},}_{\text{core}} \underbrace{\operatorname{seq}A}_{\text{postprocess}})$$

$$e(\mathcal{F}) = (\underbrace{\operatorname{seq}A}_{\text{preprocess}}, \underbrace{\overbrace{\operatorname{seq}A \cup E}^{\text{sub-phase 1}}, \overbrace{\operatorname{seq}A \cup E}^{\text{sub-phase 2}}, \overbrace{\dots}^{\dots},}_{\text{core}} \underbrace{\operatorname{seq}A}_{\text{postprocess}})$$

$$E = \{e(f) : f \in \mathcal{F}\} \text{ set of instances' effects}$$

To simplify the following formulas, we introduce the seq operator, which generates an arbitrary sequence from some or all items of the specified set. Furthermore, if seq is used inside parentheses, like $(\operatorname{seq}A, \operatorname{seq}B)$, it means that we concatenate the (two) generated sequences.

3.3.2 Kinds of Effects

The grouping of effects and actions depends on the goals of the feedback system, but our terminology mandates assigning at least one of the following kinds to them. An action or effect could be assigned to more than one kind.

Subject-related. Effects and actions of this kind will modify the subject items, their properties, or their context. For example, a negative code review could eliminate various code elements, which shrinks the set of subject items for the next iteration.

Process-related. These effects and actions control the flow of the processes. For example, annotating the faulty method could cause the fault localization process to terminate.

Meta. Effects and actions that act on their feedback system have to be assigned the 'meta' kind. They could introduce new feedback instances or change the definition, hence the behavior of effects and actions. For example, the "I do not know." and the "None of the above." feedback instances could cause the set of opinions to expand for further iterations.

4 GENERALIZED NOTATION OF FEEDBACK-SYSTEMS

In this section, we elaborate on a proposed notation for the previously introduced terminology. Although, the mathematical formalism used in previous sections is adequate to define a feedback system, a specialized generative syntax could convey the same meaning in a more condensed and presumably easier to understand form. To explain the various aspects of this notation, we utilize the terminology of the `syntax` LaTeX package (Wooding,) and the ISO/IEC 14977:1996 standard (ISO,). To understand this section without expert knowledge about the standard mentioned above, only to most common notations were used.

Atomic (from the viewpoint of this terminology) constituents, like labels, actions, and opinions, are highly dependent on the field of interest. We suggest using the standards or the well-known notations to define these primitives.

Feedback instances are written as a set of assignments, which marks the label and the opinion it is mapped to (Grammer 4.1).

⟨*feedback instance*⟩ ::= '`{`' ⟨*assignments*⟩ '`}`'
⟨*assignments*⟩ ::= ⟨*assignment*⟩ ['`,`' ⟨*assignments*⟩]
⟨*assignment*⟩ ::= ⟨*label*⟩ '→' ⟨*opinion*⟩

Grammer 4.1: Notation for feedback instances.

Please note that instances do not have a name in order to prevent confusion about the role of feedback instances and types. If the feedback system requires the naming of the instances, they have to wrap them into feedback types (Grammer 4.2). A feedback type is denoted as a set of instances with a name marked in its prefix.

⟨*feedback-type*⟩ ::= '`@`' ⟨*name of type*⟩'`{`' ⟨*instances*⟩ '`}`'
⟨*instances*⟩ ::= ⟨*feedback instance*⟩ ['`,`' ⟨*instances*⟩]

Grammer 4.2: Notation for naming and grouping feedback instances.

Empty feedback-types and instances are discouraged to prevent confusion. If marking "no feedback given" is required, stakeholders should use "NONE". It represents a special feedback type, with several subtypes (subsets) or instances representing the various scenarios when the user cannot give feedback. The effect of "NONE" contains only actions, which do nothing, denoted with "NOP" (short for "no operation"). This suggestion does not prevent the system from ignoring any other feedback types or instances, i.e., does nothing in response. The proposed notation contains a predefined effect, with the name "IGNORE". By default, the "IGNORE" effect is assigned to the feedback-type "NONE".

The feedback instance in which an effect is assigned has to be marked explicitly (Grammer 4.3). However, our proposed notation allows us to define the effects before they are assigned to feedback. Hence they could be referenced by name and assigned to multiple feedback. To avoid unused effects, it is strictly forbidden to define them without assigning at least one feedback.

