DOI: https://doi.org/10.15276/hait.06.2023.19
UDC 004.41+005.2

Task execution flow management in the software development process under the minor change event

Oleksii B. Kungurtsev[1)]
ORCID: https://orcid.org/0000-0002-3207-7315; akungurtsev19@gmail.com. Scopus Author Id: 57188743440
Radim V. Chorba[1)]
ORCID: https://orcid.org/0009-0005-9879-4375; radim.chorba@gmail.com
[1)] Odessa Polytechnic National University, 1, Shevchenko Ave. Odessa, 65044, Ukraine

ABSTRACT

In modern project management methodologies, insufficient attention is devoted to the process of promptly responding to minor changes during task execution, which necessitate adjustments to the priorities of ongoing tasks. The existing approaches are not sufficiently detailed for a fundamental reassessment of priorities while such changes significantly impact project execution. The available materials and approaches do not provide ready-made solutions. This article proposes a task planning model during project execution. The model comprises the following key elements: Executor, Task Set, Task Execution Progress, and Calculation of Task Execution Quality Indicators. The Executor element contains information for identifying the developer and allocating their working time. It is anticipated that under exceptional conditions, a portion of non-working time may be scheduled for task execution. The Task Set element represents planned temporal characteristics and the priority of each task. The Task Execution Progress element contains information about actual dates, hours, and durations of segments during which the task was executed. The calculations of task execution quality indicators enable obtaining operational information about the progress of specific projects and assessing the effectiveness of process management. Basic algorithms for managing task sequences have been developed. The "Addition of a New Task" algorithm implements a task queue based on priority and start and end dates. The "Task Priority Change" algorithm envisages the possible repositioning of a task, as well as cases of task transfer to another executor or rescheduling tasks during non-working hours. Additionally, algorithms for notification of critical planning changes for dependent tasks ("Notification of Critical Planning Change for Dependent Tasks") and critical deprioritization of dependent tasks ("Notification of Critical Deprioritization for Dependent Tasks") have been developed. The proposed model and algorithms allow for accommodating micro-changes in the project and responding to their occurrence. The validation of research results in a real project demonstrated the effectiveness of the proposed model and algorithms while concurrently revealing a certain range of open questions requiring further consideration. Future research directions include the classification of micro-change scenarios, analysis of possible scenarios for suspending the execution of current tasks, and the development of scenarios and algorithms for selecting executors.

Keywords: Software development; project management; tasks planning; task queue; tasks priotitization; task priority change; project microchanges

***For citation*: Kungurtsev O. B., Chorba R. V. "Task execution flow management in the software development process under the minor change event". *Herald of Advanced Information Technology.* 2023; Vol. 6 No. 4: 297–307. DOI: https://doi.org/10.15276/hait.06.2023.19**

1. INTRODUCTION

Commonly known solutions for planning work on software projects, which allow for task scheduling over a specific work period (release, sprint in the Scrum methodology, etc.) [1, 2], [3] include products such as Atlassian Jira, MS Project, Primavera, Redmine, and to some extent, next-generation products like Trello, Asana, and similar ones. However, during the execution of a planned block of tasks, numerous unforeseen tasks arise, such as emergencies, urgent client inquiries, immediate management requests, and others, requiring resolution within a relatively short timeframe.

Due to the distinct nature of these tasks compared to changes in requirements, both functional and non-functional, which are typically outlined as product requirements, these tasks do not directly, impact the resulting functionality of the product. However, from a development process perspective, these tasks have a certain priority and are mandatory to address, thus consuming time that could be spent on resolving production tasks. In this article, we will refer to such tasks as "micro-changes". Since the nature and predicted volume of these tasks cannot be accurately known during the planning stage of a task block, and their impact on the team overall work is uncertain, planning for addressing these tasks is considered impractical.

Given that computational power of computer systems has significantly increased in the last 5

years, but engineers' ability to create software solutions has not grown proportionally [5], the issue of effective planning and control of such work is pertinent. Therefore, there is a problem of the impact of many sporadically occurring delays from tasks, the precise planning of which is not feasible, on the main project execution process.

In practice, research [6] shows that less than 15% of teams declaring the use of Agile/Scrum methodologies actually adhere to basic agile principles [1, 3], [5]. Therefore, it is reasonable to assume that in more than 85 % of teams working with flexible methodologies, engineers and managers rely on empirical approaches to reprioritize tasks in situations of micro-change occurrences.

Traditional planning technologies, such as Waterfall [2], do not provide a specific mechanism for responding to the emergence of unforeseen tasks of the discussed type (micro-changes). High-level project status analysis approaches within this family of methodologies, such as Critical Path [2, 7], allow for assessing the impact on the project delivery date post-factum (or at each specific moment when the deviation from the work schedule is already known).

The application of the Monte Carlo simulation method allows for assessing risk impact to the project and obtaining estimates of possible deviations from the project execution plan. Based on simulation data, the project manager can evaluate deviations and make decisions regarding the necessity of creating reserves regarding budget and execution timelines. While this approach allows estimating necessary reserves for each project phase at the macro level, it does not provide a specific mechanism for managing micro-changes in the project [8].

Thus, we identify a significant gap in project management methodologies in responding to micro-changes, both for projects managed by classical methodologies (Waterfall) and for modern methodologies (Agile).

