Organization and Verification of Software Production Processes

Mikhail Grigorev a, Alexander Ivashko b, Inna Grigoreva c, Marina Vorobeva d, Artem Vorobeve e

University of Tyumen, 6 Volodarskogo St., 625003 Tyumen, Russia

E-mail: a m.v.grigorev@utmn.ru, b a.g.ivashko@utmn.ru, c i.i.grigoreva@utmn.ru, d m.s.vorobeva@utmn.ru, e a.m.vorobeve@utmn.ru

Abstract: The article proposes an approach to verification of the software production process. The essence of the approach is to present the development process as a sequence of artifacts distributed in time and interconnected by certain relationships. A finite state machine using the model checking approach checks this distributed sequence. The results of this method allow to reduce the number of errors that occur in the production process and to localize the place of their occurrence.

Introduction
In the modern software manufacturing industry much attention is paid to the development process. In the works of B. Boehm, R. Futrell, E. Braude, L. Lockwood, D. Shafer, it is convincingly shown that the successful development of software depends not only on obtaining a successful software product, but also on obtaining successful processes of developing software systems. In the absence of feedback and improvements, even the most practical project is bound to fail. There are methods that indirectly provide an opportunity to assess the quality of the organization of processes: attestation, audit, validation, verification, reviewing, testing, examination. As a rule, these methods are connected only with some disciplines of software engineering and are declarative in nature.

The modern level of formalization of methods of software engineering predetermines the development of formal methods for inspecting the quality of the organization of the software development process.

The aim of the work is to improve the quality of software engineering by formalizing the verification of the process of team project development.

Verification of software production processes
In the absence of feedback and improvements even the most practical process is bound to fail. You need to localize the problem location to continue working. There are many methods to identify problems and shortcomings. Inspections provide quick, effective and easily applied methods for identifying problems and finding their potential solutions [1]. However, it should be noted that inspection is not capable of replacing poor-quality design. The methods of inspection include expert evaluation, reviewing, testing, auditing, validation, verification and attestation.

In order to successfully carry out project appraisal, inspection methods should not only affect the project, but also the development process, which also ensures successful software engineering [2]. Next, let us consider the validation and verification methods that are most applicable to the inspection of software production processes.
Validation and Verification (V&V) of software is an orderly approach in the evaluation of software products, applied throughout the life cycle [2, 3, 4]. Efforts within the framework of verification and validation work are aimed at ensuring quality as an integral characteristic of the software and meeting user requirements. V&V directly addresses software quality issues and uses appropriate testing techniques to detect any defects. V&V can be used for intermediate products, however, to the extent that corresponds to the intermediate "steps" of the relevant life-cycle processes.

Description of the semantics of the development process

When applying the object-oriented approach to modeling, a four-level architecture defined by the Object Management Group (OMG) should be observed [5].

According to the Object Management Group (OMG) specifications, the M3 level represents the Meta Object Facility (MOF) modeling technology [5]. The M2 level represents the implementation of the Unified Modeling Language (UML) [6] metamodels and the Software Process Engineering Model (SPEM) [7]. Based on the UML metamodel, the SPEM profile is defined, which allows to formally describe the structure of the development process. At the next level (M1), the process modeling level, a method repository is modeled based on the SPEM profile and the UML metamodel. Based on the SPEM metamodel, a repository of methods is implemented.

According to the SPEM specification, any team process must determine the roles, activities (tasks) for each role, the set of artifacts used in a particular activity, and the set of artifacts produced as a result of this activity. In this case, the role carrying out the activity, will be responsible for each generated artifact.

Dynamic model of the software development process

To perform verification, it is necessary to set an estimate of each state of the system. A sequence of positive assessments of different states will indicate a correct (permissible) process execution. Since the state of the process is characterized by a set of artifacts, to assess the state, each artifact must be evaluated separately, and, based on the estimates presented, given an aggregate estimate characterizing the state.

Assertion 1. Evaluation of the correctness of the working product must include the result of checking the correctness of its development, which requires the formation of a system of formal rules for evaluating the development process.

However, it is not enough just to point out the incorrect execution of the process, it is necessary to determine the place of the error (errors) that led to the incorrect implementation of the process. To assess the state of the development process, you need to evaluate the way to achieve this state. This path may contain changing several artifacts. The verification procedure consists in verifying the correctness of the development of artifacts, thus assessing the correctness of the path to the implementation of the project activity. The result of the verification will be a sequence of changes in the estimates of the artifacts.

Assertion 2. A dynamic model of the software product development process, including an assessment of the correctness of the artifact, can be presented in the form of the Kripke model.

