An Abstract State Machine (ASM) Representation of Learning Process in FLOSS Communities

Patrick Mukala(✉), Antonio Cerone, and Franco Turini

Department of Computer Science, University of Pisa, Pisa, Italy
{patrick.mukala, cerone, turini}@di.unipi.it

Abstract. Free/Libre Open Source Software (FLOSS) communities as collaborative environments enable the occurrence of learning between participants in these groups. With the increasing interest research on understanding the mechanisms and processes through which learning occurs in FLOSS, there is an imperative to describe these processes. One successful way of doing this is through specification methods. In this paper, we describe the adoption of Abstract States Machines (ASMs) as a specification methodology for the description of learning processes in FLOSS. The goal of this endeavor is to represent the many possible steps and/or activities FLOSS participants go through during interactions that can be categorized as learning processes. Through ASMs, we express learning phases as states while activities that take place before moving from one state to another are expressed as transitions.

Keywords: Process modeling · Abstract State Machines (ASMs) · FLOSS communities · Learning processes

1 Introduction

The idea of process definition entails specifying the activities and flow of occurrences thereof between learning actors within the settings of Free/Libre Open Source Software (FLOSS) communities. The current literature is endowed with extensive exploration, critiques and development of specification languages in software engineering and modeling [1–7]. These languages and associated methods help in simulating and possibly verifying behaviors and functionalities of computer programs before they are developed. Specifically, some works [5, 8] draw attention to an important role of specification methods with regard to producing simulation models. These are models that depict a representation of some functionality as it is expected to occur in a specific domain. Furthermore, there has been an increasing interest in the area of specification languages for process modeling [9–13] and one of them, the Process Specification Language (PSL), provides a set of concepts and terms used for the description of process reengineering, process realization, process simulation etc. [9, 10]. Another specification language that can be used for both software engineering and process modeling is Abstract State machines (ASM) as suggested by Farahbod, Glässer and Vajihollahi [14]. In additional reports on this method [15, 16] Börger gives a detailed annotation of ASM biography since the inception of this area of research.

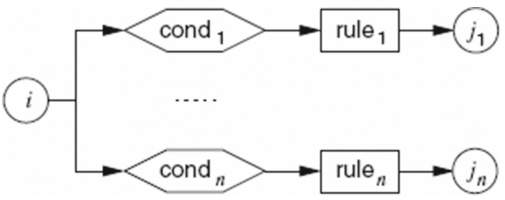
C. Canal and A. Idani (Eds.): SEFM 2014 Workshops, LNCS 8938, pp. 227–242, 2015.
DOI: 10.1007/978-3-319-15201-1_15

As part of our work, we chose the ASM approach as a way of specifying and defining learning processes because of its implementation success rate under industrial constraints for rigorous process modeling, software and hardware development, as emphasized by Börger, [17] and also because of its practicality and simplicity in documenting the steps from high-level abstraction of specifications to their decomposition until the ground models are produced.

Hence, the goal of this paper is to model learning processes from FLOSS repositories and clearly explain them through ASMs from the initial natural language in which they have been thus far expressed so as to enhance the understanding of learning behaviors and patterns through participatory activities in FLOSS communities. The paper is structured as follows. We briefly discuss Abstract States Machines and their relevance to our work in Sect. 2. Section 3 gives a general description of ASMs constructs and requirements as needed in this context. In Sect. 4 we present the developed ASMs Ground Models and specifications. In Sect. 5 we show how to validate the ASM specification of FLOSS learning processes by process mining FLOSS repositories. Finally, Sect. 6 concludes the paper.

2 Abstract State Machines (ASMs): Motivation

ASMs [14–18] can be understood as extensions of FSM (Finite State Machines) where any desired level of abstraction can be achieved by permitting possibly parameterized locations to hold values of arbitrary complexity, whether atomic or structured: objects, sets, lists, tables, trees, graphs, whatever comes natural at the considered level of abstraction [18]. Contrary to FSM, ASMs represent FSM instructions as control state rules as depicted in Fig. 1 below.



if *ctl state* = *i* **then**
  if $cond_1$ **then**
    $rule_1$
  ctl state := j_1
  • • •
  if $cond_n$ **then**
    $rule_n$
  ctl state := j_n

Fig. 1. Viewing FSM instructions as control state ASM rules

Figure 1 depicts an automaton where *i, j*1. , *j* are internal (control) states, $cond_v$, for $1 \leq v \leq n$, represent the input condition $i_n = a_v$ (reading input a_v) and $rule_v$, for $1 \leq v \leq n$, represent the output action *out* := b_v (yielding output b_v), which go together with the *ctl state* update to j_v. Simply put, these rules represent actions that need to be taken or activities carried out in the event of conditions (if any) being true before moving from one state to another.

The potential of using Abstract States Machines (ASMs) for modeling learning processes in volatile and dynamic FLOSS environments is twofold. First, the ASM

approach can help elaborate and express information we gather about activities in FLOSS communities and turn this information into necessary *ground models*. Ground Models are defined as functionally complete but abstract descriptions of sufficient but not more than necessary rigor, which can be read and understood as a representation of the problem to be solved and also contain only what the logic of the problem requires being modeled [17]. Finally, ASMs allow for the implementation and verification of the ground models reliably by refining them as needed step by step and through a hierarchy of intermediate sub-models which represent a major component of the problem [17]. We refer the reader to Börger's work [17] for detailed theory and semantic foundations of this approach.