The aim of this research is to find an approach that would enhance engineers' productivity when micro-changes occur and experimentally validate it.

2. EXISTING SOLUTIONS REVIEW

Articles [3, 9] thoroughly emphasizes the advantages of flexible methodologies (Agile) in a rapidly changing environment. Indeed, this family of project management methodologies is oriented towards satisfying the customer as a key value. However, the article does not propose an approach to managing micro-changes in projects. Furthermore, micro-changes affect both projects managed by flexible methodologies and classical methodologies, making the discussed problem relevant for both approaches to the project management.

The challenge of selecting an executor for a task is analyzed in article [10]. The authors suggest a mathematicl method for selecting a task executor based on each engineer's capabilities and task execution needs, using multi-factor analysis. However, the article does not consider the execution of an algorithm in a situation where engineers are already working on a block of tasks, and a new micro-change, to some extent, affects the tasks they are already performing or will perform in the current work block. The research also does not address the frequency of task switching as an important factor that needs to be avoided to protect engineers from burnout and, consequently, a sharp decrease in productivity [11]. The article only considers the application of the methodology for Agile methodology, while, as mentioned earlier, micro-changes can also impact projects executed using traditional methodologies.

Research [12] proposes a modern multi-criteria approach to task allocation based on machine learning technologies. Despite advocating its applicability to flexible methodologies in the article, we anticipate that the approach can also be used in classical project management methodologies.

In the study [13], the impact of context changes, which are a specific case of micro-changes, on the productivity of engineers working in a multi-project environment is analyzed. However, recommendations for mechanisms to minimize loss of production time are not provided.

In article [14], the author focuses on studying the reasons for interruptions in the work process, which in some cases also constitute micro-changes, and their impact on the productivity of engineers. The article offers only general approaches to reducing the negative effect of interruptions without specific action algorithms.

Considering characteristics of micro-changes such as their potential urgency and potentially short task execution period, it is necessary to note that the

described approaches do not fully consider the need to support a comfortable working environment. As noted in materials [15, 16], [17], frequent task switching, changing focus, and other abrupt changes in activity significantly negatively impact the productivity of engineers and, as a result, lead to emotional burnout, resulting in a catastrophic drop in engineer productivity and, consequently, failure of task delivery deadlines.

From the presented analysis, it is evident that there is a pressing need to develop an algorithm for task reprioritization for a team in the event of a micro-change during the execution of a planned block of work. Such an algorithm should aim to simplify the decision-making process during the micro-change handling [18, 19] and increase team productivity by reducing forced stoppage time and minimizing context switches.

3. GOAL AND TASKS OF THE RESEARCH

The aim of this study is to enhance the productivity of engineering teams by improving the task reprioritization algorithm considering the micro-changes in a project.

To achieve the stated goal, the following tasks need to be addressed:

1. Develop a model for task planning in the project, taking into account micro-changes.
2. Design an algorithm to respond to the addition of a new task in the current iteration of the project.
3. Develop an algorithm to respond to changes in the priority of a task in the current iteration of the project.
4. Create algorithms for notifying stakeholders of changes.

4. MODEL

It is proposed to present the scheduler model in the form of a tuple:

$$Planner = Performer, mTask, taskCompletion, Statistics >, \quad (1)$$

where *Performer* – engineer assigned to the task; *mTask* – set of the tasks; *taskCompletion* is flow of the task execution; *Statistics* – are calculations on the information stored in the model.

The "performer" element should contain information to identify the developer and allocate his working time (schedule):

$$Performer = PerformerName, Schedule >, \quad (2)$$

where *PerformerName* – name of the engineer; *Schedule* – his working schedule.

Taking into account the possible micro-changes described above, it is proposed to outline the schedule this way:

$$Schedule = restTime, nonWorkTime, workTime >, \quad (3)$$

where *restTime* – rest time, not subject to work planning; *nonWorkTime* – are non-working hours, may be scheduled to perform work under emergency conditions; *workTime* – working time, subject to work planning.

Rest time is to be presented by a tuple:

$$restTime = timeB, timeF >, \quad (4)$$

where *timeB* – beginninig of the time period (in form of a time of the day, for example, 23:00); *timeF* – ending of a time period (in form of a time of the day, for example, 7:00).

Non-working time is fragmented and consists of separate elements ($Piece_i$), for each of which the duration is indicated in the time of day, for example, 7:00 – 9:00, 17:00 – 23:00.

$$nonWorkTime = \{mPieceN_i\} i = \overline{1, h}, \quad (5)$$

where $piece_i = timeBN_i . timeFN_i >$.

Working time is fragmented by segments of time allocated for work:

$$workTime = timeB, timeF, mPiece >, \quad (6)$$

where $timeB$ – the beginning of a working time; $timeF$ is the ending of a working time; $mPiece$ – a set of time fragments: $mPiece = \{piece_j\}, j = \overline{1, k}$.

The number of fragments is determined by tasks and their distribution in time; if there is a single task, the set contains one element.

The maximum length of the fragment – *minuteMax* is equal to the duration of the working day, which is defined in the organization. The minimum length of the fragment – *minuteMin* is determined by the ability of the developer to switch to a new task and perform the minimally significant work for this task. Let us consider $minuteM \in \geq 1hour$.

