electronics

Article

Classification of Task Types in Software Development Projects

Włodzimierz Wysocki [1,*], Ireneusz Miciuła [2] and Marcin Mastalerz [3]

[1] Department of Software Engineering and Cybersecurity, Faculty of Computer Science and Information Technology, West Pomeranian University of Technology, 70-310 Szczecin, Poland
[2] Department of Sustainable Finance and Capital Markets, Institute of Economics and Finance, University of Szczecin, 70-453 Szczecin, Poland
[3] Department of Computer Science in Management, Institute of Management, University of Szczecin, 70-453 Szczecin, Poland
* Correspondence: wwysocki@zut.edu.pl

Abstract: Managing software development processes is still a serious challenge and offers the possibility of introducing improvements that will reduce the resources needed to successfully complete projects. The article presents the original concept of classification of types of project tasks, which will allow for more beneficial use of the collected data in management support systems in the IT industry. The currently used agile management methods—described in the article—and the fact that changes during the course of projects are inevitable, were the inspiration for creating sets of tasks that occur in software development. Thanks to statistics for generating tasks and aggregating results in an iterative and incremental way, the analysis is more accurate and allows planning the further course of work in the project, selecting the optimal number of employees in task teams, and identifying bottlenecks that may decide on faster completion of the project with success. The use of data from actual software projects in the IT industry made it possible to classify the types of tasks and the necessary values for further work planning, depending on the nature of the planned software development project.

Keywords: software; knowledge management; reasoning; information extraction; rule mining; knowledge acquisition; engineering

check for updates

Citation: Wysocki, W.; Miciuła, I.; Mastalerz, M. Classification of Task Types in Software Development Projects. *Electronics* **2022**, *11*, 3827. https://doi.org/10.3390/electronics11223827

Academic Editor: Maria Liz Crespo

Received: 3 November 2022
Accepted: 19 November 2022
Published: 21 November 2022

1. Introduction

The contemporary intensity of changes in the surrounding reality due to technological progress makes management by economic units extremely demanding and difficult. Therefore, it is necessary to react quickly and effectively to changes that create new conditions for business activity. In the era of knowledge-based economy, information, information systems co-created by it, and related information technologies are extremely valuable and inextricably linked with knowledge [1]. The usefulness of IT systems is enormous and directly influences the increase of the possibilities of management units by reflecting their innovativeness and technological potential, which is of great importance when making strategic business decisions.

Globalization, the era of e-economy, and computerization are the topicality of modern economic activity [2]. That is why the production of an IT product (IT product) is so fundamental. Managing the production of IT products is a new scientific and technological discipline that has emerged at the interface between computer science and management engineering. All economic entities that base their activities on IT solutions are interested in the practical results of research in this discipline. In addition, it should be stated that, indirectly from all kinds of improvements in software development and management, any activity that uses any IT software will benefit. The digitization of the economy results in the need for reliability and security of emerging applications, IT systems, or transaction services. A lack of reliability of IT software can cause enormous losses. That is why it is so important to develop and create IT software that is as reliable as possible. Therefore, the work efficiency of teams producing IT software—characterized by a cyclical nature—is so important. Moreover, the literature highlights the dependence of the implementation effectiveness of subsequent software development stages on current project stages and the maturity level of the IT product.

The software development process is constantly evolving; there are technological changes and ways of organizing the work of project teams. In the knowledge economy era, most economic activities require systems and all kinds of IT products [3]. This forces the appropriate acceleration of software development, while at the same time maintaining the required quality. That is why it is so important to have the right team to implement often complex and innovative IT solutions. On the other hand, numerous project teams cause problems in effective communication in the group and cause the need to optimize work organization. To ensure this optimization, tools and IT products designed for managing software development processes are often used.

Traditional approaches to IT project management have become insufficient due to the high variability of requirements and the need to change the work organization of project teams [4]. It is particularly noticeable in innovative software development projects for the e-economy, where flexibility in the production process and originality of solutions are required. Therefore, despite the support of software development processes with increasingly better tools and systems, the pursuit of optimal use of project resources—including human teams—is still a considerable challenge. Since the beginning of the 21st century, the agile approach has been increasingly used in the practice of software development. The literature on the subject shows a significant improvement in the number of successful projects and the optimization of the use of resources necessary in software development through the use of continuously developed agile methods. This is particularly important due to the constant changes taking place in the requirements for the software being developed [5]. Therefore, traditional methodologies cannot be successfully applied to the numerous changes that are natural in innovation projects because they are extremely static and, apart from the first phase, consist of a fixed number of tasks. The next stage of the methodology development is based on iterations, which assumes that we learn about new requirements in the development process; requirements change and the existing ones are detailed. However, in the current agile methodologies, the requirements are incomplete and we do not know the details; this is also due to the nature of the innovative projects [6]. We have a set of requirements in the form of epics and user stories. Then, in the course of the software development process, creating new requirements becomes the cause of new tasks. Another source of new implementation tasks is the tests of current versions that detect defects or reveal new implementation possibilities. Automation of this process by the task generator in agile methodologies allows for the creation of appropriate increments of tasks; the number and the nature of which depends on the size and type of the project. On the other hand, the iterative measurement of tasks allows for subsequent iteration planning, with the assumption of the invariability of certain aspects for the best estimation. Therefore, planning methods based on summary work are nowadays of great practical importance for the management of manufacturing processes.

Project management systems for software development collect a lot of data about tasks and information related to them necessary to perform the work and the necessary exchange of information between the project team [7]. However, there is still room for improvement when it comes to providing information that will optimize IT project management. Because the data and information collected in the systems are semantically poor, it is necessary to analyze the works that are performed during the implementation of tasks to correctly classify them, which will also allow the estimation of their size, thus increasing the quality of planning. The isolation and classification of the characteristic types of tasks, and the work performed within them, will allow for the automatic execution of analyses and estimations to create insightful statistics and reports necessary for optimal forecasting decisions. In addition, taking into account the knowledge about the current technological and business components will give an appropriate insight into the current state of work in the project and will allow for more optimal planning of the software development process.

This work aims to identify the impact of the type of tasks on efficiency and to indicate those factors that positively affect the effectiveness of software development teams. The model of task types was built based on the data analysis of real software projects from the financial sector, managed by the Jira system. Connecting the model with the Jira system enables easy data acquisition for analysis and increases its commercial potential. A separate abstract layer of the model, in combination with a dedicated database, supports the possibility of creating interfaces to other IT project management systems. The article attempts to classify project tasks, thus determining the necessary expenditure on tasks of various types. In conjunction with the results of research on tasks created during the production process and cyclical works, it allows for planning projects. Linking task types with project team roles allows the simulation of project work and supports project management by identifying bottlenecks in the manufacturing process and avoiding over-employment.

The article discusses the basic concepts of the context in which software development projects are implemented and the development of methodology for the optimal management of these processes. At the same time, the principles and practices, as well as the lifecycle of software development projects, are characterized, which is important for the possibility of introducing improvements in this process. The concept of the programming process model is inspired by the agile approach to managing software as it is developed. Extending the original approach to the roles of team members and task types allows more precise work planning in the project and management of the project team, including detecting bottlenecks or unused resources. The article discusses the programming process model that is oriented toward the manual recognition of task types, thus enabling effective support for planning tasks carried out in innovative IT projects.

2. Literature Review

Software development in IT companies is mainly carried out through appropriate organization of the work of project teams. Unfortunately, in projects involving the search and creation of new solutions where the variability and unusual nature of ideas and requirements are natural, there is a constant need to improve the management of this process. Due to the innovative nature of IT projects, project teams are burdened with high risk. This forces the search for optimal project management methods and techniques that will allow for better control and use of the company's resources. Nowadays, analytical methods and tools embedded in IT systems are a great help in supporting project management.

Traditional software development project management methodologies are currently being replaced by, or supplemented with, agile methodologies. The development of project management methods in the IT industry has resulted from dissatisfaction with the small number of successful projects. A big problem was discovered in the management of innovative projects, where rigid and strongly formalized traditional software development methodologies did not work and even made it difficult to introduce changes and the proper functioning of projects [8]. Agile methods are used primarily due to the desire to reduce the number of errors, to shorten the time needed to create finished products, or to reduce the total production costs [9]. However, as noted by J. Shore and S. Warden, when making decisions regarding the implementation of agile practices, it is difficult to unequivocally state greater successes in the implementation of IT projects [10]. This is due to the variety of interpretations of the concept of "IT project success" and the fact that the use of the same methodologies in different companies from the same industry results in different results [11]. There is a common view in the literature that the key to success in project management is learning about appropriate techniques and tools [12]. However, the instrumental layer of an appropriate project management methodology alone will not ensure its success if not properly applied by its members, because the most important part of software development projects are the people [13]. Therefore, an appropriate balance should be found between the hard (budget, schedule, and implementation time) and soft (communication, changes, motivation, and competencies) elements of the project. Therefore, regardless of the methodology used, project management is closely related to people management, and practice shows that most of the problems affecting project success result from the omission of the "purely human aspects" of team management [14]. Accordingly, this article aims to identify the main problems related to people management in the success of an IT project by enabling effective task planning carried out in innovative, and thus variable, software development processes.

Since the beginning of the 21st century, we have witnessed a dynamic development in agile methodologies [15]. This is due to the adaptation of the management process to projects with a high degree of innovation. In these cases, hard methodologies such as PRINCE2 and PMI PMBoK do not work due to the detailed and long-term planning stage that is not able to take into account future changes [16,17]. With these characteristics of the projects, the too-high level of standardization of activities does not work, because there are often unsuccessful attempts at establishing the project lifecycle and detailed requirements already at the project initiation stage. Even though this approach gives a lot of comfort to the implementers, in the case of innovative projects, it does not meet the real needs that will be known only at further stages of implementation. The progress of work on the innovative project reveals the necessity to repeatedly verify many assumptions, actions, and plans, because, only as the implementation progresses, knowledge and actual visualization of the developed solutions in the shape of the final product are acquired. Additionally, certain paths for reaching specific results turn out to be ineffective only after the verification and testing phase. External changes should also be mentioned, i.e., changes that are beyond the control of project teams, e.g., changes in regulations and legal norms or the current market situation. Often, consistent plan implementation leads to the creation of functionalities that will not be adapted to reality or that will be useful only after costly modifications. Therefore, this article attempts to classify the characteristic types of tasks in software development projects that will allow for more effective planning and adaptation of the required work to dynamically changing reality.