2.1 Abstract State Machines (ASMs) for Learning Processes in FLOSS Communities

Modeling learning processes through ASMs grew out of a need to express the flow of occurrence for processes. A Ground model provides a simple representation of states, processes and transitions that we believe are fit to explain and give the readers a clear picture of learning processes in FLOSS communities. A number of studies [19–25] provide a lot of grounds for the identification of terms and concepts one can use to identify learning activities, participants and related classes in FLOSS communities.

In the context of our work, we identified two main learning processes in FLOSS communities (Undirected Learning and Directed Learning), with the second learning process (Directed Learning) unfolding from 2 perspectives in 4 different formats, thus totaling the number of processes to 9. These are:

- Undirected Learning: This process can also be referred to as Peer-2-Peer or Reflective Learning. This kind of learning is assumed to take place between any numbers of participants. In this process, any participant can be both a receiver (Novice) and a sender (Expert). At this level, the assumption is that learning occurs between mates with a diversified expertise background who learn from each other.
- Directed Learning: This process refers to involvement of more knowledgeable participants or expert members in helping less expert members to develop their skills with some level of guidance or supervision. The occurrence of the process is twofold:
  - Pulling: This is the process where a participant who is less expert on any topic would initiate a need to learn by reaching out to the more advanced participants that can culminate in a supervised or guided learning process. This can in turn occur according to the four formats as follow:
    - Modeling: In this process, the Expert's activities and actions are systematically monitored and observed by the Novice. This can happen as the receiver aims to emulate the sender given the latter's reputation on their FLOSS contribution. An example could be tracking the sender's commits in SVN, their comments on mailing lists etc.;
    - Coaching: As the term explains, this involves giving direct monitoring and guidance to the requester's and observing his/her performance;
    - Scaffolding: In this process, the sender analyses and determines the receiver's level of capacity and allows him/her the opportunities to acquire

knowledge accordingly. For example, supplying materials (tutorials etc.) on specific problems and a solution approach etc. based on the requester's background.
    - Fading: This process depicts involving a requester in practical execution of tasks for skills acquisition. However, as the requester's performance matures, the sender gradually gives them autonomy to apply their skills.
- Pushing: This is the type of directed learning that occurs when the sender takes the initiative to make available opportunities of knowledge acquisition for requesters. Just like the pulling, this process can also be understood in 4 formats: Modeling, Coaching, Scaffolding and Fading.

Therefore, given this classification, one can identify nine learning processes, namely Undirected/Reflective Learning, Directed-Pulling-Modeling, Directed-Pulling-Coaching, Directed-Pulling-Scaffolding, Directed-Pulling-Fading, Directed-Pushing-Modeling, Directed-Pushing-Coaching, Directed-Pushing-Scaffolding and Directed-Pushing-Fading.

3 ASM Requirements and Constructs

In a FLOSS community, the transfer of software engineering skills between participants occurs in an informal fashion but can be tracked through participants' activities. In order to accomplish this, some qualitative works on these activities have helped us identify and formulate the 9 learning processes introduced in Sect. 2.1 through which learning for this purpose occurs in FLOSS communities.

The purpose of building the ASMs is to develop models that can express the occurrence of these learning processes. It is crucial noting that learning in FLOSS for all processes that we have identified happens in three important phases that corresponds to three stages of software engineering skills development. Such phases are: initiation, progression and maturation. For each stage, a number of tasks/activities are carried out by both the Novice (knowledge requester) and the Expert (knowledge provider). Details of these activities in each phase or related thereof are graphically represented in the Ground Models in the next section. While these phases and states apply for all the identified learning processes, the demarcation thereof can be reflected through instances of activities for each learning process. This simply means that for all the identified instances of the learning process, there is a difference in the way the Novice and the Expert would interact. For instance, with regard to an activity such as *CommentOnCode*, an Expert in Reflective Learning process will provide an opinion on improving the commented code or any other observation with an understanding that the code's owner is a peer who can agree or disagree with the suggestions. However, when an explicit relationship has been established like in the context of the Directed-Pulling-Modeling learning process, an Expert reacts mainly to the Novice's comments with the intent to provide guidance with the same *CommentOnCode* activity. The difference in the steps undertaken to fulfill the same activity for these two learning processes lies in the levels of responsibility, role of the Expert and Novice as well as their mutual consideration for both learning processes. In spite of such differences, all the learning processes can be considered through three phases, namely initiation, progression and

maturation, which are expanded with related activities respectively in Figs. 2, 3 and 4. The initiation phase sanctions the start of a learning process as can be seen in Fig. 2 while the completion of a learning process can be demonstrated through the Novice ability to undertake activities of the maturation phase as depicted in Fig. 4.

Therefore, the ASM Ground Models depicted below are built based on a number of ASM constructs including states and transitions. Each Ground Model represents a learning phase, with a number of states and activities. These activities are ASM transitions and they determine moving from one state to another. In summary, we have Participants (Novice or Expert) that take part in a learning process (L_P) that occurs through three phases with corresponding states and transitions (activities).