For a set of estimates of artifacts, the totality of which represents the state of the process, the Kripke model will have the following form:

\[
V = (S, S_0, R, T),
\]

where:
- \( S \) – the set of states in the development process; \( S_0 \subseteq S \) – the initial state;
- \( R \subseteq S \times S \) – the ratio of transitions for all possible states, i.e. for each state \( s \in S \) there must exist a state \( s' \in S \), such that \( R(s, s') \);
- \( T: S \rightarrow 2^{VR} \) – function of interpreting the state of the system as a set of estimates of artifacts;
- \( VR \) – a set of artifact estimates, corresponding to the atomic utterances of the Kripke model.

The state of the process can be described using the estimates: \( s: WP \rightarrow B \). From a given estimate, we can write down the first-order formula, which is true for this estimate.
It is necessary to determine the transitions between the states of the software development process. For a set of \( WP \) artifacts, we'll create a copy of \( WP' \) to treat \( WP \) as the current set of artifacts, and \( WP' \) is the set of artifacts for the next state of the development process. If \( R \) is the ratio of transitions, the record \( R(WP, WP') \) will mean the formula that represents it.

The set of verification results for the VR verification rule consists of the sets \( = b \), where \( wp \in WP \) and \( b \in B \).

The verification result will be determined when the following formula is satisfied:

\[
R(WP, WP') \equiv wp_1' \land ... \land wp_n'.
\]  

However, the two-valued logic is not expressive enough to verify the software processes, as the artifact should not be absolutely correct for the project to be feasible. Thus, it is necessary to define two groups of verification rules: critical ones, non-fulfillment of which leads to gross errors impeding project implementation and recommendatory ones, representing the best methods of software engineering.

The domain of labels \( B = \{-1, 0, 1\} \), where -1 means non-compliance with the critical rule, 0 - recommendation, and 1 - successful execution of any rule.

It is necessary to redefine the formula \( R \) to evaluate the verification result using the ternary logic. Thus, for labels 1 and -1, it is necessary to use the conjunction operator, and for label 0 - the disjunction operator to obtain the required final result.

\[
R(WP, WP') \equiv (wp_i ... wp_j = 1) \land (wp_k ... wp_l = -1) \lor (wp_m ... wp_n = 0).
\]  

The verification procedure consists in determining a subset of states \( a \in A \) in which the formula \( R \) is realized.

It should be noted that the result of the transition formula will be the value from the domain \( \{0,1\} \). The assertion is proved.

A comprehensive description of the order of events in time without the involvement of time in an explicit form allows to specify temporal logics. Temporal logics are usually classified according to whether the time structure is linear or branching. Since Kripke's model defines all possible states, the branching time temporal logic (CTL) must be applied to evaluate the correctness of the path.

Evaluation of the temporal formula on the Kripke model realizes the verification of the correctness of the development process. To perform the check, it is necessary to present the Kripke model as an infinite tree unfolded from the initial vertex of the transition graph.

Assertion 3. The result of the verification of the development process is the evaluation of the temporal relationship on the Kripke model.

The semantics of the temporal formula on the model (2) correspond to the following relation:

\[
\models: (V \times S \times Formula) \rightarrow \{True, False\},
\]  

where \( Formula \) determines the correctness of the path by estimating the path with a temporal formula of the \( EG \) form.

Thus, the result of the verification of the formula is a unique identification of the admissibility of the path. The assertion is proved.

The formal context-free grammar of the temporal formula \( \phi \):

\[
\phi ::= \bot \mid T \mid p \mid (\neg \phi) \mid (\phi \land \phi) \mid (\phi \lor \phi) \mid (\phi \Rightarrow \phi) \mid \phi \land \phi \mid AX \phi \mid EX \phi \mid AF \phi \mid EF \phi \mid AG \phi \mid EG \phi \mid A[\phi U \phi] \mid E[\phi U \phi].
\]
This grammar defines atomic formulas of the first order $\bot, \top, p$, logical operators $\neg, \land, \lor, \Rightarrow, \Leftrightarrow$.
The temporal operators of the branching time logic are made up by path quantifiers:

- $A \phi$ - for all calculation paths,
- $E \phi$ is for some path,
- as well as path operators:
  - $X \phi$ - for the next state,
  - $F \phi$ - for any state,
  - $G \phi$ - for all states,
  - $\phi U \psi$ - for all states in which $\phi$ is met, until $\psi$ is executed.