All factors that make it difficult, or even impossible, to precisely define and describe the results of the project at the stage of its definition result from the following elements that occur when creating software (especially innovative software) [18,19]:

- Customers and users are not sure what result they expect and have difficulties with formulating requirements;
- Many details and solutions will only be revealed during the project implementation;
- Details of implementing innovative projects at the planning stage are not feasible;
- The way of thinking changes during software development;
- External forces (such as competitors' products or services) lead to changes or expansion of requirements.

In addition, the implementation of innovative IT projects is associated with the risk of other irregularities. Kruchten (2004) lists several main problems that can affect any project, which are [20]:

- Ad hoc requirements' management;
- Ambiguous and imprecise communication;
- Fragile system architecture;
- Overwhelming complexity and undetected inconsistency in requirements, designs, and implementation;
- Insufficient testing;
- Subjective assessment of the project status;
- Uncontrolled introduction of changes;
- Insufficient automation.

The above factors and elements of project implementation management, and the inability to solve them in accordance with the traditional approach, contributed directly to the development of methods for making traditional methodologies more flexible and, as a result, prepared the ground for the Manifesto for Agile Software Development that announced in 2001 the principles for agile software development methodologies [21,22]:

- Individual people and their interactions (rather than processes and tools);
- Working products, i.e., software (more than comprehensive documentation);
- Cooperation with the client (more than negotiating a contract);
- Reacting to changes (rather than sticking to the plan);
- Customer satisfaction is the highest priority and should be achieved through early and continuous delivery of valuable software;
- Variability of requirements applies to both new and changing requirements during the project implementation; the adaptive software development process is able to keep up with changes in advance;
- Frequent delivery of working software in periods from a few to several weeks, with shorter time frames being preferred;
- Communication that should be realized directly;
- Working software, because this is the most important measure of progress in the implementation of works in the project;
- Adaptability of software development consists in the ability to maintain an appropriate pace of work during the implementation of the project by all members of the project team and to adapt the product (software) to frequently changing requirements.

The principles of an adaptive approach to software development project management indicate the direction for project teams, while specific practice is necessary for the actual implementation of works [23]. The process structure and specific practices create a minimal flexible framework for self-organizing teams. IT tools are essential to accelerate software development and to reduce costs. Contracts are crucial for the development of the customer–supplier relationship. Documentation supports communication [24]. However, the key issue is to provide the project team with feedback to answer the question of where the team currently is in the software development process [25]. This is possible thanks to iteration and incremental software lifecycles. The essence of the iterative process is the frequent delivery of working pieces of software (successive increments) that implement selected sets of functions that together make up the usefulness of the final product. The iterative software development cycle leads to a management style where long-term plans are fluid, while a stable plan can be created for a short period of time. Iterative and incremental software development leads to completely new relationships with the business client and different principles of the project team's functioning.

The concept of the iterative and incremental programming development cycle as a remedy for dilemmas of the e-economy era has resulted in the creation of new methodologies for IT project management [26]. These methodologies, called agile, do not cut off completely from document-oriented formalized traditional methodologies, but have specific features (adapted to the requirements of modern software development projects) [27,28]:

- Adaptive, not predictive; traditional methodologies do not cope with frequent changes in requirements, while agile ones accept them on principle;
- People-oriented, not process-oriented;
- Creative style of work.

Organizations are increasingly implementing digital transformation plans, due to the threat of disruptions, in order to keep pace with the growing pace of business. Agile software development plays a huge role in this process. Many of the digital workflows in use today are based on agile principles. Thanks to a flexible scalable IT infrastructure, cloud computing is evolving in line with the needs of agile software development. The DevOps concept removes the traditional distinction between software development and operations. The software is used as a tool in SRE–DevOps implementation and systems management and automation of operational duties [29]. CI/CD methodologies confirm that software will change frequently and provide tools that help developers deliver new code faster [30]. Agile methodologies come in a variety of forms to meet the needs of any project [31]. Even though agile approaches are different, all of them are based on the key ideas contained in the agile approach. For this reason, any framework or behavior that complies with these principles is termed agile. Regardless of the specific agile approaches that a team decides to apply, the benefits of an agile methodology can only be fully realized through the collaboration of all parties involved [32]. In recent years, a large number of agile software development project management methodologies have emerged. The most popular are [33–36]:

- Kanban. The phrase "Kanban" (which comes from Japanese) is translated as "visual board or signboard" and is associated with the idea of "just in time". The Kanban concept gradually found its way into agile development teams. This approach develops project management using visual methods. The Kanban board—divided into columns to illustrate the flow of the software development process—is used to supervise projects. Teams benefit from greater visibility as they can track their progress through each stage of development and can plan upcoming tasks to deliver the product on schedule. To ensure that team members always have a smooth workflow and are aware of the appropriate stage of development, this method requires comprehensive communication and transparency.
- Scrum. The agile scrum development approach, which is represented by multiple development cycles, is one of the best-known examples of an agile methodology. Scrum breaks down development processes into units known as "sprints", much like Kanban. By maximizing and devoting time to developing each sprint, only one sprint is managed at a time. Consistent results, emphasized by scrum and agile techniques, allow designers to modify priorities in such a way that any incomplete or delayed sprint attracts more attention. The daily scrum is where activities are coordinated to develop the best sprint strategy; the scrum team has exclusive design roles such as scrum master and product owner.
- XP (extreme programming). The extreme programming (XP) methodology places great emphasis on collaboration, dialogue, and feedback. It emphasizes the constant improvement and happiness of the client. This approach uses sprints, or short development cycles, similar to scrum. It is created by the team to create a highly effective and productive atmosphere. The extreme programming technique is very helpful in a situation where the customer's needs are continuous and changing. It encourages developers to accept changes to customer requirements, even if these requirements appear at an advanced stage in the development process. In extreme programming, the design is evaluated from the outset by gathering input data that increase system performance. Additionally, it offers a quick way to meet any customer requirements.
- Crystal Clear family, a concept developed by Alistair Cockburn, an expert in object-oriented design. Each project class may have a different methodology (Crystal Clear Method). Crystal is a collection of smaller agile programming approaches that include Crystal Yellow, Crystal Clear, Crystal Red, Crystal Orange, and more. It was first introduced by Mr. Alistair Cockburn, one of the key figures in creating the Agile Manifesto for Software Development. Each one has a unique structure that distinguishes it from the others based on variables such as system criticality, team size, and project priorities. The type of Crystal agile approach is selected depending on the criticality of the project or system. Crystal strives for on-time product delivery, regularity, reduced administration with high user interaction, and customer satisfaction, similar to other agile approaches. The Crystal family, which has earned the title of Lightest Ways of Agile Methodology, promotes the idea that each system or design is unique and requires different practices, processes, and principles to be applied to obtain the best results.
- Adaptative software development, an extensive adaptive methodology developed by Jim Highsmith.
- Dynamic system development. This method of dynamic systems development was created to meet the demand for a unified industry charter for fast software delivery. The software development process can be planned, run, managed, and scaled using the comprehensive structure provided by DSDM, which maintains that quality and on-

time delivery can never be compromised and that design modifications are always to be expected. This belief is based on eight principles and a business-based methodology.

- Lean development, a "lean" software development. The basic idea of the approach is the elimination of losses understood as elements that do not add any value to the product. The aim of such action is to deliver the finished product to the customer as soon as possible. Lean software development is based on the values and principles of adaptive project management. The term "lean software development" is considered by Mary Poppendieck and Tom Poppendieck, who, in their book *Lean Software Development: An Agile Toolkit,* presented, among others, the seven main principles of lean management and a set of 22 techniques supporting the approach.

Agile management methodologies are a group of methodologies that are characterized by an adaptive and variable approach to managing software development projects. Additionally, agile methodologies were developed much earlier than the agile manifesto itself. The first works on adaptive project management methods date back to the 1980s (an example is the rapid application development methodology) and the concept of agile methodologies was introduced in the mid-1990s [37]. The idea of a time frame is a well-defined process dedicated to software development control at the lowest level in an iterative cycle with several review points. Reviews help ensure the quality and efficiency of software development. By delivering the software on time at the lowest level, the timely production at the highest level (i.e., the project level) is ensured [38]. The basic principle of the project plan is to prepare a schedule of planned increments and, within them, the planned time frames, which will create a complete project schedule capable of changes with emerging new requirements. The use of the time frame technique, together with the MoSCoW prioritization technique, ensures no delays in project implementation and the delivery of ready-made software that will meet business goals within a given time [39].

Projects with a high degree of innovation are very difficult to include in a complete schedule and scope of work. Therefore, adaptive methodologies describe functionalities (i.e., independent elements of the subsystem), which in subsequent releases can be quickly changed and handed over for implementation. Agile methodologies, as opposed to traditional methodologies, rule out the validity of long-term planning [40,41]. Therefore, the plans are speculative and not deterministic. This allows you to adapt to all types of changes that appear during the implementation of the project. Additionally, the distinguishing factor of agile methodology is a strong emphasis on the cooperation and integration of the project team, because only in this case is the smooth flow of information and effective communication ensured. The general scheme of the project lifecycle in the case of agile methodologies is based on five phases, indicated by J. Highsmith [18,42]:

1. Creating a vision of the project by defining its scope and principles of cooperation within the project team;
2. Planned speculation by specifying functional elements for the product, creating time-limited iteration plans and major milestones of the project itself;
3. Exploration, i.e., a quick start by providing the user with functional elements and implementing production methods that minimize the costs of changes, including the creation of an adaptable and collaborative project team;
4. Adaptation, i.e., the assessment of the product, process, project, and project team, and then the correction of existing plans and design practices;
5. Closing the project by creating a database of experiences for the next project.