These terms (phases, states and transitions) capture the main constructs that explain the different phases and activities a Novice goes through during a Learning Process. The Expert plays a critical role during these phases as a knowledge provider. The three ASM Ground Models as depicted in Figs. 2, 3 and 4 describe at some extent the interaction Novice-Expert during the Learning Process in terms of activities they perform in each respective phase. For clarity purposes, in order to illustrate the control flow as efficiently as possible, we consider only two participants (one Novice and one Expert) taking part in a learning process at a time.

We can thus express in ASM notation the basic specification that a phase in a learning process L_P can be any of list as enclosed in the brackets, namely initiation, progression and maturation. $\exists$ *Phase:* L_P —> *{Initiation, Progression, Maturation}*. We can further express that in the initiation phase, a state can take any of the enclosed values (*Observation* or *ContactEstablishment*): *State: Initiation —> {Observation, ContactEstablishment}*. The same applies for the remaining two phases expressed respectively: *State: Progression —> {Revert, Post, Apply}* and *State: Maturation —> {Analyze, Commit, Develop, Revert, Review}*. Finally, for consistency we can also express that two types of participants (Expert and Novice) take part in the learning process as $\exists$ *Participants: Participant* $\in L_P$ —> *{Expert, Novice}*.

4 ASMs Specifications and Ground Models

A Learning Process (L_P) is assumed to take place between any numbers of participants. In this process, any participant can be either a Novice or an Expert depending on the level of expertise and participant's profile. At this level, the assumption is that learning occurs between mates with a diversified expertise background who learn from each other. In Sects. 4.1–4.3 we describe the three phases (initiation, progression and maturation) and, for each phase, activities are identified and described phase-dependently.

4.1 Initiation Phase Ground Model

Two main states explain this phase: *Observation* and *ContactEstablishement*. This simply refers to the steps in which the Novice or Expert will attempt to establish some form of contact between them. As depicted in Fig. 2, in the first state (*Observation)* of the state machine, both the Novice and Expert undertake a number of activities. When a Novice seeks help, he/she can perform activities such as *FormulateQuestion,* and/or

IdentifyExpert and then *PostQuestions or CommentPost,* whereas the Expert can provide help after he/she performs either *ReadMessages* on the mailing lists or *ReadPost* from forums *or ReadSourceCode* as any participant commits code to the project, or *CommentPost*. After completing these activities, we move to the second state (*ContactEstablishement*).

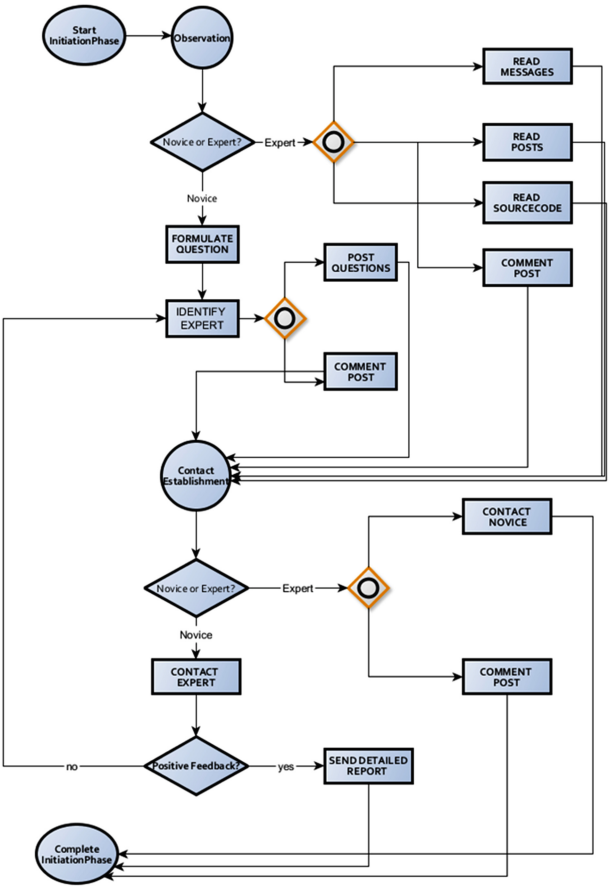


Fig. 2. Initiation ASM Ground Model

In the second state (*ContactEstablishement*) the Novice and/or Expert attempts to make contact in order to establish collaboration or to provide the required help. Hence, the *Novice* at this point, can simply perform *ContactExpert.* If the Expert responds positively, he/she can perform *SendDetailedRequest,* otherwise the cycle is restarted to identifying an expert. The Expert on the other side, just like the Novice seeking to be part of some form of knowledge channel, can perform *ContactNovice* and show interest in helping or simply perform *CommentPost* as shown in Fig. 2.