**Assertion 4.** Verification is feasible for a finite number of iterations.

Since the development process is represented by a single path in the calculation tree, the temporal formula should be determined by the type of $EG \phi$. Let $\mathcal{M}$ be a model of temporal logic of branching time. The semantic ratio for the above formula will be determined as follows:

$$\exists (s_1 \rightarrow s_2 \rightarrow \cdots) (s = s_i) \forall i ((\mathcal{M}, s_i) \models \phi).$$  \hspace{1cm} (6)

Checking one rule for the correctness of the artifact is determined at a separate point in time, the interleaving semantics are used for this. The estimation of each artifact of a $wp'_i$ state is defined by the formula $wp'_i = f_i(WP)$. The interleaving semantics are reflected in the disjunction of the form:

$$\mathcal{R}(WP, WP') \equiv \mathcal{R}_0(WP, WP') \lor \cdots \lor \mathcal{R}_{n-1}(WP, WP').$$  \hspace{1cm} (7)

where:

$$\mathcal{R}_i(WP, WP') \equiv (wp'_i \Leftrightarrow f_i(wp)) \land \bigwedge_{j \neq i} (wp'_j \Leftrightarrow wp_j).$$  \hspace{1cm} (8)

The verification procedure represents the verification of the temporal formula by sequential verification of the correctness of the performance of each artifact. The implementation of the procedure requires the formation of the Kripke model for the development process, its further transformation into the Moore finite automaton, which allows to generate a report of verification results, as well as a finite state machine that realizes the temporal formula, i.e. the following condition is fulfilled:

$$V, s_0 \models \phi, \text{ where } s_0 \in S.$$  \hspace{1cm} (9)

Forming possible ways to implement the software development process from the current state allows us to construct the Kripke model, which specifies all sequences of changes in artifact estimates leading to the next state.

**Creation of the Kripke model for the development process:**

1) a list of verifiable artifacts is determined (atomic utterances characterizing states);
2) all possible states of the development process are generated as a result of a description of all combinations of artifact evaluations as a result of checking the rules;
3) transitions are established between each pair of states, and also arcs for each state, indicating this state.

The next step in the verification method is the conversion of the Kripke model to a finite Moore automaton, registering each change in the state of the automaton in the output sequence. The transformation algorithm consists of the following steps:

1) each state of the Kripke model is uniquely transformed into a state of the automaton, each state of the automaton being marked as permitting;
2) each arc $(s, s') \in R$ is transformed into an automaton transition $(s, \alpha, s')$, where $\alpha$ is the atomic change in the set of state artifact estimates;
3) the output alphabet of a finite automaton is formulated, each symbol of which corresponds to a set of estimates of the artifacts of the corresponding state;
4) the output function is defined, which maps each state of the automaton to the symbol of the output alphabet.

The finite automaton is a mathematical model of a device the memory capacity of which is determined by a constant value regardless of the size of the input data [9, 10, 11, 12].

The Kripke model \((A, A_0, R, T)\), where \(T: A \rightarrow 2^{|R|}\), can be transformed into an automaton:

\[
B = (\Sigma, A \cup \{i\}, \Delta, \{i\}, A \cup \{i\}),
\]

where:

\(\Sigma = 2^{|R|}\) - a set of verification results verification results,
\(A\) - the set of states of the development process,
\(\Delta\) - the ratio of transitions,
\(i\) - the initial state.

For each pair of states \(a, a'\) in \(A\), the transition ratio \(\Delta\) contains the triple \((a, a', a')\) if and only if \((a, a') \in R\) and \(a = T(a')\). In addition, \((i, a, a) \in \Delta\) if and only if \(a \in A_0\) and \(a = T(a)\).

There are finite automata that allow you to generate output sequences.

We transform the automaton \(B\) into the corresponding Moore automaton. We obtain the following automaton:

\[
M = (\Sigma, A \cup \{i\}, \Delta, \{i\}, A \cup \{i\}, G),
\]

The result of the operation of \(M\) will be the sequence of all states of the verification process, as well as a sequence of estimates of the developed artifacts.

The Moore state machine obtained by transforming the Kripke model of the software development process has the form:

\[
M = (\Sigma, S, \Delta, s_0, S, G),
\]

where:

\(S\) - a finite set of states,
\(s_0 \in S\) - the initial state,
\(\Sigma\) - a finite input alphabet,
\(\Delta \subseteq S \times \Sigma \times S\) - the ratio of transitions,
\(G: S \rightarrow S\) - the output function.

The final step in the verification procedure is to check the implementation of the temporal formula that specifies the conditions for the permissible execution of the development process. The implementation of this stage is carried out by a finite automaton, which includes all the previous stages, thus forming a mathematical model of verification.