Agile project management methodologies require an appropriate level of project maturity for organizations and project teams [43]. In addition, agile methods continue to be improved in order to most effectively respond to the needs of managing still-innovative software development projects. Hence, there are so many methods that are still looking for new, more perfect, solutions; although, they undoubtedly already seem to be better adapted than the classic methods to dynamically changing project environments. Implementation based on iteration allows for effectively adapting signals that come from both inside the project and from the external environment.

Classifying specific types of tasks in software development projects will allow for determination of the requirements, the time necessary, and the expenditure that will be needed to implement them. This will allow for more effective work on further adaptive project planning methods. Linking task types with the roles of the project team will allow you to simulate project work, which will support project management by identifying bottlenecks in the software development process. In addition, it will allow for avoiding over employment and will allow support for quick "what if" simulations. At the same time, the introduction of the task classification in terms of business and technological components, in conjunction with the employee competency model, will allow for automatically selecting the composition of the project team and optimally managing the team; this has the highest priority in agile methodologies. This is of great importance for the optimization of software development process management and leads to the development of perfect methods in response to the dynamism of real-world changes. It will have replicative and predictive capabilities to plan the project, to simulate it during changes, and to detect bottlenecks during the entire process. In this article, experiments were carried out on many real IT projects to classify task types and to attempt to find the relationship between the nature of the planned project and the specificity and number of these tasks.

3. Materials and Methods

The model of task types was built on the basis of the data analysis of real software projects from the financial sector, managed by the Jira system. Table 1 contains summary data of these projects. In four projects, the team worked in accordance with traditional methodologies, to which more and more elements of the agile approach were introduced. In two projects, the teams worked in accordance with the scrum approach. It is quite a large data set that allows for the analysis of phenomena related to modern manufacturing processes. The total number of project issues exceeds 30,000, which makes it possible to use artificial intelligence methods. The total duration of the projects is approximately 22 years, which does not mean that historical records are dealt with. Project teams worked on most projects in parallel.

Table 1. Projects, the data of which were used for research on task types.

Project	Methodology	Number of Issues	Effort Person/Hour	Duration of the Project		Team Size		
				Years	Months	MIN	Avg	Max
P1	Hybrid	10,773	67,444	8.0	96.3	2	14	26
P2	Hybrid	2492	17,063	3.6	43.6	1	12	25
P3	Hybrid	7754	41,530	3.2	37.9	1	19	31
P4	Hybrid	1212	7453	2.6	30.6	1	8	21
P5	Scrum	5466	55,354	3.4	40.9	6	22	39
P6	Scrum	3949	27,397	1.9	22.8	1	23	41
	Total	31,646	216,241	22.7			98	

In Jira, teams use issues to describe and track specific tasks to be done. Issues are the basic building blocks of projects. The issue can be of a specific type. The most common type of issue is task, which in practice is used for many purposes. The second most frequent type is bug, which describes the defects found in the developed software and the work needed for fixing them. The full list of types used in the projects with their frequency is shown in Figure 1.

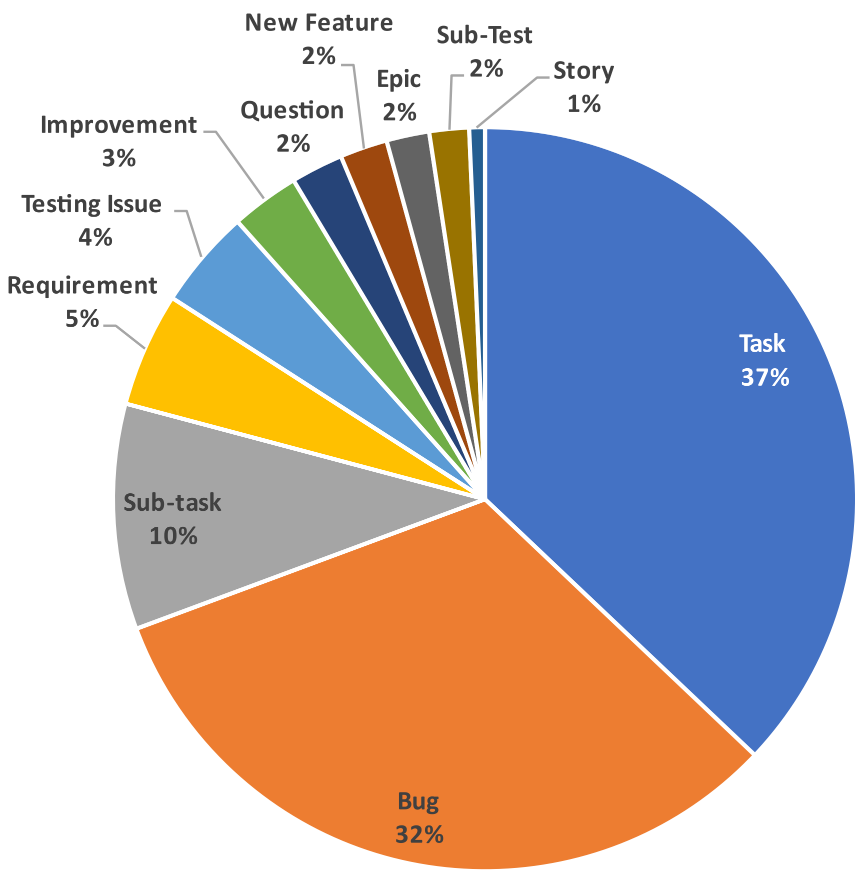

Figure 1. The occurrence frequencies of issue types in the researched projects.

The most commonly used issue types are task, bug, and subtask. The summary determines the idea of the actions to perform. The description field contains a detailed description of the work to do. A significant feature of the issue is its state. Tracking issue state changes allows for the recognition of the performed work. The workflow describes the values of the state field and the allowed transitions between them. Figure 2 shows a typical simplified workflow used in the analyzed projects.

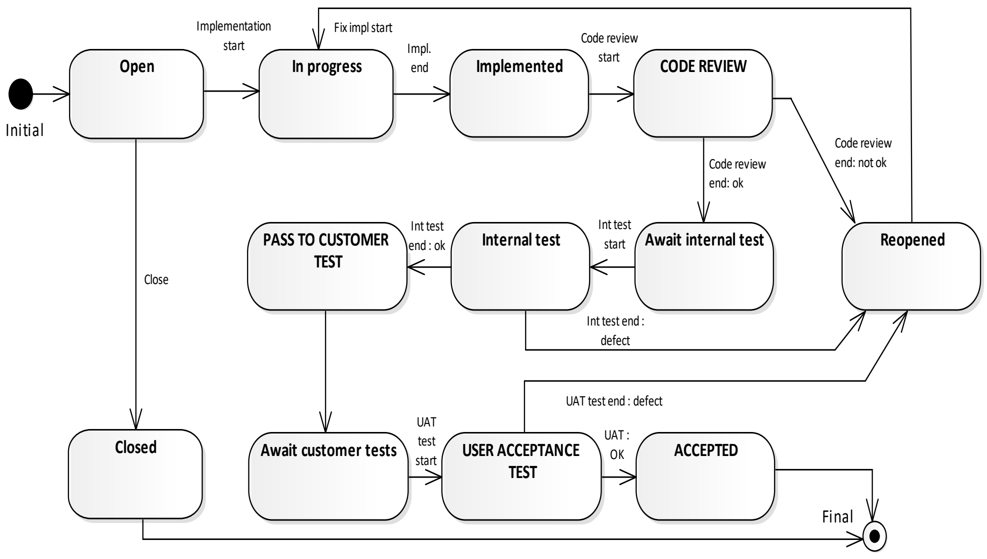

Figure 2. A typical simplified issue workflow used in projects.

As shown in Figure 2, transitions between issue states correspond to performing basic actions in implementing new system functions or fixing bugs. For example, the DevOps (development and operations) methodology helps to establish cooperation between developers and operators to automate the continuous delivery of new software, which is expected to contribute to shortening the development cycle and to creating high-quality software [44,45]. Another development of DevOps is the concept of development, security, and operations (DevSecOps), which at the same time is designed to integrate security methods with the software development process, where security measures are built in to ensure the integrity and availability of the application [46].

A valuable feature of the Jira system is storing the history of changes in the value of issue elements, including the state. Storing history supports the tracking of issue execution. Not all issues use the state to track work. The Jira system allows employees to register working hours devoted to work on an issue. For some types of work, there is no need to keep track of their state, as they are repetitive works performed as needed. In this case, the possibility of registering working hours is sufficient.

The presented approach is based on a precise division of the development process into activities and roles in line with the RUP methodology, adapted to hybrid and agile processes [47,48]. The previous series of articles described the agent–object model of the manufacturing process (AGOMO), which was first used to assess the maturity of RUP processes, then to plan hybrid water–scrum–fall processes [49,50]; it became an inspiration for the research presented here.

As we showed in the previous section, the types of Jira issues are insufficient to clearly define the work's purpose and type. The model was created in order to fill the gap that prevents linking the work carried out with the composition and competencies of the project team. The combination of these two areas will allow for a more precise quantitative analysis of the projects' works and the detection of the causes of the observed phenomena. It is a way to optimize the efficiency of development processes and to increase the level of maturity of project teams and organizations [51].

Software projects consist of tasks that a project team performs to build an IT system [52,53]. The task represents the Jira issue. Each of the tasks performed has a clearly defined goal. This goal can be, for example, implementing a requirement, testing a component or the entire module, administering environments, or managing a project; it determines its type. Members of the project team perform the tasks. Roles define the competencies and responsibilities of those carrying out the tasks. One person can have many roles; one role can have many people.

The task of implementing system functions requires performing some essential subtasks. They include implementation, i.e., the creation of component source code, code review, creation of unit tests, testing and verification of functions, and acceptance tests. Such tasks correspond to the issue, the state of which changes according to the workflow shown in Figure 2. The implementation tasks that consist of subtasks are named stateful tasks in the model.

Each task and subtask contains the number of project team members' working hours. Task and subtask types enable association efforts with the roles of project team members. They help to recognize bottlenecks or over-employment in completed IT projects and to avoid them during project planning.