As said above, this phase of learning occurs for all instances of learning processes, the demarcation lies on two critical factors: the level of the Expert's involvement and

the content of messages and deliverables exchanged. In future, we hope to unmask and explain these differences as we empirically explore data from sample FLOSS projects. Nevertheless, at this point one can note that in the context of undirected learning, a Novice can be seen as a participant with considerable knowledge and skills because the exchange is assumed to take place between two colleagues. This could be the default perceptive knowledge exchange in such community environments where participants learn from each other. However, with the remaining of the learning processes, the emphasis is on the formal communication channel that exists between the concerned participants as well as the knowledge gap and disparity between them. This gap makes it possible for some level of mentorship as it allows for modeling, coaching, and fading and scaffolding as previously eluded. Hence, the semantics of activities such as *ContactExpert*, *ContactNovice*, *PostQuestions*, and *CommentPost* will vary accordingly although this specification is quite representative of the steps that actually take place in the process of observing occurring activities and possibly establishing these ties.

This description can also be summarized by the following ASM code providing the specification as graphically represented by the Ground Model in Fig. 2.

```
Let Lpact = activity occurring in LP phase;
Let P denote Participant in Floss Community in LP and PFloss denotes the total number of
participants;
Let Pe and Pn respectively denote sets of participants that are expert and novice.
The keyword Choose simply denotes a choice to be made from listed options.

If state = observation then
        If participant = n then
                FORMULATEQUESTION (n)
                IDENTIFYEXPERT (n)
                Choose Lpact in (POSTQUESTIONS (e),
                                COMMENTPOST (e)) do
                        Lpact
        If participant = n then
                Choose Lpact in (READMESSAGES (PFloss),
                                READPOST (PFloss),
                        READSOURCECODE (PFloss),
                            COMMENTPOST (PFloss)) do
                        Lpact
                                                        //Move to second state
If state = ContactEstablishment then
        If participant = n then
                CONTACTEXPERT (n)
                If SENDFEEDBACK (e) then
                        SENDDETAILEDREPORT (e)
                Else
    StartInitiationPhase
                        IDENTIFYEXPERT (n)
        If participant = Expert then
                Choose Lpact in (CONTACTNOVICE (n),
                                COMMENTPOST (n)) do
                        Lpact
```

4.2 Progression Specification and Ground Model

As with the first phase of the Lp, the enactment steps of these activities are quite similar with the only demarcation factors being the level of the Expert's involvement and the content of messages and deliverables exchanged. In this phase, while the Novice involved in the learning process starts gradually performing some apparent activities pertaining either to developing source code, commenting source code and actively engaging in the community discussions, the Expert's role shifts towards assessing and assisting the Novice where needed to ensure that the skills are effectively applied. Like in the next phase, this role is carried out in almost the same way while unleashing the Novice's full autonomous operation.

In this phase, the specifications above and Ground Model provided in Fig. 3 can be summarized as follows. Three main steps take place for both participants after establishing contact. These are *Revert, Post* and *Apply* and they denote important states within this phase.