**Approbation of the results of work**

To carry out the experiment, a modification of the Rational Unified Process (RUP) was developed, including changes targeted at a small development team. The possibility of using this modification in educational projects is taken into account. In the course of solving the task of developing an information system project, a group of 3 people must perform work in two phases of the RUP: "Beginning", "Research". Each participant carried out various role tasks during the implementation of the project.

To solve the experimental problem, a minimal set of artifacts is identified that are significant results in the context of training projects. This set is represented by artifacts: the business use case model, the business analysis model, the “Vision” document, the use case model, additional specifications, activity
dia gram, design model, sequence diagram. For this set of artifacts, rules that define their semantic relationships are defined, which allow us to identify not only errors in the developed work products, but also make it possible to control the correctness of the process.

During the experiment, 16 projects were implemented. The implementation of each project was divided into 8 stages. Each odd stage consisted in the development of a specific set of artifacts. After each stage, a search was made and errors were taken into account, which could be admitted in the developed work products. Correction of errors was carried out at even stages. The implementation of the projects was organized in two independent series. In the first series, the project participants themselves carried out the search for errors. In the second - a system that provided recommendations for their correction.

Fig. 1 shows the dependence of the average number of errors found at various stages of the project for two series of experiments. The results of the experiment show that the application of the developed verification method has made it possible to reduce the error growth rate as the project is being implemented.

Figure. 1. The average number of project errors detected at different stages of the experiment
1 - a series in which errors were identified by project participants (2 - trend line),
3 - series, in which errors were detected by the system (4 - trend line).

In addition, the analysis of errors made by developers during the implementation of the project confirmed that successful development depends not only on obtaining a correct final result, but also on obtaining successful development processes. During the experiment it was shown that errors committed in artifacts may not correlate with errors made during development, for example, for some projects there is a discrepancy between the correctness of the development process and the correctness of the artifacts. Thus, the results of error estimates prove the necessity of differentiating the process estimates and estimating the correctness of the artifact.

Summary
The proposed method of verification allows to build routes for changing artifacts in the software production process, which allows to identify both errors in work products and incorrect organization of the process. The finite state machine implementing the proposed method of verification makes it possible to identify the causes of errors and to assess the impact of these errors on the further development process. The obtained automaton can be applied both to the whole process, and to a separate development stage.

Automatic verification of the process of developing software projects implemented in training groups has shown the possibility of reducing the error growth rate as the project is being implemented. The
successful application of the software complex, which implements the developed models of verification of the software development process, confirms the adequacy of the constructed models.

However, it should be noted that if the production process becomes more complicated in terms of the number of artifacts, or in the set of dependencies, the verification procedure will require more computing and time resources. Also, in the future it is planned to work on identifying the optimal number of inter-artifact dependencies, which allows to carry out production processes.

References
[1] A. V. Grigoryev, A. A. Kropotin and A. G. Ivashko, "Database schema method for automatic semantic errors resolving during information systems integration," 2016 IEEE 10th International Conference on Application of Information and Communication Technologies (AICT), Baku, 2016, pp. 1-7.
[2] Robert T. Futrell, Donald F. Shafer, Linda Shafer, Quality Software Project Management, Prentice Hall Professional, 2002.
[3] Eric J. Braude, Software design: from programming to architecture, J. Wiley, 2004.
[4] IEEE Guide to the Software Engineering Body of Knowledge – SWEBOK®. California, Computer society.
[5] Object Management Group (OMG). Meta Object Facility (MOF) Core specification Version 2.0 [WWW document]. URL http://www.omg.org/spec/MOF/2.0/PDF/
[6] Object Management Group (OMG). Unified Modeling Language infrastructure specification Version 2.0 [WWW document]. URL http://www.omg.org/spec/UML/2.0/Infrastructure/PDF/
[7] Object Management Group (OMG). Software & Systems Process Engineering Metamodel specification (SPEM) Version 2.0 [WWW document]. URL http://www.omg.org/spec/SPEM/2.0/PDF
[8] K.A. Bowen, Model Theory for Modal Logic: Kripke Models for Modal Predicate Calculi, Springer-Science+Business Media, B.V., 2013.
[9] John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman, Introduction to Automata Theory, Languages, and Computation, Pearson Education, 2014
[10] Justin S. Davis, Justin Davis, Robert Bryan Reese, Finite State Machine Datapath Design, Optimization, and Implementation, Morgan & Claypool Publishers, 2008.
[11] Peter Linz, An Introduction to Formal Languages and Automata, Jones & Bartlett Publishers, 2016.