Issues whose states do not change during the project are represented in the model by recurring tasks. They have been called recurring because a single task of this type can be performed as needed for the project's duration. For example, this might be a project development management task performed by an administrator when defects arise or when new software components need to be configured. For recurring tasks, you can calculate the average labor intensity in a period and, on this basis, assume what the fixed costs of the project are [54]. Genuinely recurring tasks are, for example, meetings such as daily stand-up meetings, workshops with clients, or steering group meetings.

The authors analyzed the tasks of the studied projects in terms of their type. The results, in the form of graphs showing the percentage number and the sizes of stateful and recurring tasks, are shown in Figure 3. The chart on the left shows the ratio of the number of stateful tasks to the recurring tasks in the projects. The chart on the right shows the ratio of the effort of stateful tasks to recurring tasks.

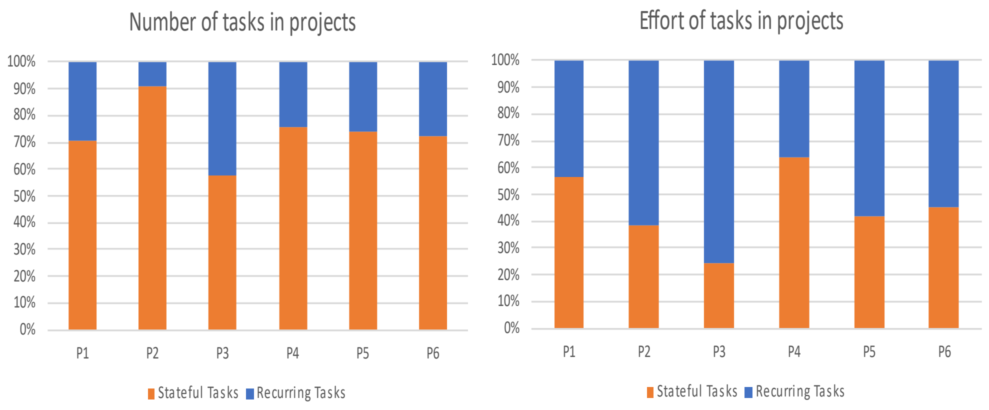

Figure 3. The number and effort of stateful and recurring tasks in the researched projects.

The case of the P2 project is significant, where the number of stateful tasks is greater than the recurring tasks (amounting to over 90%), while the effort of these tasks is slightly over 40%. It is very interesting, because stateful tasks are responsible for implementing the functions of the built system and for fixing the detected defects in it. From the developer's point of view, this workload should probably be the greatest. From the point of view of a researcher of software development processes, this phenomenon arouses great curiosity. In order to satisfy it, it is first necessary to precisely define the types of tasks in the projects, which this article implements on the basis of actual and implemented projects from practical activity.

4. Results

The classification of task types is based on the division of tasks into stateful and recurring tasks. By definition, stateful tasks represent work related to the implementation of system components and the fixing of defects. Stateful tasks are processed by a subtasking algorithm. Recurring tasks are responsible for the remaining works. Recurring tasks can be classified on the basis of summary and extended-text descriptions included in the issue. Initially, these tasks were classified manually. Subsequently, simple classification algorithms were created based on keyword searches. Recurring tasks are also broken down into subtasks that correspond to the registered work.

Figure 4 shows a diagram of the classification algorithm activity. The algorithm first checks how many times the issue has changed its state; if at least three times (more than open and close), the workflow is the basis for dividing the task into subtasks. The split algorithm tries to create subtasks (e.g., development, testing, code review). For this, it uses the value of the state before and after, the time of the change, the role of the person, and the registered works and then assigns completed work to the created subtask. The result is a stateful task.

If the task did not change state or if the algorithm did not detect any subtasks, the task type is determined by searching for keywords in the text description. When it fails, the tasks go to a spreadsheet, where they are manually classified. Then, the algorithm checks who was working on the task. If many people worked on a task on one day, it is a recurring group task (e.g., stand-up and other meetings). If one person worked on a task on one day, it is a recurring individual. The way of dividing the recurring task into subtasks depends on the individual/group classification. If the programmer and the tester alternately execute a task, it is stateful; the algorithm divides them into subtasks according to the roles of the team members.

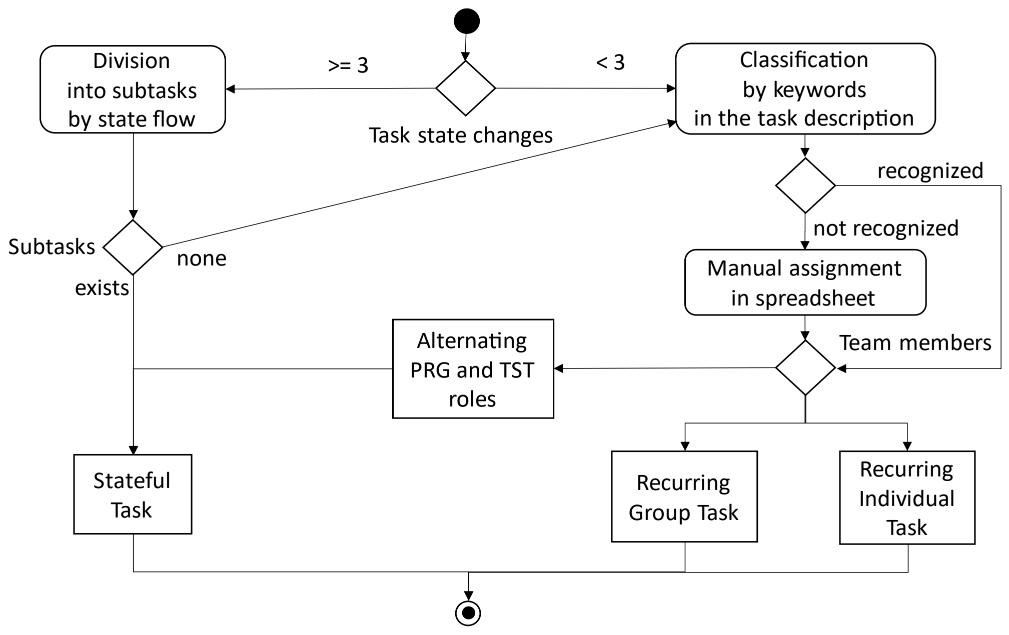
Division into subtasks by state flow
>= 3
Task state changes
< 3
Classification by keywords in the task description
recognized
not recognized
Manual assignment in spreadsheet
Subtasks
none
exists
Alternating PRG and TST roles
Team members
Stateful Task
Recurring Group Task
Recurring Individual Task

Figure 4. Algorithm of task classification.

The algorithm for breaking stateful tasks into subtasks is very complex due to the many ways that users use Jira to work on a project issue. Searching for keywords is not very accurate and supporting it by manual classification makes it impossible to use the classification in practice. Therefore, it is planned to replace these solutions with the NLP (natural language processing) classification. However, there are some good points to manual ranking. During the work on the algorithm, the process of the manual analysis of recurring tasks detected more than 50 types of tasks that were aggregated into three groups containing 14 main types of tasks.

Stateful tasks consist of tasks for implementing new features and for fixing bugs reported by customers and testers. Recurring tasks are divided into three main groups: implementation, meetings, and organizational tasks. A complete list of the main task types is provided in Table 2. The task types listed in the table form a hierarchical structure divided into types and groups.

The task classification algorithm—classifying tasks performed alternately by programmers and testers as stateful tasks—increased the number and effort of state tasks. An updated version of the graphs in Figure 3 is included in Figure 5. The changes are significant. For example, for project P3, the percentage of stateful tasks increased from 58% to 66% and the percentage of stateful task efforts increased from 24% to 43%.

The development of task types and an algorithm enabling automatic classification of the researched projects allowed for a detailed analysis of the tasks of the researched projects. Below, we present effort charts (Figure 6) for six groups of task types in the researched projects.

The P1 project was implemented in a hybrid methodology. It is the longest project among the researched, with a duration of more than 8 years. The project went through many phases of implementation, delivery, and maintenance. Therefore, the values and the ratio of effort in the project were averaged. They can serve as a benchmark when compared with other hybrid and traditional designs.

The P2 project was carried out in the hybrid methodology. It is characterized by a large number of hours used for various types of meetings. On the other hand, the very small scope of work in the management field suggests that the project may have been managed collectively. The very little work involved in fixing defects found by clients indicates that time spent in meetings was well spent.

Table 2. Types of stateful and recurring tasks used by the model.

Kind	Group	Type
Stateful	DEV requirement, function, feature	DEV-PRG: analysis, design, programming
		DEV-TST: testing, verification
	BUG-DEV defect from internal tests	BUG-DEV-PRG: programming
		BUG-DEV-TST: testing, verification
	BUG-CLI defect found by customer	BUG-CLI-PRG: programming
		BUG-CLI-TST: testing, verification
Recurring	R-DEV development	A&D: analysis and design
		BLD: build versions, find the causes of errors, create scripts
		CFG: software configuration
		CUST: training, technical support, commuting
		MIGR: data migration from previous systems
		RQM: requiremets engineering
		RVW: code review
		TST: subsystem, module tests, manual tests
	R-MEET meetings	M-CUST: other customer meetings
		M-DEV: development team meetings with external teams
		M-INT: development team internal meetings
		M-STDNP: daily stand-up meetings
		M-WRK: requirements workshop with the customer
	R-ORG organizational	ADM: administration of environments and servers
		PM: project management

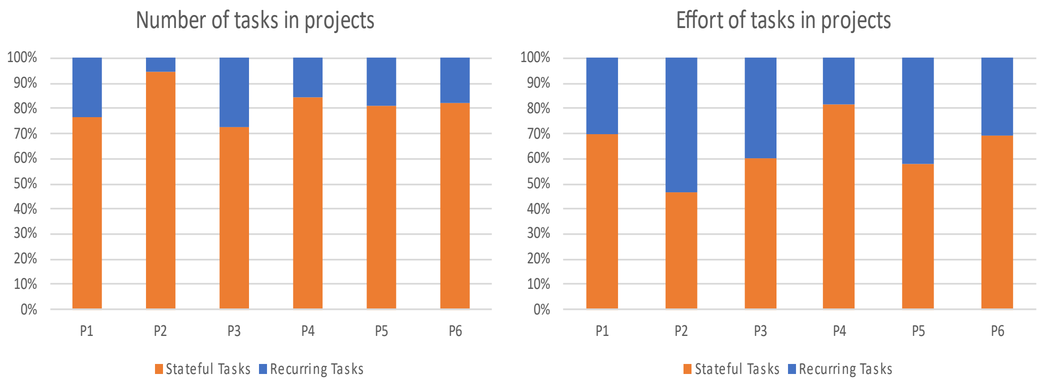

Figure 5. Updated numbers and effort of stateful and recurring tasks.