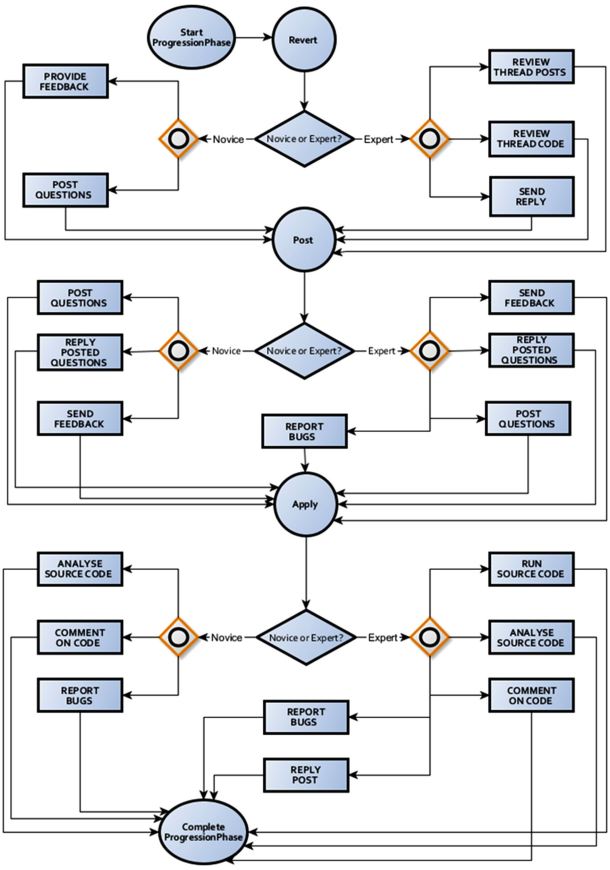


Fig. 3. Progression Phase Ground Model

These activities denote providing some level of feedback after being contacted, then providing the required help for Novice and implementing new knowledge through a set of new activities. In *Revert*, the Expert will then, if she/he accepted the request from the Novice, attempt to perform *ReviewThreadPosts,* where questions or need for clarification have risen, and *ReviewThreadCode,* for the purpose of critiquing and fixing as required if needs be. He/she can also perform *SendReply* in an attempt to answer any direct questions and help requests or just reacting to a discussion in a forum. While the Novice, at this state, can only react to the Expert's help or feedback and provide insights on the extent to which the Expert's input was helpful through *ProvideFeedback* or simply pose more questions through activity *PostQuestions.* In state *Post,* both the Novice and Expert illustrate their activities by performing a number of tasks that can all be grouped under this state. Some of these tasks include: *PostQuestions*, *ReplyPostedQuestions*, *ReportBugs* and *SendFeedback.* Hence, the flow of occurrence of these tasks happens as follows: the Novice will, during this state, perform a number of activities in the context of posting. These activities include *PostQuestions, ReplyPostedQuestions* and possibly *SendFeedback,* when needed, while the expert will directly or indirectly perform *SendFeedback, ReplyPostedQuestions (*possibly questions and requests from novice*), PostQuestions,* in order to enquire more if there is a need for clarity, as well as *ReportBugs,* as a response to Novice's need to understand why some piece of code cannot run properly, for example. In the final state, *Apply,* core development and practical activities are undertaken from both sides. The Novice can start exercising the new acquired skills through activities such as *AnalyseSourceCode,* when he/she looks at new commits, new pieces of code being posted by community members and hence, the novice will also be able to perform *RunSourceCode* on these pieces of code and comment on them through *CommentOnCode* and reporting bugs through *ReportBugs.*

The expert in turn can monitor the Novice through also a set of almost similar activities but for the purpose of evaluating the level of skill acquisition. These activities include *RunSourceCode* and *AnalyseSourceCode*, to identify flaws in the Novice's works, and, if necessary, *ReportBugs, CommentOnCode* and also *ReplyToPost.* Any more activities in this phase could trigger further states, but we set the limit of the scope at this point.

4.3 Maturation Specification and Ground Model

To conclude with the last part of the specifications, as with the two previous phases, most activities here in this phase have gained a certain level of maturation. It means in this phase, the role of the Expert becomes more or less a sporadic assessor that progressively considers the Novice as a colleague and member of the same community. The Novice, on the other hand, can possibly start at some extent new knowledge exchange channels as an Expert, to transfer the newly acquired skills during participation in FLOSS environments. The following specification can be easily understood by looking at the graphical representation in Fig. 4.

In this last phase of the learning process, the activities are presented to assert how the novice has mastered the skills learnt during the learning process. Five main groups of activities make up this phase of the process as referred to as states. These include *Analyze, Commit, Develop, Review* and *Revert*. In the *Analyze* state, the Novice or Expert engage in a set of activities to examine the maturity of the learning process. The Novice is assumed to have acquired enough skills to be able to undertake a set of activities, such as *AnalyzeDiscussions,* in order to actively engage and contribute to comments and posts in the team about topics in the sphere of the skills acquired and possibly becoming an Expert to a new Novice. Activity *AnalyzeSourceCode* consists in analyzing the code (when applied) in order to understand and critique that piece of software and, finally, activity *AnalyzeThreadProgression* is performed in order to be part of a discussion and exchange a channel that engages on a topic related to a new skill learnt. The Expert will perform the exact same activities but tracking the Novice's progress. Thus, these activities include *AnalyzeThreadProgression, AnalyzeSourceCode* and *AnalyzeDiscussions.*

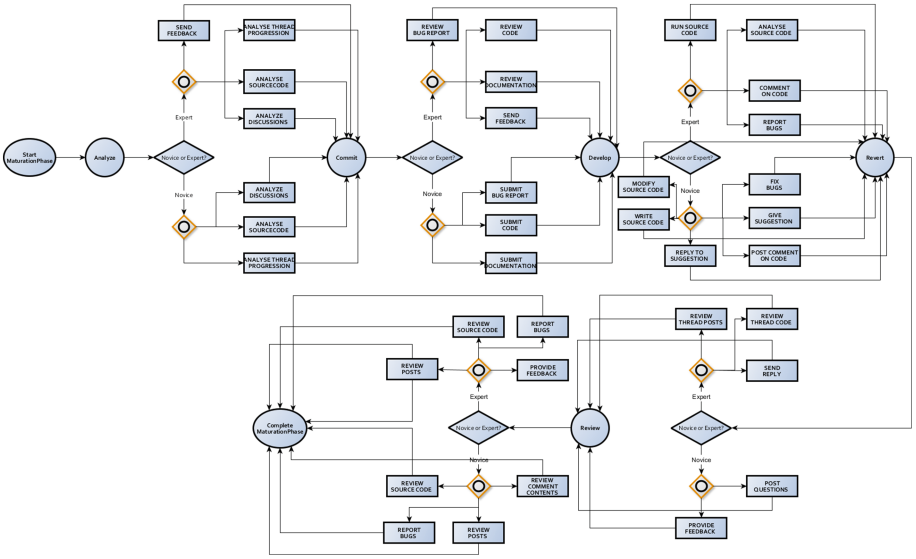


Fig. 4. Maturation ASM Ground Model

In the second state (*Commit*), the assumption is that as the Novice fosters his/her skills on a specific area, he/she can now commit some deliverables at the repositories that can be evaluated and criticized by the community. These activities include *SubmitBugReport,* where the Novice will commit any fix or bug report for the interest of the entire community, *SubmitCode,* where the Novice will commit some code for any piece of software and participate to the project and build reputation for a possible role transition, and also *SubmitDocumentation,* where the Novice is able to submit

documentation in terms of requirements elicitation documents, help document, user manuals, tutorials etc. The Expert on the other hand, is assumed to conduct the same activities for monitoring purposes and give feedback as needed.

