Chapter 4
Human Resource Allocation and Scheduling for Software Project Management

Constantinos Stylianou and Andreas S. Andreou

Abstract Software project management consists of a number of planning, organizing, staffing, directing and controlling activities. Human resources feature prominently in all of these activities and, as a consequence, they can affect and determine project management decisions. Therefore, in order to help guarantee the success of a software project, managers must take into consideration this type of resource when performing the aforementioned activities. This chapter specifically investigates human resources from a planning perspective and, in particular, focuses on the responsibilities of allocating developers and teams to project tasks, scheduling developers and teams, as well as forming development teams. These responsibilities are often challenging to undertake because they are accompanied by time, budget and quality constraints, which software project managers find difficult to balance correctly. The purpose of the chapter is to explore the most recent research work in the field of human resource allocation and scheduling, and to specifically examine the motivation behind each approach and the goals and benefits to real-world practitioners. In addition, the chapter investigates development team formation, which can be considered as an indirect method of allocating human resources to a software project. This perspective, in particular, sheds light on current and future trends, which lean towards incorporating human-centric aspects of software development in planning activities.

C. Stylianou
Department of Computer Science, University of Cyprus, Lefkosia, Cyprus
e-mail: cstylianou@cs.ucy.ac.cy

A.S. Andreou (✉)
Department of Electrical Engineering/Computer Engineering and Informatics, Cyprus University of Technology, 31 Archbishop Kyprianou Avenue, P.O. Box 50329, Lemesos 3036, Cyprus
e-mail: andreas.andreou@cut.ac.cy

4.1 Introduction

Human resource allocation involves assigning a developer to carry out a task and attempts to answer “Who will work on what?”, whereas human resource scheduling involves specifying the time frame in which a developer will work on a task and tries to answer “Who will work when?”. They are both part of a software project manager’s planning activities, but depending on the practices followed by each software development company and the information regarding the actual software project, the way they are carried out can vary. In some cases, a software project manager is required to allocate and schedule one or more developers to each task, whereas in other cases, tasks are distributed to already predefined teams of developers. Finally, there are times when a project manager needs to only put together a group of developers without assigning them to specific tasks or scheduling them, in effect carrying out team formation. Generally, they are both carried out at the initial phases of a software project. However, in most cases, information at the beginning of a software project is often imprecise or unavailable. As a result, they are considered two of the most challenging responsibilities of software project managers, and they are part of the first activities that can significantly affect the progression and overall success of a software development project. Adding further complexity for project managers is the fact that allocating developers to tasks and scheduling tasks and developers are not independent activities and that treating them so may be considered unsuitable (Chang et al. 2008). Allocation and scheduling both affect the availability of developers, so in order to avoid conflict, both activities need to be worked on simultaneously.

To help them, software project managers make use of their own past experiences and previously acquired knowledge together with the wide range of available commercial tools and techniques, such as Microsoft Project, Project KickStart, Basecamp, MatchWare MindView and RationalPlan MultiProject. A study of the impact of project management information systems by Raymond and Bergeron (2008) found that such systems improve efficiency and effectiveness with respect to project planning and control activities, as well as general project performance and overall success. However, not many of these available applications are tailor-made for the software development industry. A survey conducted by McBride (2008) highlights this, especially for monitoring, controlling and coordination activities, in which project managers use a number of different mechanisms for a given activity but also use the same mechanism for a number of activities.

As a result of a lack of specific tools, software project managers still seek the aid of more practical and intelligent models and techniques to overcome the challenges posed within these activities. Many research studies have contributed to the attempt to solve the problem of human resource allocation and scheduling specifically for the software industry, in particular, by utilizing specialized operational research techniques found in the fields of mathematical modelling and computational intelligence. The first part of the literature review in this chapter (Sect. 4.2), therefore, is dedicated to providing an overview of some of the most recent approaches proposed that specifically focus on the use of these techniques in order to handle the most common human resource allocation and scheduling issues and to provide an automated way of supporting software project managers with practical benefits to the software industry.