The P3 project was carried out in a hybrid methodology. It has high administrative and management costs. The ratio of repairing defects found by customers to repairing defects found by testers is interesting. There is no such tendency in the other projects, except perhaps for the P4 project. This may indicate inaccurately defined requirements or difficult contact with the customer.

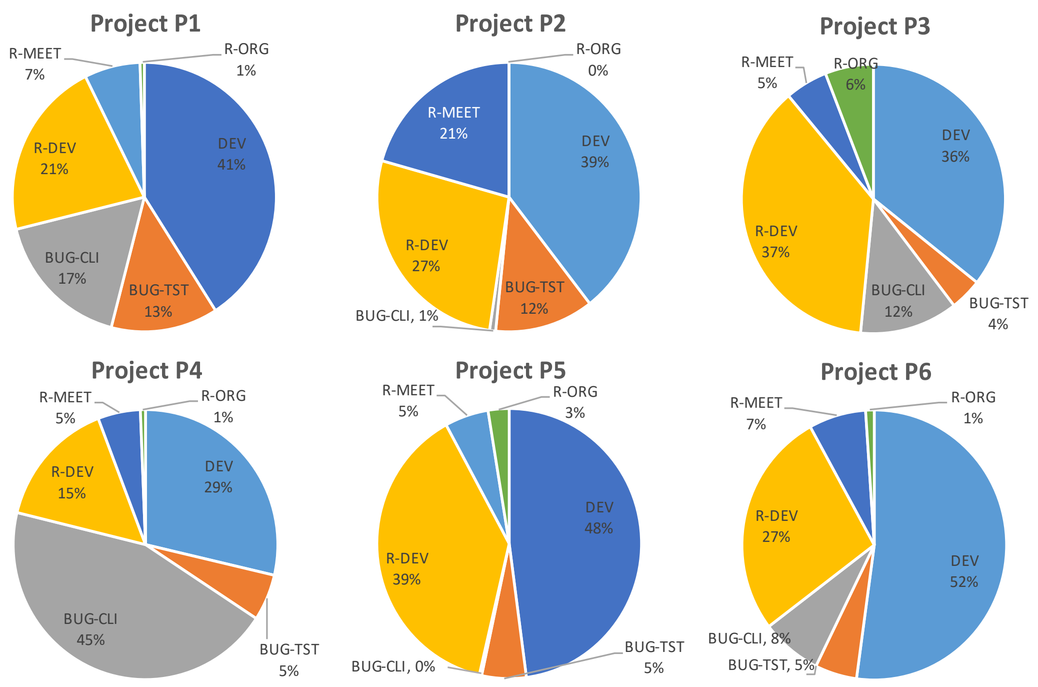

Figure 6. Effort of the six main groups of task types in projects.

The P4 project was implemented in the hybrid methodology. The distribution of the workload of tasks in the P4 project differs from others in terms of a very large amount of work to fix the defects detected by customers. The cost of this work is one-and-a-half times greater than that for implementation tasks and five times greater than the cost of repairing defects detected by the development team. The situation is similar to the P3 project, only the management and development costs are much lower. It is possible that the P4 project is in the maintenance phase of a project, with a large number of defects.

The P5 project was implemented using the scrum methodology. At the high level of abstraction given by the task type group analysis, no difference can be seen between this project and the hybrid projects. What is important is the lack of resources to rectify defects reported by customers. It is very possible that the project did not enter the customer implementation phase and was not put into production.

The P6 project was also implemented using the scrum methodology. More than half of the work was devoted to product implementation. The product has likely been delivered to the customer, as indicated by 8% of the efforts to fix defects reported by customers. The meetings have a significant share in the work on the project, which is consistent with the scrum methodology. The outlays for the maintenance and management of the project are small.

Graphs of total efforts by groups of task types give a very synthetic view of the projects. We can get a deeper look at the differences between projects by focusing attention on the detailed effort of cyclic task types. Figure 7 shows radar charts of recurring task effort, shown as a percentage of total recurring effort. The details of the R-DEV group from Figure 6 are shown here. The left side of the graph shows the labor intensity of meetings, cooperation with the client, administrative work, and management. The right side of the chart shows the expenditure on recurring development tasks. The charts differ from each other to reflect the characteristics of the projects. The charts show the "fingerprints" of the projects, making it possible to identify their detailed characteristics and to compare them with each other.

In most of the charts, the left organizational and management side dominates over the right developer side. The P1, P2, P3, and P6 projects follow a similar pattern in the recurring works chart, which indicates greater expenditure on meetings and management than on development work. Analysis and design play a large role in the P3, P5, and P6 projects.

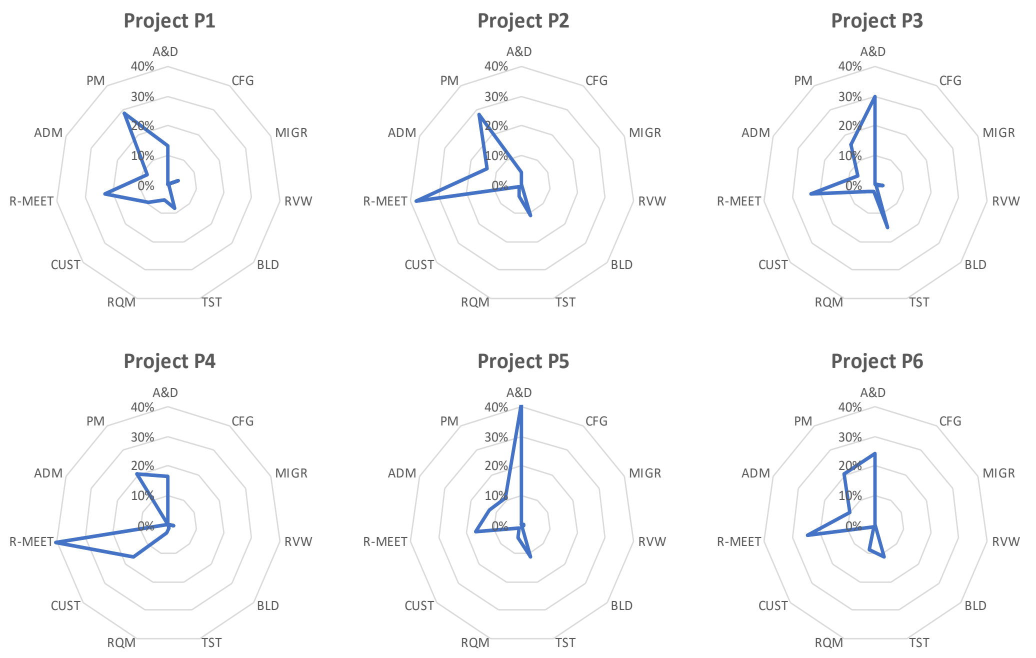

Figure 7. Structure of expenditure on recurring tasks in the researched projects.

Comparing the details of recurring work in projects allows you to determine the minimum, average, and maximum values of recurring work, which will enable the use of linguistic variables in the work for project planning. On the basis of static summary graphs, the nature of the project can be determined and projects can be compared with each other. It is also important to determine the area of expenditure of recurring tasks in projects and to determine their minimum, average, and maximum values. This is important for project planning. The entered types of tasks can be useful to answer the question of how tasks are created and performed during the development process.

The previous chapters introduced the division of tasks performed in the software process into stateful tasks related to the implementation of new tasks and recurring ones, with works performed periodically. This division may lead to the assumption that stateful tasks related to new features are mainly created at the beginning of the project. As for bugs, it would be prudent to assume that they arise after implementing certain requirements or features. The very name of recurring tasks (e.g., daily stand-ups) suggests that they are performed in equal intensity throughout the manufacturing process. Creating new tasks is very important when planning the development process, because the implementation of tasks cannot be started when they have not yet been created. This fact limits the size of the planned project team.

The model of task types and the research carried out on actual projects show what this case looks like in reality. Figure 8 shows when the state tasks in the researched projects were created. The charts do not show the number of created tasks, rather their effort, which better reflects the total size of tasks created in a month. DEV—new functions; BUG-DEV—defects detected by testers; BUG-CLI—defects detected by the customer (see Table 2 for details).

Most of the charts show that new features are developed throughout the life of the project. This phenomenon may come as a big surprise, especially since the P1-P4 designs were produced according to a hybrid approach in which traditional practices had a large share. The situation in the long-term P1 project is understandable, because it consists of many phases and, in each of them, new functions were created to be implemented. The P2 project is an exception among the examined projects, because new functions are created during the first 8 months at the beginning of the development process and then, for 24 months, they are implemented, and repairs of defects detected by the development team are created.