Each fragment can be presented in way:

$$piece_j = idT_i, timeB_j, minuteN_j >,$$

where idT_i – task identifier, if there is no task it is set to zero; $timeB_j$ – time of the schedeuled beginning of the task execution; $minuteN_j$ – estimated amount of minutes for the task execution.

In exceptional cases, some tasks may be performed outside working hours. In this case, it will be presented similarly to working hours:

$$nonWorkTime = timeB, timeF, mPiece > \\ piece_j = idT_i, timeB_j, minuteN_j >.$$

Notification – communication to the person involved into a process in some role. It is used to inform the stakeholder or engineer about a change in the status of the task. It can be a reminder about the need to perform the task (sent to the engineer responsible for the task), information about a change in prioritization, a notice of cancellation of the task or other, which is sent to the customer requested the task.

It is suggested to submit the message in the form of a tuple:

$$Notification = TextT', Task, \\ NotrifType, NotifFreq>, \quad (7)$$

where $TextT'$ – notification text; *Task* – task to which notification is related; *NotifType* – type of the notification from the set ('remainder', 'priorityAlert', 'movedToQueueAlert', 'taskCancelledAlert'); *NotifFreq* is frequency of the notification.

Task is outlined as a set:

$$mTask = \{task_i\} i = \overline{1, nTask}, \quad (8)$$

where each task is a tuple:

$$Task = TextT, idT, TimeB, TimeF, \\ Dur, Priority, TimeFR, DurR, \\ Customer, Dependencies >, \quad (9)$$

$whereTextT$ – task description; idT – task identifier; $TimeB$ – expected time of the beginning of work on the task (date, time); $TimeF$ – expected (requested) task delivery date (date, time). This field may not be filled in if a specific date and time when the task must be completed has not been received from the customer. However, the team during iteration planning can set or change this value downwards in a situation where the execution of this task from the team's point of view blocks other tasks or for some reasons this task must be completed earlier; Dur – estimated task execution timeframe (amount of hours and minutes); $Priority$ – task priority.

Values of priority are: *P*0 – Critical (needs to be executed as soon as possible); *P*1 – High (execute with priority); *P*2 – Normal (default priority); *P*3 – Low (nice to have this task executed); *P*4 – Minimal (if there is no any higher priority task); $TimeFR$ – real task completion time (date, time); $DurR$ – real task execution timeframe (amount of hours and minutes); $Customer$ – requestor of the task;$Dependencies$ – set of the tasks which are dependent on this task (can be empty).

The execution of the task involves the determination of real dates, hours and duration of fragments during which the task was performed.

$$taskCompletion = idT, mPiece' >, \quad (10)$$

where $mPiece'$ – set of time segments during which the task was performed.

Each segment is defined by a start time and a duration

$$piece' = timeB', minuteN >.$$

Calculations based on model data. The data contained within the model allow for obtaining real-time information about the progress of specific projects and assessing the effectiveness of managing the design process.

Below are some possible computational quality management characteristics:

- Deviation between actual and planned task execution times.
- Degree of plan execution.
- Relative quantity of canceled tasks.
- Degree of execution of canceled tasks.
- Relative quantity of tasks not completed on time.
- Relative quantity of tasks not completed.
- Utilization of working time.

Task management algorithms

Algorithm 1 – Add a new task

Tasks are added to the execution queue based on priority and specific start and/or completion times if specified. Various scenarios of task addition during the iteration execution phase are illustrated in Fig. 1.

Fig. 1 depicts the initial work plan in the first block, consisting of three tasks with priorities of 1, 1, and 2, and durations of 12, 8, and 8 time units, respectively. The timeline is oriented from left to right, with task 1 being in progress for 4 time units.