During the next state (*Develop*), the Novice carries out a number of activities that demonstrate his/her ability to develop. These activities include:

- *FixBugs,* where he/she attempts to fix any reported bugs in the project;
- *GiveSuggestion,* as part of reviewing peers' works, which provide alternatives when needed (for example what the appropriate function might be to perform a particular task etc.);
- *PostCommentOnCode,* in order to make sure that appropriate indicative comments in the source code are posted for enlightenment;
- *ReplyToSuggestion,* to reply and critique suggestion from other experts or novices in an active fashion,
- *WriteSourceCode,* in order to commit pieces of software;
- *ModifySourceCode,* to modify any code and implement suggestion as requested.

In turn, the Expert carries out a number of activities as well during the learning process and can perform *RunSourceCode,* in order to be able to perform *Analyze-SourceCode,* and, if possible, as needed, perform *CommentOnCode* and *ReportBugs* to the benefit of the Novice.

The *Revert* state in this phase is essentially the same state as in the progress phase. This contains all feedback activities between the Novice and the Expert. Three classes or activities occur under this state: *ReviewThreadPosts, ReviewThreadCode* and *SendReply.* The Expert will then, if she/he accepted the request from the Novice, attempt to perform *ReviewThreadPosts,* where questions or need for clarification have risen, *ReviewThreadCode,* for the purpose of critiquing and fixing as required if needs be. The Expert could also perform *SendReply,* in an attempt to answer any direct questions and help requests or just reacting to a discussion in a forum. The Novice, at this state, can only react to the Expert's help or feedback and provide insights on the extent to which the Expert's input was helpful through *ProvideFeedback* or simply pose more questions through *PostQuestions.*

In the last state, called *Review*, the Novice or Expert engage in a set of activities to examine the maturity of the learning process in reviewing a number of posts and artifacts according to the level of competency. The Novice can undertake as part of his/her ability to Review a number of activities that can be assimilated to three main review activities such as *ReviewCommentContents,* in order to contribute to comments and posts in the team about topics in the sphere of the skills acquired and possibly becoming an expert to a new novice, *ReviewPosts,* on mailing lists and forums so as to react as needed to comments and posts related to a particular content that is the subject of learning, *ReviewSourceCode,* which explains the ability to analyze the code (when applied) and identify flaws that can be reported or fixed. The Expert, in this last phase of the learning process, will be performing the same activities as the Novice, but on the Novice's progress work. Hence, he/she will perform *ReviewPosts* on posts from the Novice and react as needed to comments and posts related to a particular content that is

the subject of learning. The Expert can also perform *ReviewSourceCode* in order to be able to perform *ReportBugs* and also perform *ProvideFeedback* when necessary.

5 Using Process Mining to Validate the ASM Model

The ASM models as built in this paper form part of the undertaking to identify traces of learning processes from FLOSS repositories. This evidence-based undertaking can be accomplished through process mining. Process mining is a method of reconstructing processes as executed from the event logs [29]. These logs are generated from process-aware information systems such as Enterprise Resource Planning (ERP), Workflow Management (WFM), Customer Relationship Management (CRM), Supply Chain Management (SCM), and Product Data Management (PDM) [28]. The logs contain records of events such as activities being executed or messages being exchanged on which process mining techniques can be applied in order to discover, analyze, diagnose and improve processes, organizational, social and data structures [29]. This can also be understood as the automated discovery of processes from event logs resulting in a generation of a process model (e.g., a Petri net or a workflow net) that describes the causal dependencies between activities [28].

More specifically, the goal of process mining is the extraction of information on the process from event logs using a family of a posteriori analysis techniques. These techniques enable the identification of sequentially recorded events where each event refers to an activity and is related to a particular case (i.e., a process instance) [28]. They also can help identify the performer or originator of the event (i.e., the person/resource executing or initiating the activity), the timestamp of the event, or data elements recorded with the event. Being able to retrieve such information is critical in our endeavor as we attempt to study the generation and originators of learning patterns from data recorded in FLOSS repositories. However, in these repositories, the structure of data files does not correspond to the required format of a log required for process mining. This can be illustrated by the Process Mining Meta Data Model in Fig. 5 below.

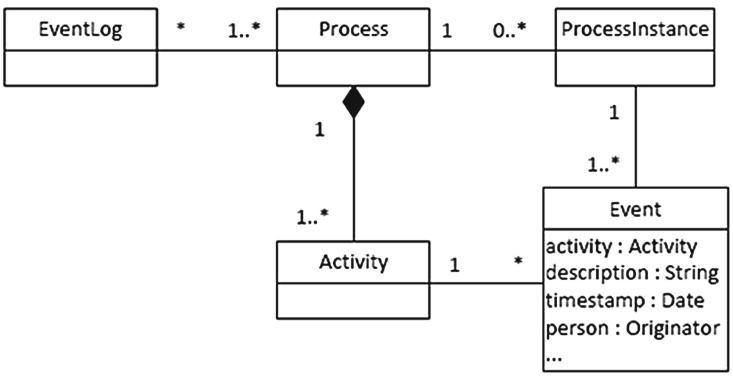


Fig. 5. Process Mining Meta Data Model

The idea as expressed in the model is that a log, an event log that is ready for process mining should abide by a number of structural properties to facilitate its processing and analysis. It should contain data organized and clustered in processes; each of these processes has instances uniquely identifiable with a set of activities. A process instance can also be referred to as a case instance includes a number of events that consist of activities being executed at a given point in time. An example could be a log of an insurance company might contain information about a billing and refund process. A refund process has a number of process instances uniquely identified by the claim number. Activities that should be executed in the refund process may include registering the claim, and checking the insurance policy. An example of an event is "On Thursday September 23, 2010 Alice checks the insurance policy of the persons involved in claim 478-12" [26].