⟨*effect of feedback instance*⟩ ::= ⟨*feedback instance*⟩ '→' (⟨*effect definition*⟩ | ⟨*effect reference*⟩)

⟨*effect of feedback-type*⟩ ::= ⟨*feedback-type definition*⟩ '→' (⟨*effect definition*⟩ | ⟨*effect reference*⟩)

⟨*effect definition*⟩ ::= ['@' ⟨*effect name*⟩] ⟨*effect*⟩

⟨*effect reference*⟩ ::= '@' ⟨*effect name*⟩

⟨*feedback-type definition*⟩ ::= (⟨*feedback-type*⟩ | ⟨*feedback-type reference*⟩)

⟨*feedback-type reference*⟩ ::= '@' ⟨*name of type*⟩

Grammer 4.3: Assigning effects to feedback instances and types.

Our notation encourages the preprocessing-core-postprocessing partition of effects (Grammer 4.4). Any of these phases could be empty, marked by nothing (empty string) between their separators. Our notation also allows us to reuse already defined phases.

⟨*effect*⟩ ::= '(' ⟨*preprocess*⟩ '||' ⟨*core*⟩ '||' ⟨*postprocess*⟩ ')'

⟨*preprocess*⟩ ::= ⟨*phases*⟩

⟨*postprocess*⟩ ::= ⟨*phases*⟩

⟨*core*⟩ ::= ⟨*phases*⟩

⟨*phases*⟩ ::= (⟨*phase*⟩ | '@' ⟨*name of phase*⟩) [',' ⟨*phases*⟩]

⟨*phase*⟩ ::= (['@' ⟨*name of phase*⟩] ⟨*steps*⟩ | '')

⟨*steps*⟩ ::= ⟨*step*⟩ [',' ⟨*steps*⟩]

Grammer 4.4: Denoting effect's phase structure.

The only difference between the notation of feedback types' or instances' effects is that the core phases of types' effects could also contain instances' effects (Grammer 4.5).

⟨*step*⟩ ::= ⟨*action name*⟩ ['(' ⟨*label*⟩ ',' ⟨*opinion*⟩ ')']

⟨*step of types' core*⟩ ::= ⟨*action name*⟩ ['(' ⟨*label*⟩ ',' ⟨*opinion*⟩ ')'] | ⟨*effect reference*⟩

Grammer 4.5: Reusing effects of instances in type's effect.

If the arguments of an action are irrelevant, they could be omitted. In any other case, stakeholders should mark the arguments for actions but not for the instances' effects; hence they have no arguments. When an effect's inner workings depend on the feedback instance it is assigned to, it has to be defined as two distinct effects.

5 EXAMPLES

In this section, we elaborate on two examples of the usage of the proposed terminology. Both offer a solution for the same problem but (as our terminology emphasizes it) approach it from different views. Their common goal is to utilize the experience of the developers during the fault localization process. We use the names of the first authors of each paper that present the feedback systems: *Horváth* for (Horváth et al., 2020) and *Gong* for (Gong et al., 2012).

5.1 Selection of the Subjects

Horváth et al. (Horváth et al., 2020) inspected their proposed method with two granularity levels. Hence the experiments require the definition of two subject sets: one for the source code lines and one for the methods. On the other hand, Gong, et al. (Gong et al., 2012) executed their experiment solely on the source code lines. The formalization using our terminology is shown in Example 5.1.

$$S^{(\text{lines})}_{\text{Horváth}} = \{lines\} \qquad S^{(\text{methods})}_{\text{Horváth}} = \{methods\}$$

$$S_{\text{Horváth}} = S^{(\text{lines})}_{\text{Horváth}} \cup S^{(\text{methods})}_{\text{Horváth}}$$
$$S_{\text{Gong}} = \{lines\}$$

Example 5.1: The subject sets of the example feedback systems.

5.2 Grouping the Subjects

The first significant difference between the two approaches is the number of possible labels used to group their subject items. Horváth, et al. emphasize the contextual relationships between the code elements. They use a subject-dependent labeling $\phi^{r}_{\text{Horváth}}$, i.e. the labels will be according to the code element currently under inspection (r).