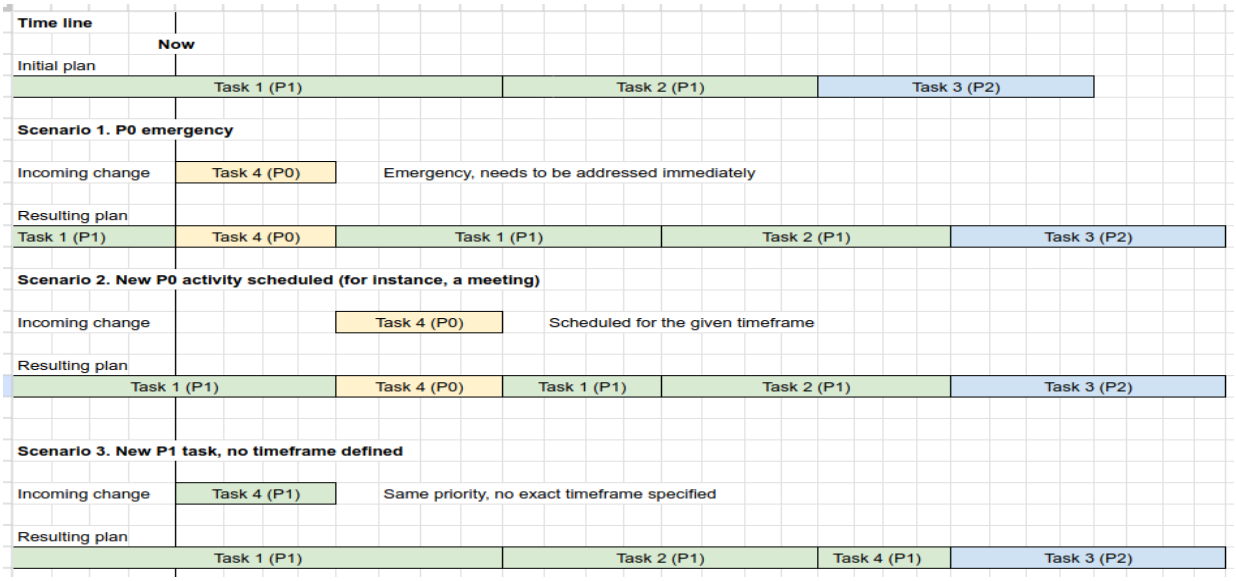

Fig.1. **Time diagrams of some task addition scenarios**

Source: **compiled by the authors**

Microchanges are represented in three blocks. Scenario 1 involves adding a task with an urgent priority (P0), which needs to be completed immediately. An example of such a task could be an emergency or unforeseen force majeure circumstances. In this case, the execution of Task 1 is urgently halted, and Task 4 is taken up for execution. After completing Task 4, work on Task 1 resumes. The priority of other tasks remains unchanged, but they are shifted along the timeline.

In scenario 2, Task 4 is added, also with an urgent priority, but the start and end times of this task are specified, and the start time does not coincide with the current time. Thus, the execution of Task 1 will be interrupted to perform Task 4, but this interruption will not be urgent. Similar to scenario 1, after completing Task 4, work on Task 1 resumes. The priority of other tasks remains unchanged, but they are shifted along the timeline.

Scenario 3 involves the appearance of Task 4 with the same priority as the task currently being executed. Unlike previous scenarios, the new task is placed in the queue according to priority, i.e., between tasks 2 and 3. In this case, only Task 3 progresses along the timeline.

Scenarios of adding a task with a priority lower than any of the planned tasks are not shown in the figure, as it is evident that such a task will be put to the end of the queue.

1.1. Task is received in the form of

$$Task_j = TextT_j, TimeF_j, Dur_j, Priority_j, Customer_j >.$$

1.2. Task identifier is created automatially.

1.3. $TimeB$ is set to the current date and time if not provided explicitly for the task.

1.4. The availability of time to complete the $Task_j$ is checked. The time free from work tl is identified in the $workTime$ set.

Options:

1.4.1. There is enough time $tl \geq Dur_j$.

1.4.1.1. If there is a task in the schedule with a lower priority than $Task_j$ and its due date allows to move it to the right (later time), then the shift and insertion is performed according to the priority of the tasks in such a way that priority tasks, as well as tasks with an earlier due date, are executed first.

1.4.1.2. For every task which was affected, $TimeS$ is changed by adding Dur_j value.

1.4.1.3. For the new task $TimeS$ is set to $TimeS_i + Dur_i$, where i is the index of the task which is pereceding to the newly added.

1.4.2. There is no enough time $tl < Dur_j$.

1.4.2.1. If task $Task_j$ has $Priority_j$ <= $Priority_i$, which means thet priority of the newly added task is lower than priority for the tasks which were planned already, for each of $mTask$ in the $workTime$ set, system reports the inability to plan the task $Task_j$ within the normal working time schedule for the given engineer.

The options proposed are:

– transfer $Task_j$ to another developer;

– request the possibility for the developer to perform the task within the *nonWorkTime* time

period (in case of the negative answer, this step is repeated but this option becomes unavalable);

– cancel the execution of the task in the current iteration and put the task in the queue for planning and execution in normal mode (to the backlog);

– reject the execution of the task, which means task will be not executed at all.

1.4.2.2. If task $Task_j$ has $Priority_j$ > $Priority_i$, which means that priority of the new tas is bigger than at least one of the tasks in $mTask$ in the $workTime$ timeframe, task $Task_j$ is inserted before the first task for which $Priority_j$ > $Priority_i$. For the rest of the tasks possibility to keep the task in the iteration is analysed. Rest of the tasks are analysed for the possibility to keep them within the current iteration, considering the $workTime$, corresponding to the initial sequence, which means decreasing $Priority_i$ and increasing $TimeF$. At the same time, *TimeS* is changed by adding the value of Dur_j

For the tasks which can not be kept in the $workTime$ timeframe, the following options are available:

– transfer task to another developer;

– request the possibility for the developer to perform the task within the *nonWorkTime* time period (in case of the negative answer, this step is repeated but this option becomes unavalable).

1.5. In the situation when the algorithm execution has reached this step, meaning the task is placed in the execution plan, and also for the very first task in $mTask$ condition $Prioity_j$ >$Prioity_0$ is true, which means that current task has a lower priority than the new one. The question arises about the appropriateness and the most opportune moment to interrupt the execution of the current task. This is exactly the question that I aim to address in my work. Due to operational necessity, such a need arises often; however, the delay caused by interrupting the ongoing task is usually much greater than the sum of the durations of the interrupting tasks. This introduces additional delays and risks to the execution of the current block of project tasks.

1.6. Information about changes in the task execution plan is stored in *Statistics*

1.7. Algorithm 3 and algorithm 4 are executed.

Algorithm 2 – Changing the priority of one of the tasks in the current iteration

If it is necessary to change the priority of the task, its position in the execution queue will change according to the new priority and the start and end dates, if they are set.