However, in Floss repositories, data can be often found in form of statistical details or email messages exchanged in forums. Therefore, these ASM models will be used to help construct the logs containing all the information we need to represent the learning processes. These models can guide in identifying activities in these data as specified in the model and build an event log on this basis.

Furthermore, the output of process mining plays three important roles. These include discovery, conformance and extension [28]. In discovery, the idea is that a new model is discovered from the event log and it provides insights on processes in the systems. With conformance, there is an a priori model which is used to verify if the events recorded in the log conform to such model; this is used to detect deviations, locate and explain them in order to take appropriate actions. The last role of process mining is extension, where the a priori model is extended or enriched with new aspects, for example the extension of a process model with performance data.

In our context, we intend to use process mining for the second role, which can be referred to as conformance. Rozinat and van der Aalst [26] highlight that the question of conformance arises when there is a need to check for conformity. Given the existence of predefined models that specify how the processes should (or are expected to) be executed as our ASM models, conformance helps determine at what extent these models relate to the actual process models generated as a result of recorded data. For conformance, two techniques are mainly considered to this end, namely Delta Analysis and Conformance Testing.

Delta analysis is defined as a way of comparing the discovered model (process model) with some predefined model while conformance testing attempts to determine the "fit" between these two models [27]. In our case, delta analysis implies comparing the obtained process models as a result of process mining FLOSS repositories with our ASM models. This analysis can help validate the ASM models that describe how learning occurs or provide new insights that can help enrich this area of research and probably lead to their realignment if any discrepancies are detected in order to improve the process as seen in Fig. 6 below.

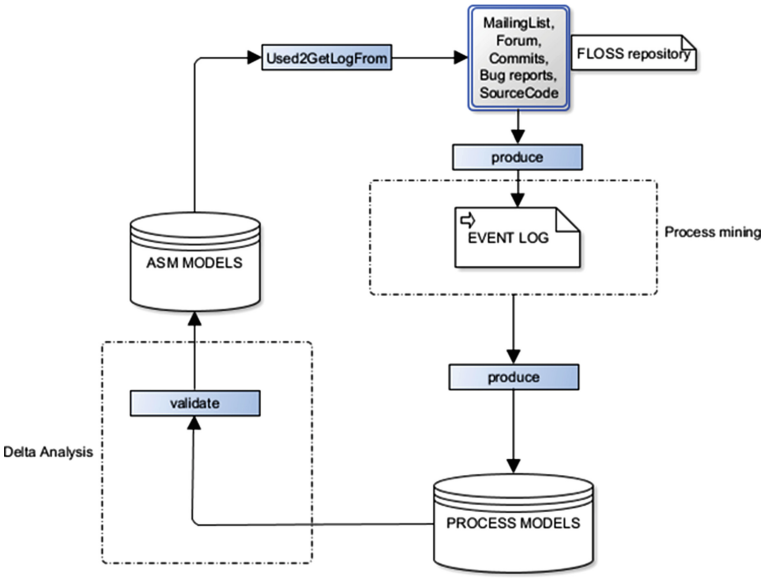


Fig. 6. ASM Ground Model used for Process Mining

6 Conclusion and Future Work

Mining Software Repositories for any purpose has been and still is the heart of numerous endeavors aiming at understanding open source software development. As we set to investigate and understand learning processes in FLOSS communities using process mining, ASM specifications can provide a detailed guideline in acquiring and retrieving relevant data. With the specifications at hands, in order to track learning processes in these environments, one can explore as much as possible data from repositories such as MailingList archives, Bug reports and CVS in order to understand patterns and related learning activities.

Our requirements as specified can be traced on these sets of data at an acceptable level of granularity.

We hope that these ASM specification and Ground Models are enough, at this point at least, to draw a representational idea as to the degree to which our identified learning processes occur in FLOSS communities.

As these activities in different phases unfold and are completed, it is critical to note that the representation of any learning process is enriched with an analysis and description of the impact of newly acquired skills on the Novice's contributions pre and post learning on a given project. In the future, we plan to conduct some empirical experiments based on these specifications in order to explore and analyze these learning processes from FLOSS repositories using Process mining techniques.