Gong, et al. allow the user to express their opinion on any subject items independently. Hence their feedback-system assigns a unique label to each code element, i.e. its labeling (ϕ_{Gong}) is the identical function and the set of subject items and their labels are equal. The formal definition of these are shown in Example 5.2

$$L_{\text{Horváth}} = \{\text{this}, \text{context}, \text{other}\}$$
$$r \in S_{\text{Horváth}}, \text{ the element under inspection}$$
$$\phi^{r}_{\text{Horváth}}(s) = \begin{cases} \text{this}, & \text{if } s = r \\ \text{context}, & \text{if } s \in S^{(\text{lines})}_{\text{Horváth}} \wedge \\ & \quad s \text{ and } r \text{ in the same method} \\ \text{context}, & \text{if } s \in S^{(\text{methods})}_{\text{Horváth}} \wedge \\ & \quad s \text{ and } r \text{ in the same class} \\ \text{other}, & \text{else} \end{cases}$$
$$L_{\text{Gong}} = S_{\text{Gong}}$$
$$\phi_{\text{Gong}}(s) = s$$

Example 5.2: The labeling of the example feedback systems.

5.3 Available Opinions

It can be stated that both systems use the same set of options: a subject is either faulty or clean, but the user may not provide any opinion about a subject (Example 5.3).

$$O_{\text{Horváth}} = O_{\text{Gong}} = \{\text{clean}, \text{faulty}, \text{nothing}\}$$

Example 5.3: The opinions of the example feedback systems.

5.4 Feedback Instances and Types

The number of accepted feedback instances differs greatly in the two systems. Gong, et al. allow the usage of every possible marking (with any opinion) of all labels (i.e. lines of code), see Example 5.5. On the other hand, Horváth, et al's system only accepts four different feedback instances (Example 5.4).

5.5 Actions and Effects

Both systems define a small set of actions. These are the application of R1 and R2 rules for Gong, et al. and "terminate"[1], "change the labeling", and "set to zero" for Horváth, et al. They both use the action which does nothing.

[1]The feedback gathering process could end in two ways: either the user interrupts the process or the system terminates it as an effect of some feedback.

@fault is found$\{\{$
  this $\rightarrow$ faulty,
  context $\rightarrow$ clean,
  other $\rightarrow$ clean$\}\}$
@element and context are not faulty$\{\{$
  this $\rightarrow$ clean,
  context $\rightarrow$ clean,
  other $\rightarrow$ faulty$\}\}$
@element is not but context is faulty$\{\{$
  this $\rightarrow$ clean,
  context $\rightarrow$ faulty,
  other $\rightarrow$ clean$\}\}$
@don't know$\{\{$
  this $\rightarrow$ nothing,
  context $\rightarrow$ nothing,
  other $\rightarrow$ nothing$\}\}$

Example 5.4: Feedback types of Horváth et al's system.

$$\begin{gathered} \dots \\ \{l_1 \rightarrow \text{clean}, l_2 \rightarrow \text{faulty}, \dots, l_{42} \rightarrow \text{nothing}, \dots\} \\ \dots \\ \{l_1 \rightarrow \text{clean}, l_2 \rightarrow \text{clean}, \dots, l_{16} \rightarrow \text{clean}, \dots\} \\ \dots \\ l_i \in L_{\text{Gong}}, \text{ labels (i.e., the source code lines)} \end{gathered}$$

Example 5.5: Feedback instances of Gong et al's system.

The first one of Horváth, et al's system is a process-related action, which will terminate the fault localization algorithm, i.e., the process of the feedback system itself. The second one will update the labeling of the subject items by changing the inspected to the next item; hence it is a meta action. The last one will set the SBFL score of some subject items. Gong, et al. only use subject-related actions. See section 3.3.2 for details about the kinds of actions and effects.

$$A_{\text{Horváth}} = \{\text{terminate}, \text{change } \phi^{r}_{\text{Horváth}}, \text{set to } 0, \text{NOP}\}$$
$$A_{\text{Gong}} = \{\text{apply rule R1}, \text{apply rule R2}, \text{NOP}\}$$

Example 5.6: Actions for the example systems.

The feedback-system of Horváth et al. use these actions to construct the four distinct effects assigned to the feedback-types (Example 5.7).