2.1. Priority change is accepted in form of $idT, Priority'$, where idT is the task identifier; $Priority'$ is the new priority

2.2. New position for the $Task_i$ is determined.

2.2.1. If $Priority < Priority'$ which means task priority is increased, the new position is found by traversing the $mTask$ by increasing the index until both conditions are met: $Priority' \geq Priority_i$ and $TimeF \leq TimeF_i$.

2.2.2. In other case, new position is found by traverse $mTask$ by decreaseing index until both conditions are met: $Priorty' \leq Priorty_i$ and $TimeF \geq TimeF_i$.

2.3. The task is moved to a new position, while the rest of the tasks are shifted while preserving the original order.

2.3.1. If the new position of the task is equals to zero, which means that the task must replace the task currently being performed, the question arises about the need to interrupt the execution of the current task, similar to the one considered in clause 1.5.

2.4. For each task starting from the new position, estimated start time is updated as $TimeS_i = TimeS_{i-1} + Dur_{i-1}$.

2.5. For each task starting from the new position, condition *TimeS* + *Dur* > *TimeF* is verified.

2.5.1. For those tasks where this condition is true, the task completion deadline will be violated. Options available are:

- transfer the task to another developer;
- offer the selected developer to complete the task in the *nonWorkTime* timeframe (in case of the negative answer, this iteration of the algorithm is repeated, but this option becomes unavailable);
- ignore (in this case, it is accepted as normal that the result of the task will be delivered later than the expected deadline).

2.6. Information about changes in the task execution plan is stored in *Statistics*

2.7. Algorithm 3 is executed.

Algorithm 3 – Notification of a critical change in the planning of dependent tasks

As a result of the operation of Algorithms 1 and 2, caused by a microchange, for some tasks the expected completion time may change in such a way that it turns out to be greater than what is requested in the task. In this case, it is necessary to notify the customer about a possible delay.

3.1. All tasks affected by the priority change are recursively detected as the result of execution of algorithm 1 or 2

3.2. For those with *TimeS* + *Dur* > *TimeF*, *Notification* is sent to the $Customer$. Notification has *Task* equals to the current task in question, *NotifType* equals to 'priorityAlert', *TextT* contains the details of the impact on the task rescheduling, including the potential affect on the *TimeF*.

3.3. Notification is not performed for the rest of the tasks.

Algorithm 4 – Notification regarding critical prioritization of dependent tasks

As a result of the execution of Algorithm 1, caused by a microchange, a decision may be made for part of the tasks to cancel its execution in the current iteration or its impracticality. In this case, it is necessary to notify the customer about a critical change in the work schedule for the task requested by him.

4.1. $Notification$ is sent for every task pushed out of the iteration. $Notification$ has *Task* equals to the current analysed task, *NotifType* equals to ‘movedToQueueAlert’, $TextT'$ with the details of the critical priority change and inability to meet the deadline.

4.2. $Notification$ is sent for each task which was cancelled on the step 1.4.2.1. $Notification$ has *Task* equals to the current analysed task, *NotifType* equals to *‘taskCancelledAlert’,* $TextT'$ with the information of the inability to execute the task.

5. PROBATION OF THE RESULTS OBTAINED

To conduct the scientific research, testing of the proposed models and algorithms was carried out on a real project. Since, as mentioned above, tasks in projects vary in duration, to validate the model and assess the efficiency of the proposed algorithms, it is necessary to gather information that closely approximates natural conditions.

To ensure the validity of the testing, the project team chosen as experimental must meet the requirements of a Scrum team, as described earlier, and the project performance results, as well as the measured characteristics, must be predictable to avoid the influence of fluctuations on measurement outcomes.

To meet these requirements, a real software development project was selected, susceptible to the influence of micro changes, and for which historical information was available to populate the model. Additionally, the actual application of algorithms in this project allows tracking changes in results.

As the experimental project, a development project in a stable state was chosen (project duration: 7 years, product release cycle: 3 months, project management methodology: Scrum, sprint duration: 2 weeks, team composition changes: absent, project team composition: 2 Scrum teams, totaling 15 engineers).

Before the experiment began, the team and the project were in a stable state, releasing planned sets of new features without significant quality problems (an average of 1 hotfix per year from 2019 to 2022).

There is no documented methodology for responding to micro changes in the project.

Therefore, the system chosen for the experiment meets all the specified requirements and has sufficient stability characteristics to exclude the influence of random factors on the experiment results.

The chosen evaluation criterion was the percentage of unfinished tasks at the end of a two-week development iteration (sprint). Preliminary analysis showed that this parameter has a certain cycle associated with the completion of the release cycle. Due to the organization work specifics, on the last iteration of the release cycle, the team focuses on testing (the percentage of unfinished tasks is minimal), while on the first iteration of the new release cycle, the team transitions to new functionality (the percentage of unfinished tasks is maximal). Subsequently, during the release cycle, the considered indicator typically decreases (see Figure 2, the considered indicator is shown on the graph with a thin line, the trend line is thick). Here, the completion of the release cycle falls on iterations 1, 7, and 13, and the start corresponds to iterations 2, 8, and 14.