References

1. Fensel, D.: Formal specification languages in knowledge and software engineering. Knowl. Eng. Rev. **10**(4), 361–404 (1995)
2. Tse, T.H., Pong, L.: An examination of requirements specification languages. Comput. J. **34** (2), 143–152 (1991)
3. Shaw, A.C.: Software specification languages based on regular expressions. In: Riddle, W.E., Fairley, R.E. (eds.) Software Development Tools, pp. 148–175. Springer, Heidelberg (1980)
4. Harmelen, F.V., Aben, M.: Structure-preserving specification languages for knowledge-based systems. Int. J. Hum Comput Stud. **44**(2), 187–212 (1996)
5. Bjørner, D., Henson, M.C.: Logics of Specification Languages, vol. 18. Springer, Berlin (2008)
6. Cooke, D., Gates, A., Demirörs, E., Demirörs, O., Tanik, M.M., Krämer, B.: Languages for the specification of software. J. Syst. Softw. **32**(3), 269–308 (1996)
7. Tan, Y.M.: Introduction. In: Tan, Y.M. (ed.) Formal Specification Techniques for Engineering Modular C Programs, pp. 1–15. Springer, New York (1996)
8. Overstreet, C.M., Nance, R.E., Balci, O., Barger, L.F.: Specification languages: understanding their role in simulation model development (1987)
9. Gruninger, M., Tissot, F., Valois, J., Lubell, J., Lee, J.: The process specification language (PSL) overview and version 1.0 specification. US Department of Commerce, Technology Administration, National Institute of Standards and Technology (2000)
10. Gruninger, M., Menzel, C.: The process specification language (PSL) theory and applications. AI Mag. **24**(3), 63 (2003)
11. Catron, B.A., Ray, S.R.: ALPS: a language for process specification. Int. J. Comput. Integr. Manuf. **4**(2), 105–113 (1991)
12. Schlenoff, C., Knutilla, A., Ray, S.: Unified process specification language: requirements for modeling process. Interagency Report, 5910 (1996)
13. Schlenoff, C., Ray, S., Polyak, S.T., Tate, A., Cheah, S.C., Anderson, R.C.: Process specification language: an analysis of existing representations. US Department of Commerce, Technology Administration, National Institute of Standards and Technology (1998)
14. Farahbod, R., Glässer, U., Vajihollahi, M.: Specification and validation of the business process execution language for web services. In: Zimmermann, W., Thalheim, B. (eds.) ASM 2004. LNCS, vol. 3052, pp. 78–94. Springer, Heidelberg (2004)
15. Börger, E., Stärk, R.F.: Abstract State Machines: A Method for High-level System Design and Analysis (with 19 Tables). Springer, Heidelberg (2003)
16. Börger, E.: The origins and the development of the ASM method for high level system design and analysis. J. Univ. Comput. Sci. **8**(1), 2–74 (2002)
17. Börger, E.: High level system design and analysis using abstract state machines. In: Hutter, D., Stephan, W., Traverso, P., Ullmann, M. (eds.) FM-Trends 1998. LNCS, vol. 1641, pp. 1–43. Springer, Heidelberg (1999)
18. Börger, E.: The ASM Method for system design and analysis. a tutorial introduction. In: Gramlich, B. (ed.) FroCos 2005. LNCS (LNAI), vol. 3717, pp. 264–283. Springer, Heidelberg (2005)
19. Glott, R., SPI, A.M., Sowe, S.K., Conolly, T., Healy, A., Ghosh, R., West, D.: FLOSSCom - Using the Principles of Informal Learning Environments of FLOSS Communities to Improve ICT Supported Formal Education (2011)

20. Glott, R., Meiszner, A., Sowe, S.K.: FLOSSCom Phase 1 Report: Analysis of the Informal Learning Environment of FLOSS Communities. FLOSSCom Project (2007)
21. Cerone, A.K., Sowe, S.K.: Using Free/Libre Open Source Software Projects as E-learning Tools. Electronic Communications of the EASST, 33 (2010)
22. Fernandes, S., Cerone, A., Barbosa, L.S.: Analysis of FLOSS communities as learning contexts. In: Counsell, S., Núñez, M. (eds.) SEFM 2013. LNCS, vol. 8368, pp. 405–416. Springer, Heidelberg (2014)
23. Rubin, V., Günther, C.W., van der Aalst, W.M., Kindler, E., van Dongen, B.F., Schäfer, W.: Process mining framework for software processes. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2007. LNCS, vol. 4470, pp. 169–181. Springer, Heidelberg (2007)
24. Cerone, A.: Learning and Activity Patterns in OSS Communities and their Impact on Software Quality. ECEASST, 48 (2011)
25. Sowe, S.K., Stamelos, I.: Reflection on Knowledge Sharing in F/OSS Projects. In: Russo, B., Damiani, E., Hissam, S., Lundell, B., Succi, G. (eds.) Open Source Development, Communities and Quality. LNCS, vol. 275, pp. 351–358. Springer, New York (2008)
26. Rozinat, A., van der Aalst, W.M.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**(1), 64–95 (2008)
27. van der Aalst, W.M.: Business alignment: using process mining as a tool for Delta analysis and conformance testing. Requir. Eng. **10**(3), 198–211 (2005)
28. van der Aalst, W.M., Rubin, V., Verbeek, H.M.W., van Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting. Softw. Syst. Model. **9**(1), 87–111 (2010)
29. De Weerdt, J., Schupp, A., Vanderloock, A., Baesens, B.: Process mining for the multi-faceted analysis of business processes—a case study in a financial services organization. Comput. Ind. **64**, 57–67 (2012)