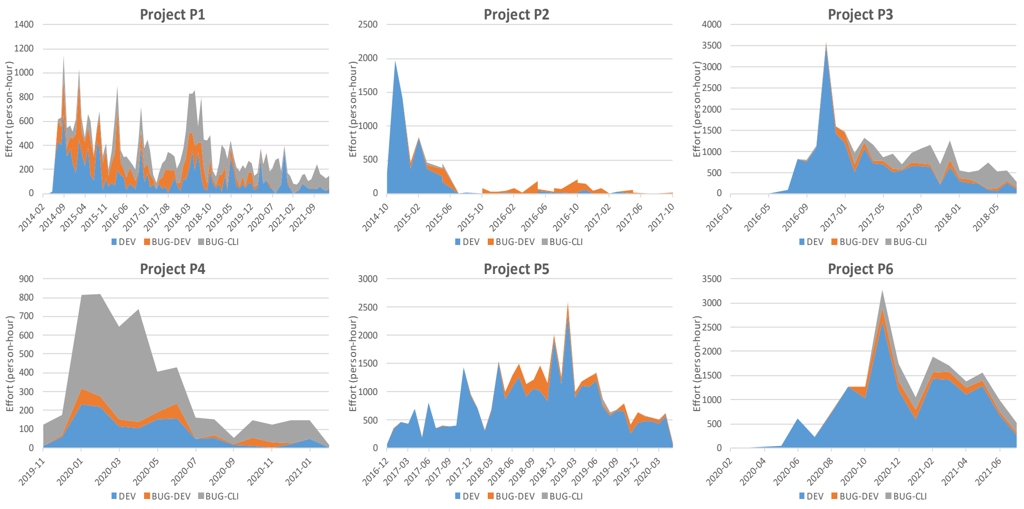

Figure 8. Stateful tasks created in the development process of the researched projects.

The graphs of the P3–P6 projects show, however, that new functions are created until the very end of the manufacturing process, although to a lesser extent. The reasons for this are interesting. Does it result from a long-term process of acquiring new requirements parallel to the development process? Or, maybe the reason is getting to know the details of the requirements obtained earlier? Unfortunately, the data placed in the Jira system do not answer these questions directly, because they do not take into account the requirements engineering processes. An interesting phenomenon is also the periodic increase and decrease in both the work on new functions and the repair of detected defects.

Work on the implementation of stateful tasks proceeds in a different rhythm than the creation of new stateful tasks. The number of man-hours used per month for stateful assignments depends on the number of people on the project team and their assigned roles. It should be taken into account that the team also performs recurring tasks. Figure 9 shows the monthly expenditures on the execution of stateful works in the researched projects.

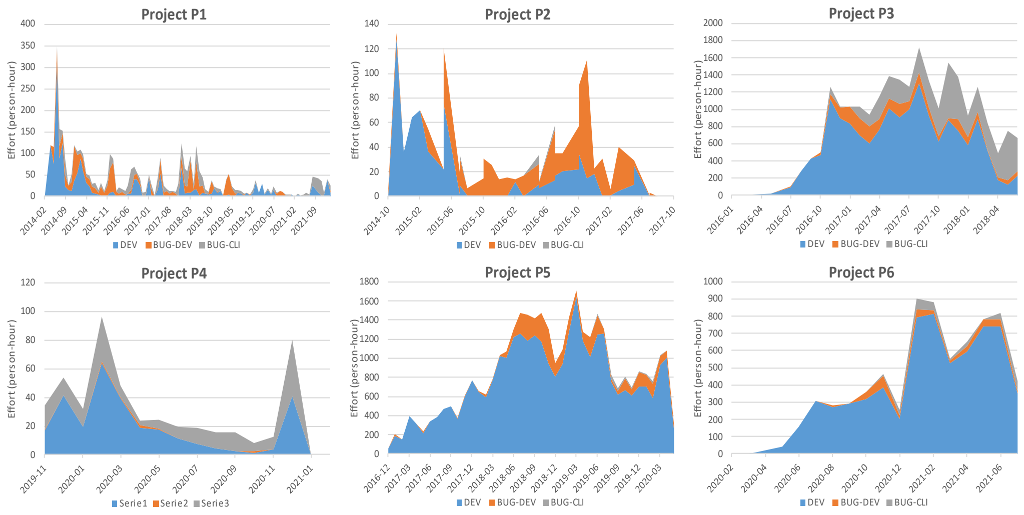

Figure 9. Work on stateful tasks in the researched projects.

The charts show that, in most projects, bug fixes detected by the development team are delayed in relation to the implementation of the functions of the developed software. Even more delayed is the repair of errors detected by the client, because they are the last detected. This phenomenon is best seen in the graphs of the P3 and P6 projects. Comparing the work charts with the charts of the created tasks (see Figure 8) gives better insight into the project. For example, in the P2 project, after creating many thousands of new features, there is a break for several months, and then defects detected by the project team are created. The P2 project work graph shows that there was no break in the project, the work was less intensive, but at that time defect fixes and then the implementation of new features were ongoing.

The answer to the question of what the effort of recurring work in the software development process looks like can be found in Figure 10. The graphs show a similar periodicity as the graphs of the effort of stateful tasks. The source of these periodic disturbances is very interesting. Probably, to some extent, the outlays for stateful and recurring work are complementary, i.e., increases in the first graph correspond to decreases in the second.

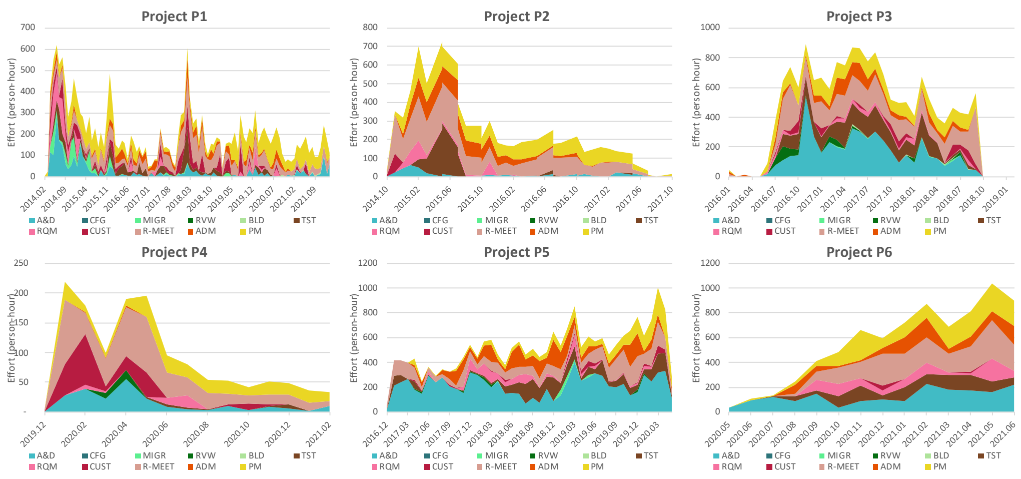

Figure 10. Work on recurring tasks in the researched projects.

The model of task types, in conjunction with the roles of the project team members, allows for the analysis of the work carried out in the project. On this basis, it is possible to trace the implementation of tasks and to recreate the composition of the project team. The next step in the development will be the possibility of simulating the work in the software development process or of planning the composition of the project team based on task plans.

The project team consists of the roles and the number of jobs of people employed in a given role. The analysis of actual project data is the source of the model, hence the lack of certain roles in the team, e.g., the role of an analyst. The current set of roles is defined as follows: ADM—administrator, PM—team leader, PRG—developer (who also deals with design and collection of requirements), TST—tester. The team consists of roles and the number of positions for a given role.

The adopted set of roles is not consistent with the agile approach represented by the scrum methodology. The scrum team chiefly consists of three roles: the scrum master, the product owner, and the development team. Developers are everyone belonging to the development team who are involved in software development. However, in practice [21], it is worth distinguishing the role of a tester (whose main activities are software testing and quality assurance), a programmer (who creates production code), and an administrator (who manages development and production environments and tools supporting the development and implementation of emerging software).

Figure 11 shows the concept of the relationship between task types and the roles of project team members. It consists of task types, roles, and two kinds of connections: simple and proportional. A simple link between a task type and a role determines that tasks of a specific type are performed by members of the project team with that role. For example, DEV-PRG tasks are performed by people with the PRG role and BUG-CLI-TST tasks are performed by people with the TST role.

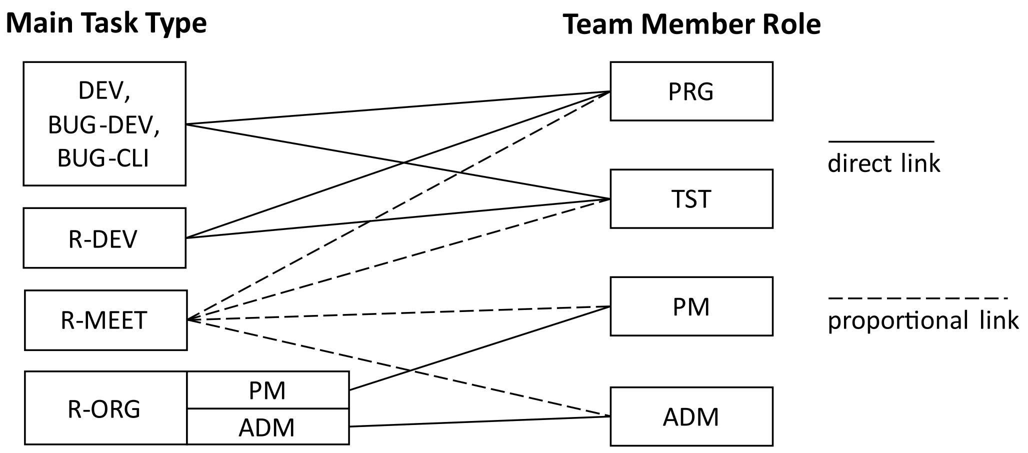

Figure 11. Relationship between the main types of tasks and the roles of project team members performing them.

A proportional link exists between meeting tasks and all roles of the project team. The idea of proportional connection is to divide the man-hours allocated to meetings among people from the project team in proportion to the number of people performing a given role. For example, there are 100 person-hours of meetings recorded in a month and programmers completed 72% of the work per month, so 72 meeting person-hours per month are added to the workload of the developers' tasks.

The workload of tasks performed monthly by roles is divided by the adopted average number of hours worked by a person per month. The score is the number of positions for that role. The number of positions is quantized to 1/4 and rounded up. The adopted average number of 165 h of work per month takes into account only non-working days. It does not take into account holidays and possible dismissals due to the employee's illness. This number can be changed freely.