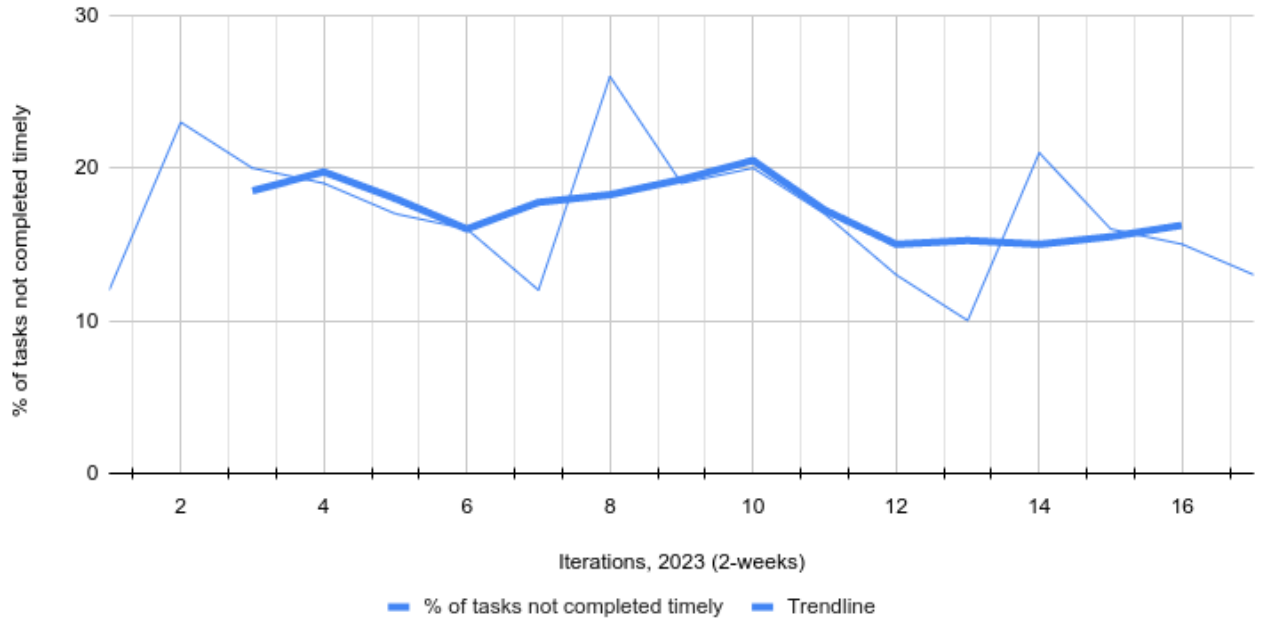

Fig.2. **The dynamics of changes in the share of incomplete tasks during iterations, 2023**
Source: **compiled by the authors**

The experiment started from iteration 12 (one iteration before the completion of the release cycle, considered the safest moment for implementing changes). In Fig. 2, the trend line for the percentage of tasks not completed on time is shown by a thick line, calculated as the average of 4 samples. It can be seen that the application of algorithms reduced the average percentage of unfinished tasks by the end of the iteration. The calculation shows that the average percentage of unfinished tasks since the start of the experiment was 14.7 %, whereas before the experiment, this indicator was 18.3%. Thus, the experiment confirmed the positive impact of applying algorithms on team productivity.

However, the experiment revealed several areas that require further improvement.

In the scenario of canceling a task scheduled for an iteration, additional free time arises. Typically, in practice, tasks are postponed to fill the freed time interval. Since this situation does not create additional risks for the project progress, its importance is less obvious, but it also creates a moment of uncertainty for engineers. The algorithm for this scenario should be formalized, despite the apparent simplicity of the situation.

The scenario where a task is completed before the deadline also does not create risks for the project, but this micro change also affects the start dates of subsequent tasks, similar to point 1.

The scenario of delaying task completion is similar to scenario 2, except that in this case, risks for the project are created, and notification may be required.

Observations 1-3 led to an understanding of the need to classify micro changes as such and analyze the reasons for their occurrence.

The relevance of studying scenarios of interrupting the execution of the current task was confirmed, as this scenario creates a moment of uncertainty in the team, often blocks the work of more than one engineer, necessitates frequent context switching, and, consequently, introduces significant risks to the project progress.

6. CONCLUSIONS

A model has been created that allows for the collection and analysis of task planning indicators in a project, taking into account micro changes. An algorithm has been developed to respond to the addition of a new task to the project. An algorithm has been developed to respond to changes in the priority of existing tasks. Algorithms for notifying stakeholders in case of changes in task execution deadlines, leading to violations of commitments regarding their completion, have been developed.

The model and algorithms were tested on a real project, and performance characteristics were collected and analyzed before and after the experiment, confirming the effectiveness of the algorithm.

Further research directions include:

- Clasterization [20] and/or classification of scenarios of micro changes and development of the formal response algorithms
- Analysis of possible scenarios for suspending the execution of the current task to increase team productivity by minimizing time losses for context switching and reducing engineers' stress levels.
- Development of scenarios and algorithms for choosing developer in situations where tasks need to be transferred to another engineer, targeting for minimizing time losses.