Gong et al. define a "unique" effect for each instance (Example 5.8), which will apply their R1 and R2 rules to subjects assigned clean and faulty, respectively (and "NOP" in all other cases).

```
@fault is found $\rightarrow$ (
  $\|$terminate, NOP(context, clean), NOP(other, clean)$\|$)
@element and context are not faulty $\rightarrow$ (
  $\|$set to 0(this, clean), set to 0(context, clean),
    NOP(other, faulty)$\|$
  change $\phi^r_{\text{Horváth}}$)
@element is not but context is faulty $\rightarrow$ (
  $\|$NOP(this, clean), set to 0(context, faulty),
    set to 0(other, clean)$\|$
  change $\phi^r_{\text{Horváth}}$)
@don't know $\rightarrow$ (
  $\|$NOP(this, nothing), NOP(context, nothing),
    NOP(other, nothing)$\|$
  change $\phi^r_{\text{Horváth}}$)
```

Example 5.7: Effects of Horváth et al's system.

```
...
$\{l_1 \rightarrow \text{clean}, l_2 \rightarrow \text{faulty}, \ldots, l_{42} \rightarrow \text{nothing}, \ldots\} \rightarrow ($
$\|\text{R1}(l_1, \text{clean}), \text{R2}(l_2, \text{faulty}), \ldots, \text{NOP}(l_{42}, \text{nothing}), \|)$
...
$l_i \in L_{\text{Gong}}$, labels (i.e., the source code lines)
```

Example 5.8: Effects of Gong et al's system.

6 CONCLUSION AND FUTURE WORKS

In this paper, we presented the first revision of a terminology for feedback systems with a notation that can describe the constituents of the systems. The capabilities of the terminology were illustrated with two detailed examples. Each of these sample systems incorporates the developers' knowledge into the fault localization processes.

The key benefit of our terminology that it will help to evaluate various feedback systems by aiding their comparison with each other or with their reference implementations. We could save a considerable amount of time during the comparison of the two systems if they were utilizing a common terminology. The effort to connect the various terms with their counterparts in the other system could be eliminated.

Our terminology revealed several other properties and connections, which we plan to enumerate and categorize in a following research. At first, we will collect more feedback systems that are already published to address SE-related issues. The identified properties of these systems will be used to improve the versatility of the terminology and the expressive power of the notation. Users' surveys will validate this enhanced terminology and notation to improve their usability further. Our longterm goal is to define a true "lingua franca" for the stakeholders to define and express the users' feedback they expect and the reactions they give.

ACKNOWLEDGMENTS

The research was supported by the Ministry of Innovation and Technology NRDI Office within the framework of the Artificial Intelligence National Laboratory Program (RRF-2.3.1-21-2022-00004).

Project no. TKP2021-NVA-09 has been implemented with the support provided by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund, financed under the TKP2021-NVA funding scheme.

REFERENCES

Gong, L., Lo, D., Jiang, L., and Zhang, H. (2012). Interactive fault localization leveraging simple user feedback. In *IEEE International Conference on Software Maintenance, ICSM*.

Hao, D., Zhang, L., Zhang, L., Sun, J., and Mei, H. (2009). Vida: Visual interactive debugging. In *2009 IEEE 31st International Conference on Software Engineering*, pages 583–586.

Hassenzahl, M. and Tractinsky, N. (2006). User experience - a research agenda. *Behaviour and Information Technology*, 25(2):91–97. cited By 1458.

Horváth, F., Beszédes, Á., Vancsics, B., Balogh, G., Vidács, L., and Gyimóthy, T. (2020). Experiments with interactive fault localization using simulated and real users. In *Proceedings of the 36th IEEE International Conference on Software Maintenance and Evolution (ICSME'20)*, pages 290–300.

ISO. Iso/iec 14977:1996 - information technology — syntactic metalanguage — extended bnf. https://www.iso.org/standard/26153.html. (Accessed on 11/01/2021).

Korel, B. and Laski, J. (1988). Stad - a system for testing and debugging: User perspective. pages 13–20. cited By 20.

Law, E.-C., Roto, V., Hassenzahl, M., Vermeeren, A., and Kort, J. (2009). Understanding, scoping and defining user experience: A survey approach. pages 719–728. cited By 599.

McIntosh, S., Kamei, Y., Adams, B., and Hassan, A. (2016). An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering*, 21(5):2146–2189. cited By 112.

Thongtanunam, P., Tantithamthavorn, C., Kula, R., Yoshida, N., Iida, H., and Matsumoto, K.-I. (2015). Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. pages 141–150. cited By 110.

Wooding, M. The syntax package. https://mirror.szerverem.hu/ctan/macros/latex/contrib/mdwtools/syntax.pdf. (Accessed on 11/01/2021).