Linking task types to the roles of project team members enables the approximation of the team composition needed to complete the project. Table 3 presents the monthly work of the P6 project, broken down by the main types of tasks and the composition of the project team reconstructed on their basis.

Table 3. Monthly work of the P6 project and reconstructed composition of the project team.

		Actual Monthly Effort				Team Number of Jobs		
Date	**DEV**	**R-DEV**	**R-MEET**	**R-ORG**	**ADM**	**PRG**	**TST**	**PM**
2020-05	155	-	-	11	0.25	1	0.25	0.25
2020-06	307	3	-	8	0.25	1.75	0.25	0.25
2020-07	284	41	21	106	0.5	2	0.5	0.25
2020-08	290	113	65	105	0.5	2.5	0.5	0.5
2020-09	361	191	133	128	0.25	3	1.25	1
2020-10	467	200	129	293	0.25	3.5	1.25	2
2020-11	255	130	256	158	0.5	2.75	0.5	1
2020-12	901	212	202	292	1	6.25	1.5	1.25
2021-01	881	193	208	328	1.25	6.25	1.5	1.25
2021-02	555	138	152	293	0.75	3.25	1.75	1.5
2021-03	650	151	198	333	0.75	4.25	1.5	1.75
2021-04	782	269	289	326	0.5	6.5	1.25	2
2021-05	821	110	196	404	1	5.75	0.75	1.75
2021-06	423	54	114	184	0.25	2.75	0.75	1.25

The number of people on the project team goes up and down in line with monthly stateful and recurring tasks. The team has been growing since the beginning of the project. In July and October 2020, it was at its highest; the number of positions was 10.25. After that, the team shrunk and the project ended in December 2020. There was a maximum number of 6.5 programmers and 1.75 tester positions per project. It is interesting that, in April and October 2020, there were two positions for the project manager in the team. Often, projects employ people to perform organizational and support work, for example, managing issues in the Jira system, maintaining Kanban boards, or creating reports.

5. Model Verification

The classification algorithm, according to Figure 4, consists of a method of dividing into subtasks and assigning task types based on a set of keywords built from the manual classification of P3 project tasks. Since the total number of issues in all projects was large, it was difficult to classify them manually. The research is intended to serve as a proof of concept, so the same set of keywords was used to classify the tasks of the other projects.

The method of dividing tasks into subtasks is closely related to the classification of tasks, because the types of subtasks affect the distinction, for example, of whether implementation or testing has been performed. If a subtask is not correctly identified, labor intensity will not be assigned to it and it will not be included in the accuracy indicator $\left(EA_p\right)$. Subtask division is complex, because people who record work in the JIRA system do it in many different ways. The method of subtask division developed most first for the P3 project, then was adapted to other projects. The primary indicator of the effectiveness of project task classification $\left(EA_p\right)$ is the ratio of the labor intensity of the recognized subtasks $(E_p^R\ )$ to the total labor intensity of the project $\left(E_p\right)$:

$$EA_p = \frac{E_p^R}{E_p} \cdot 100\%$$

where p is the project; EA_p is the index of effectiveness of classification of labor intensity of project tasks p; E_p^R is labor intensity of correctly identified project subtasks p; E_p is total labor intensity of the project p.

To further verify the accuracy of task classification, a manual check of a sample of 100 randomly selected tasks for each project was conducted. As a result, the accuracy rate was obtained (NA_p^M) determining the percentage of correctly classified tasks in the sample (N_p^{MC}) to the number of tasks in the sample (N_p^M):

$$NA_p^M = \frac{N_p^{MC}}{N_p^M} \cdot 100\%$$

where p is the project; NA_p^M is the indicator of the effectiveness of the classification of the number of tasks of the project p in the sample; N_p^{MC} is the number of correctly identified project tasks p in the sample; N_p^M is the number of project tasks p in the sample, $N_p^M = 100$.

The labor-intensity accuracy rate (EA_p^M) is obtained, determining the percentage of the labor intensity of correctly classified tasks in the sample (E_p^{MC}) to the total labor intensity of the sample (EA_p^M):

$$EA_p^M = \frac{E_p^{MC}}{E_p^M} \cdot 100\%$$

where p is the project; EA_p^M is the index of effectiveness of classification of labor intensity of project tasks p in the sample; E_p^{MC} is the labor intensity of correctly identified project tasks p in the sample; E_p^M is the total labor intensity of the project p in the sample.

Table 4 shows the results of the verification of the effectiveness of project task and subtask classification and the manual verification of the samples of project tasks.

Table 4. Results of verification of the effectiveness of the classification of tasks and subtasks of projects.

Project	EA	NA^M	EA^M
P1	91.50%	93.00%	93.72%
P2	61.50%	67.00%	31.14%
P3	99.70%	98.00%	99.85%
P4	25.00%	24.00%	52.88%
P5	99.60%	87.00%	100.00%
P6	54.70%	68.00%	99.50%

The accuracy rates of the P3 project can be considered exemplary, since a classification algorithm was developed for this project. The differences between the values of numerical and labor-intensive accuracy indicators determined during the manual verification of small samples (from 1% to 4% of the number of tasks in the project) are due to the fact that not all correctly classified tasks have labor hours recorded. The low indicator values for projects P2, P4, and P6 are due to the frequent use of a language other than English in task descriptions. The set of keywords developed for project P3 consists of English words, hence the poor transferability of the classification algorithm to some projects. In addition, the manual verification of the P4 project detected the following: a lack of keywords in the description and a large number of tasks with no change in status, resulting in a lack of subtasks and tasks with no registered work. The results in Table 4 indicate the need to translate job descriptions from the JIRA system into English before starting classification based on keywords or using NLP models.

6. Conclusions

The data of actual projects are the basis of the presented research and model. On the one hand, they increase the possibility of practical applications, on the other hand, they limit the model to the types of tasks and roles present in the researched projects. With this in mind, we tried to make the model flexible and open to modifications.

Connecting the model with the Jira system enables easy data acquisition for analysis and increases its commercial potential. A separate abstract layer of the model, in combination with a dedicated database, supports the possibility of creating interfaces for other IT project management systems.

The classification algorithms presented in the article are based on the manual recognition of task types. With manual recognition, rules based on keywords are created, which allows automatic recognition of task types at subsequent occurrences. As is known from the literature and practice, such algorithms are not very elastic and not very accurate (45% accuracy) [55]. However, with a growing base of manually recognized tasks, it will be easy to change to NLP models such as BERT [56]. This will allow fully automated operation of the task classification and subtask classification algorithm on a real-time basis. It will allow the analysis of the data collected in JIRA, the production of reports and charts to provide insight into the manufacturing process, and support for the project manager in decision making.

The division into state and cyclic tasks shows that state tasks are created during software development and their number and labor intensity depend on the size of the project. The rate of growth and completion of state tasks depends on the composition of the project team. The project's execution time depends on this rate. The number and labor intensity of cyclic tasks, on the other hand, depends on the duration of the project. Thus, the classification of tasks becomes the basis for constructing a generator of state and cyclic tasks to create software development plans. In turn, the creation of a development plan and the composition of the project team will allow the construction of a simplified simulation of the work in the project.

The ability to create a project plan and to select the appropriate composition of the project team, and, then, thanks to the simulation, to check how the work will proceed, will allow for comprehensive support of the management of the development process. Thanks to simulation, it will be possible to estimate whether the composition of the team is suitable for the specifics of the project. Simulation can show that, for example, programmers have implemented the requirements and that the team is waiting for testers to perform tests. In this way, the project manager can recognize the risk of a bottleneck in the project and prevent it in advance.

Project planning is useful not only before starting; real-time automatic task classification will allow analysis and will use the calculated task statistics to plan the next sprints or stages of the development process with increasing accuracy.

The introduction of an additional classification of tasks in terms of business and technological components, in conjunction with the employee competency model, would automatically collect information about employees' experiences in business and technology areas. This would allow the assessment of the level of employees' competences, selecting the composition of the project team and perhaps managing the team so that the competences are duplicated and dispersed among team members.

Author Contributions: Conceptualization, W.W. and I.M.; methodology, W.W.; software, W.W.; I.M. and M.M.; validation, W.W., I.M. and M.M.; formal analysis, W.W., I.M. and M.M.; investigation W.W. and I.M.; resources, W.W. and I.M.; data curation, W.W. and I.M.; writing—original draft preparation, W.W. and I.M.; writing—review and editing, W.W., I.M. and M.M.; visualization, W.W. and I.M.; supervision, W.W., I.M. and M.M.; project administration, W.W., I.M. and M.M.; funding acquisition, W.W. and I.M. All authors have read and agreed to the published version of the manuscript.

Funding: The project was financed within the framework of the program of the Minister of Science and Higher Education in Poland under the name "Regional Excellence Initiative" in the years 2019–2022, project number 001/RID/2018/19, the amount of financing PLN 10,684,000.00.

Data Availability Statement: Data are contained within the article.