REFERENCES

1. Srivastava, A., Bhardwaj, S. & Saraswat S. “SCRUM model for agile methodology”. *2017 International Conference on Computing, Communication and Automation (ICCCA)*. Greater Noida: India. 2017. p. 864–869. DOI: https://doi.org/10.1109/CCAA.2017.8229928.

2. “PMBOK® Guide (2021)”. – Available from: https://www.pmi.org/pmbok-guide-standards/foundational/pmbok. – [Accessed: 28 July 2022]

3. Kolesnikova, K. V. & Lukianov, D. V. “Analysis of the effectiveness of combining the roles of scrummaster and product owner in ScrumTeams”. *Herald of Advanced Information Technology*. 2021; 4 (1): 67–74. DOI: https://doi.org/10.15276/hait.01.2021.6.

4. Morana, G. “The beginning of a cognitive software engineering era with Self-Managing applications”. *IEEE/ACM 1st International Workshop on Software Engineering for Cognitive Services (SE4COG)*. Gothenburg, Sweden. 2018. p. 1–4, https://www.scopus.com/authid/detail.uri?authorId=57197846381.

5. West, D. “What are the three scrum roles?” – Available from: https://www.atlassian.com/agile/scrum/roles. – [Accessed: 28 July 2022].

6. Kuhrmann, M. et al. “What makes agile software development agile?” In *IEEE Transactions on Software Engineering*. 2022; 48 (9): 3523–3539, https://www.scopus.com/authid/detail.uri?Authored=14015954200. DOI: https://doi.org/10.1109/TSE.2021.3099532.

7. Saradhi, B. P., et al. “Hesitant fuzzy project planning and scheduling using critical path technique”. *Turkish Journal of Computer and Mathematics Education*. 2021; 12 (6): 5272–5286. – Available from: https://www.proquest.com/openview/e67e5f6cf7af6b9342f39a41bf50d526/1 – [Accessed: 28 July 2022].

8. Galli, B. Jh. “Statistical tools and their impact on project management–how they relate”. *The Journal of Modern Project Management*, 2021, 9 (2): 129–143. https://www.scopus.com/authid/detail.uri?authorId=35931897100.

9. Biliavskyi, V. M. & Antoniuk, O. V. “Agile management tools and their impact on the effectiveness of project implementation” (in Ukrainian). *Aviation, Industry, Society: IV International Science and Practice Conf*. Kremenchuk: Ukraine. 2023. p. 712–714.

10. Aslam. W. & Ijaz, F. “A quantitative framework for task allocation in distributed agile software development”. *In IEEE Access*. 2018; 6: 15380–15390, https://www.scopus.com/authid/detail.uri?authorId=34972616300. DOI: https://doi.org/10.1109/ACCESS.2018.2803685.

11. Pachler, D., Kuonath, A., Specht, J., Kennecke, S., Agthe, M. & Frey, D. “Workflow interruptions and employee work outcomes: The moderating role of polychronicity”. *Journal of Occupational Health Psychology*. 2018; 23 (3), 417–427, https://www.scopus.com/authid/detail.uri?authorId=57195740187. DOI: https://doi.org/10.1037/ocp0000094.

12. William, P., Kumar, P., G. S. Chhabra & K. Vengatesan, “Task allocation in distributed agile software development using machine learning approach”. *International Conference on Disruptive Technologies for Multi-Disciplinary Research and Applications (CENTCON)*. Bengaluru: India. 2021. p. 168–172, https://www.scopus.com/authid/detail.uri?authorId=57433493200.
DOI: https://doi.org/10.1109/CENTCON52345.2021.9688114.

13. Tregubov, A., Boehm, B., Rodchenko, N. & Lane, J. A. “Impact of task switching and work interruptions on software development processes”. In: *Proceedings of the 2017 International Conference on Software and System Process (ICSSP 2017). Association for Computing Machinery*. New York: USA. 2017. p. 134–138, https://www.scopus.com/authid/detail.uri?authorId=56764212800.
DOI: https://doi.org/10.1145/3084100.3084116.

14. Abad, Z. S. H., Karras, O., Schneider, K., Barker, K. & Bauer, M. “Task interruption in software development projects: What makes some interruptions more disruptive than others?” In: *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering (EASE '18). Association for Computing Machinery*. New York: USA. 2018. p. 122–132. DOI: https://doi.org/10.1145/3210459.3210471.

15. Wiesche, M. “Interruptions in agile software development teams”. *Project Management Journal*. 2021, 52 (2), 210–222, https://www.scopus.com/authid/detail.uri?authorId=42962873700. DOI: https://doi.org/10.1177/8756972821991365.

16. Walker, A “Surviving the zombie apocalypse”. – Available from: https://www.infoq.com/presentations/career-skills-self-management. – [Accessed: 28 July 2022].

17. Khristich, A. L., Kolot, S. A. & Polic, V. “Designing a professional burnout correction program based on life-purpose orientations in wartime conditions”. *Herald of Advanced Information Technology*. 2023; 6 (1): 81–96. DOI: https://doi.org/10.15276/hait.06.2023.6.

18. Oborskyi, H. O., Saveleva, O. S., Stanovska, I. I. & Saukh, I. A. “The information technologies of anti-crisis solutions search in complex dynamic systems management”. *Applied Aspects of Information Technology.* 2020; 3 (2): 72–82. DOI: https://doi.org/10.15276/hait.02.2020.7.

19. Olekh, H. S., Prokopovych, I. V., Olekh, T. M. & Kolesnikova, K. V. “Elaboration of a Markov model of project success”. *Applied Aspects of Information Technology*. 2020; 3 (3): 191–202. DOI: https://doi.org/10.15276/aait.03.2020.7.

20. Kungurtsev, O., Zinovatna, S., Potochniak, Ya. & Novikova, N. “Development of methods for pre-clustering and virtual merging of short documents for building domain dictionaries”. *Eastern-European Journal of Enterprise Technologies*. 2020; 5 (2(107)): 39–47, https://www.scopus.com/authid/detail.uri?authorId=57188743440. DOI: http://doi.org/10.15587/1729-4061.2020.215190.

Conflicts of Interest: the authors declare no conflict of interest

Received 04.09.2023
Received after revision 30.11.2023
Accepted 11.12.2023

DOI: https://doi.org/10.15276/hait.06.2023.19
УДК 004.41+005.2

Планування потоку завдань в умовах мінорних змін в процесі створення програмного забезпечення

Кунгурцев Олексій Борисович[1)]
ORCID: https://orcid.org/0000-0002-3207-7315; akungurtsev19@gmail.com. Scopus Author Id: 57188743440
Чорба Радім Валерійович[1)]
ORCID: https://orcid.org/0009-0005-9879-4375; radim.chorba@gmail.com
[1)] Національний університет «Одеська політехніка», пр. Шевченка, 1. Одеса, 65044, Україна

АНОТАЦІЯ

У сучасних методиках управління проектами недостатньо уваги приділяється процесу оперативного реагування на мінорні зміни під час виконання завдань, які вимагають коригування пріоритетів поточних завдань. Існуючі підходи недостатньо деталізовані для фундаментальної переоцінки пріоритетів в умовах суттєвого впливу таких змін на виконання проекту. Наявні матеріали та підходи не надають готових рішень. У цій статті пропонується модель планування завдань під час виконання проекту. Модель включає наступні ключові елементи: Виконавець, Набір Завдань, Прогрес Виконання Завдань та Обчислення Індикаторів Якості Виконання Завдань. Елемент Виконавця містить інформацію для ідентифікації розробника та розподілу його робочого часу. Передбачається, що в особливих умовах частину неробочого часу можна запланувати для виконання завдань. Елемент Набору Завдань представляє заплановані часові характеристики та пріоритет кожного завдання. Елемент Прогресу Виконання Завдань містить інформацію про фактичні дати, години та тривалість сегментів, під час яких виконувалася завдання. Розрахунки індикаторів якості виконання завдань дозволяють отримати оперативну інформацію про хід конкретних проектів та оцінювати ефективність управління процесами. Розроблено основні алгоритми управління послідовностями завдань. Алгоритм «Додавання нового завдання» реалізує чергу завдань на основі пріоритету та дат початку та закінчення. Алгоритм «Зміна пріоритету завдання» передбачає можливе перепозиціонування завдання, а також випадки перенесення завдання на іншого виконавця чи перепланування завдань під час неробочих годин. Крім того, розроблено алгоритми для сповіщення про критичні зміни планування для залежних завдань («Сповіщення про критичні зміни планування для залежних завдань») та критичного зниження пріоритету для залежних завдань («Сповіщення про критичне зниження пріоритету для залежних завдань»). Запропонована модель та алгоритми дозволяють враховувати мікрозміни в проекті та реагувати на їх виникнення. Підтвердження результатів дослідження на реальному проекті продемонструвало ефективність запропонованої моделі та алгоритмів, водночас виявивши певний ряд відкритих питань, які потребують подальшого вивчення. Майбутні напрямки досліджень включають класифікацію сценаріїв мікрозмін, аналіз можливих сценаріїв призупинення виконання поточних завдань та розробку сценаріїв та алгоритмів для вибору виконавців.

Ключові слова: програмне забезпечення; керування проектом; планування завдань; черга завдань, пріоритети завдань; зміна пріоритетів завдань, мікрозміни проекта

ABOUT THE AUTHORS

Oleksii B. Kungurtsev - PhD, Professor, Professor of the Department of Software Engineering. Odessa Polytechnic National University, 1, Shevchenko Ave. Odessa, 65044, Ukraine
ORCID: https://orcid.org/0000-0002-3207-7315; akungurtsev19@gmail.com. Scopus Author Id: 57188743440
***Research field*:** Methods and means of increasing the productivity of information systems; communication means with automated systems in natural language

Кунгурцев Олексій Борисович - кандидат технічних наук, професор кафедри Інженерії програмного забезпечення Національного університету «Одеська політехніка», пр. Шевченка, 1. Одеса, 65044, Україна

Radim V. Chorba - Postgraduate student of the Department of Software Engineering. Odessa Polytechnic National University, 1, Shevchenko Ave. Odessa, 65044, Ukraine
ORCID: https://orcid.org/0009-0005-9879-4375; radim.chorba@gmail.com
***Research field**:* Methods and means of increasing the productivity of the development teams

Чорба Радім Валерійович - аспірант кафедри Інженерії програмного забезпечення Національного університету «Одеська політехніка», пр. Шевченка, 1. Одеса, 65044, Україна