Acknowledgments: Many thanks to Marcin Korzeń, Jakub Swacha, and Leon Dorozik for scientific support.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Wysocki, J. The use of information technologies in the enterprise. In *Science about the Enterprise: Selected Issues*; Lichniak I.: Warsaw, Poland, 2009; pp. 347–377. Available online: http://hdl.handle.net/20.500.12182/905 (accessed on 21 June 2022).
2. Zakrzewska, M.; Miciuła, I. Using e-government services and ensuring the protection of sensitive data in EU member countries. *Procedia Comput. Sci.* **2021**, *192*, 3457–3466. Available online: https://doi.org/10.1016/j.procs.2021.09.119 (accessed on 11 July 2022).
3. Açikgöz, A.; Günsel, A. Individual Creativity and Team Climate in Software Development Projects: The Mediating Role of Team Decision Processes. *Creat. Innov. Manag.* **2016**, *25*, 445–463. [CrossRef]
4. Mieśmieńska, L. Shaping the involvement of IT employees in the personnel strategy of company X. In *Man and Work in a Changing Organization. Towards Respecting the Interests of Employees*; Gableta, M., Pietroń-Pyszczek, A., Eds.; Scientific Papers No. 223; Wrocław University of Economics: Warsaw, Poland, 2011.
5. Olsen, T.L.; Tomlin, B. Opportunities and Challenges for Operations Management. *Manuf. Serv. Oper. Manag.* **2019**, *22*, 113–122. [CrossRef]
6. Azizyan, G.; Magarian, M.; Kajko-Mattsson, M. The Dilemma of Tool Selection for Agile Project Management. In Proceedings of the The Seventh International Conference on Software Engineering Advances ICSEA 2012, Lisbon, Portugal, 16–20 October 2022.
7. Powell, T.C.; Dent-Micallef, A. Information Technology as Competitive Advantage: The Role of Human, Business and Technology Resources. *Strateg. Manag. J.* **1997**, *18*, 375–405. [CrossRef]
8. Kędziora, A.F. SCRUM Methodology in Small and Medium IT Projects. 2011. Available online: http://min.wmi.amu.edu.pl/wpcotent/uploads/2011/04/PMKEDZIORASCRUM.pdf (accessed on 14 August 2022).
9. Elssamadisy, A. *Agile Patterns of Implementing Agile Practices*; Helion: Gliwice, Poland, 2010.
10. Shore, J.; Warden, S. *The Art of Agile Development*, 2nd ed.; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2008.
11. Bielec, J. The same technology—Different results. Key elements of the success of an IT project implementation. In Proceedings of the 13th PLOUG Kościelisko Conference, Post-Conference Materials, Kościelisko, Poland, 17–20 October 2007.
12. Spałek, S. *Critical Success Factors in Project Management*; Publishing House of the Silesian University of Technology: Gliwice, Poland, 2004.
13. Spolsky, J. *IT Project Management. Subjective View of a Programmer*; Helion Publishing House: Warsaw, Poland, 2005.
14. Wróbleski, P. *IT Project Management for Practitioners*; Helion Publishing House: Warsaw, Poland, 2005.
15. Abrahamsson, P.; Salo, O.; Ronkainen, J.; Warsta, J. *Agile Software Development Methods: Review and Analysis*; VTT Publications: Espoo, Finland, 2002; pp. 1–112.

16. Dima, A.M.; Maassen, M.A. From Waterfall to Agile software: Development models in the IT sector, 2006 to 2018. Impacts on company management. *J. Int. Stud.* **2018**, *11*, 315–326. [CrossRef] [PubMed]
17. Łabuda, W. Agile and traditional approach to software development projects. *Zesz. Nauk. WWSI* **2015**, *9*, 57–87.
18. Highsmith, J. *APM: Agile Project Management. How to Create Innovative Products*; Mikom: Warsaw, Poland, 2005.
19. Bhatt, P. An influence model for factors in outsourced software maintenance. *J. Softw. Maint. Evol. Res. Pract.* **2006**, *18*, 385–423. [CrossRef]
20. Kruchten, P. *The Rational Unified Process: An Introduction*; Addison-Wesley Professional: Boston, MA, USA, 2004.
21. Manifest Agile. 2001. Available online: https://agilemanifesto.org/iso/pl/principles.html (accessed on 17 June 2022).
22. *Agile Project Management Handbook*; Version 1.1; Dynamic Systems Development Method Limited: Katowice, Poland, 2013.
23. Andersen, E. Warning: Activity planning is hazardous to your project's health! *Int. J. Proj. Manag.* **1996**, *14*, 89–94. [CrossRef]
24. Bamel, U.K. Organizational climate and managerial effectiveness: An Indian perspective. *Int. J. Organ. Anal.* **2011**, *21*, 198–218. [CrossRef]
25. Beck, K. *Efficient Programming eXireme Programming*; MIKOM Publishing House: Warsaw, Poland, 2001.
26. Bieliński, J. *The Development of Sectors in the Modern Economy*; Publishing House of the University of Gdańsk: Gdańsk, Poland, 2006.
27. Jędrzejowicz, P. *IT Management Systems*; WSM in Gdynia: Gdynia, Poland, 2001.
28. Łabuda, W. How to implement a successful IT project. In Proceedings of the Conference Materials Summarizing the Project Program for the Development of the Teaching Offer and Improving the Competences of Lecturers at the Warsaw University of Information Technology, Warsaw, Poland, 20–21 September 2011.
29. Kisielnicki, J. *Management Infrastructure—Poland in Europe*; Master of Business Administration: Poznań, Poland, 2002.
30. Raunak, M.S.; Binkley, D. Agile and Other Trends in Software Engineering. In Proceedings of the IEEE 28th Annual Software Technology Conference (STC), Gaithersburg, MD, USA, 25–28 September 2017.
31. Turk, D.; France, R.; Rumpe, B. Limitations of Agile Software Processes. *arXiv* **2014**, arXiv:1409.6600.
32. Crispin, L.; Gregory, J. *Agile Testing: A Practical Guide for Testers and Agile Teams*; Addison-Wesley Professional: Boston, MA, USA, 2009.
33. Czerska, M.; Rutka, R. Assessment of the Designers' Motivation System in the IT Industry. In *Man and Work in a Changing Organization*; Gableta, M., Pietroń-Pyszczek, A., Eds.; Naukowe No. 43; University of Economics in Wrocław: Wrocław, Poland, 2009.
34. PMI. Foundation for Business Agility | Disciplined Agile. 2020. Available online: https://www.pmi.org/disciplined-agile,dostęp (accessed on 21 August 2022).
35. Wysocki, W. A hybrid software processes management support model. *Procedia Comput. Sci.* **2020**, *176*, 2312–2321. [CrossRef]
36. Miłosz, M.; Borys, M.; Plechawska-Wójcik, M. *Contemporary Information Technologies*; Agile methodologies of software development; Lublin University of Technology: Lublin, Poland, 2011.
37. Lievens, A.; Blažević, V. A service design perspective on the stakeholder engagement journey during B2B innovation: Challenges and future research agenda. *Ind. Mark. Manag.* **2021**, *95*, 128–141. [CrossRef]
38. Wojtaszek, H.; Miciuła, I. Analysis of Factors Giving the Opportunity for Implementation of Innovations on the Example of Manufacturing Enterprises in the Silesian Province. *Sustainability* **2019**, *11*, 5850. [CrossRef]
39. Krzos, G. Measures of the success of the project manager and projects co-financed by the EU. In *Selected Aspects of Managerial Work*; Cyfert, S., Ed.; Scientific Papers No. 187; Publishing House of the University of Economics in Poznań: Poznań, Poland, 2011.
40. Miciuła, I.; Wojtaszek, H. Automatic hazard identification information system (AHIIS) for decision support in inland waterway navigation. *Procedia Comput. Sci.* **2019**, *159*, 2313–2323. [CrossRef]
41. Orzechowski, R. Business-IT Alignment in Poland, *E-Mentor (E-Biznes)* Number 1 (23). February 2008. Available online: https://www.e-mentor.edu.pl/artykul/index/numer/23/id/520 (accessed on 27 October 2022).
42. Pawlak, M. *Project Management*; Polish Scientific Publishers PWN: Warsaw, Poland, 2006.
43. Wysocki, W.; Orlowski, C. A multi-agent model for planning hybrid software processes. *Procedia Comput. Sci.* **2019**, *159*, 1688–1697. [CrossRef]
44. Zaydi, M.; Nassereddine, B. DevSecOps practices for an agile and secure it service management. *J. Manag. Inf. Decis. Sci.* **2020**, *23*, 1–16.
45. Akbar, M.A.; Smolander, K.; Mahmood, S.; Alsanad, A. Toward successful DevSecOps in software development organizations: A decision-making framework. *Inf. Softw. Technol.* **2022**, *147*, 106894. [CrossRef]
46. Myrbakken, H.; Colomo-Palacios, R. DevSecOps: A multivocal literature review. In *Proceedings of the International Conference on Software Process Improvement and Capability Determination*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 17–29.
47. Wysocki, W. Agents of RUP Processes Model for IT Organizations Readiness to Agile Transformation Assessment. In *Intelligent Information and Database Systems*; Nguyen, N.T., Ed.; Springer International Publishing: Berlin/Heidelberg, Germany, 2017; pp. 777–786.
48. Piwowar-Sulej, K. Project management as a desirable managerial competence. In *Selected Aspects of Managerial Work*; Cyfert, S., Ed.; Scientific Papers No. 187, Wyd; Poznań University of Economics: Poznań, Poland, 2011.
49. West, D. *Water-Scrum-Fall Is the Reality of Agile for Most Organizations Today*; Forrester Research: Cambridge, MA, USA, 2011; Volume 26.
50. Attri, R.; Grover, S.; Dev, N.; Kumar, D. Analysis of barriers of total productive maintenance (TPM). *Int. J. Syst. Assur. Eng. Manag.* **2013**, *4*, 365–377. [CrossRef]
51. Akbar, M.A.; Sang, J.; Nasrullah; Khan, A.A.; Mahmood, S.; Qadri, S.F.; Hu, H.; Xiang, H. Success factors influencing requirements change management process in global software development. *J. Comput. Lang.* **2019**, *51*, 112–130. [CrossRef]
52. Poppendieck, M.; Poppendieck, T. *Lean Software Development: An Agile Toolkit*; Addison Wesley: Boston, MA, USA, 2013.

53. Zolnowski, A.; Anke, J.; Gudat, J. Towards a cost-benefit-analysis of data-driven business models. In Proceedings of the 13th International Conference on Wirtschaftsinformatik, St. Gallen, Switzerland, 12–15 February 2017.
54. Iriarte, C.; Bayona, S. IT projects success factors: A literature review. *Int. J. Inf. Syst. Proj. Manag.* **2020**, *8*, 49–78. [CrossRef]
55. Mccallum, A.; Kamal, N. *Text Classification by Bootstrapping with Keywords, EM and Shrinkage*; ACL: Stroudsburg, PA, USA, 2001.
56. Devlin, J.; Ming-Wei, C.; Kenton, L.; Toutanova, C. BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding. *arXiv* **2019**, arXiv:1810.04805. [CrossRef]