Given that even to this day a large percentage of software projects are severely challenged or considered to have failed, combined with the fact that human resources are considered the only type of resource in a software project, one of the directions that the software engineering research community is trying to establish for the software development industry to follow involves including human-centric aspects. Specifically, this direction attempts to promote non-technical aspects of software development as equally important as technical aspects in approaches for human resource allocation, scheduling and team formation.

A leading trend focuses on taking into account the personality types of developers when assigning them to tasks and also when grouping them into development teams. More and more studies are now being performed aiming to observe the effects of personality types on performance, productivity, software quality and job satisfaction. Also, there have been attempts to determine the personality types required of different software development professionals, such as system analysts, programmers, testers, etc. Therefore, the second part of the literature review in this chapter (Sect. 4.3) presents various human resource allocation and scheduling approaches, as well as team formation strategies for software development teams, that incorporate personality types and results of their application in the software industry.

The aim of this chapter is to present a better understanding of research efforts for human resource allocation, scheduling and team formation in the software industry and to highlight the trends emerging. By conducting this survey, the aim is to identify any gaps in current research of software project human resource allocation in order to suggest possible areas for further investigation. To facilitate this, an initial search of digital libraries was carried out to extract primary studies and survey papers concerning software human resource allocation scheduling and team formation. The electronic databases used included IEEE Xplore, ScienceDirect, ACM Digital Library, SpringerLink, Google Scholar and Wiley Interscience. Other sources were later incorporated by using the reference lists of retrieved studies, as well as conference proceedings and technical reports. Irrelevant or duplicate articles and papers that the initial search generated were then eliminated. Due to the different terminology often used to describe human resource allocation and scheduling, a number of keyword search strings were used to obtain related articles. Specifically, conjunctions of different phrases were built by selecting from a list of various related keywords, including terms such as (1) “resource,” “human resource,” “developer,” “software developer,” “team,” “software team”; (2) “scheduling,” “planning,” “allocation,” “formation,” “assignment”; and (3) “software,” “software project,” “software development,” “software management.”

4.2 Human Resource Allocation and Scheduling Approaches

The various models and techniques adopted in the proposed attempts belong to the fields of mathematical modelling and computational intelligence as these generally contain the most commonly used methods to solve problems in the field of operational research. Operational research consists of different complex decision-making problems in various fields, such as natural sciences and engineering as well as social sciences, which are solved by locating optimal or near-optimal solutions. Human resource allocation and scheduling can be considered an operational research problem and, thus, most research efforts are focused on providing solutions using mathematical modelling and computational intelligence methods. A short description of each approach is provided so that the reader can identify the possible benefits and shortcomings of using it for allocation and/or scheduling in the real world.

Mathematical models make use of mathematical notations and concepts, such as variables, operators, equations and functions, to represent a problem and then attempt to solve the model as an optimization problem or to use the model as a prediction scheme. The types of models include linear programming, which is optimizing a linear objective function, statistical modelling and queuing theory. On the other hand, computational intelligence techniques comprise a range of nature-inspired methodologies and algorithms aiming to solve real-world problems that are both complicated and complex. Their goal is to imitate the individual and collective behaviours and qualities of living beings concerning reasoning, logic and inference, learning and processing knowledge, as well as reproduction and evolution to achieve specific goals. The most well-known techniques include evolutionary algorithms and swarm intelligence.

In general, the available methods and techniques in both categories have been adopted as means to help solve several important problems in the field of software engineering. For example, techniques that are capable of carrying out prediction have been used in models for estimating software costs and effort (Heiat 2002), classification schemes have been utilized for evaluating software quality (Khoshgoftaar and Seliya 2004), clustering algorithms have been part of attempts to group and retrieve software components in repositories (Stylianou and Andreou 2007), and methods for optimization have been applied to automatically generate test-cases (Michael et al. 1997).

4.2.1 Mathematical Modelling Approach

4.2.1.1 Linear Programming

The first type of mathematical modelling concerns linear programming, which requires a linear objective function to be minimized or maximized in order to find optimal solutions to problems described by linear relationships subject to certain problem-specific restrictions. Kantorovich (1940), a Soviet mathematician during World War II, first introduced this approach as a means to solve several planning problems for the military, including how to optimally assign, schedule and transport resources based on their availability and cost so that army expenses are reduced while enemy losses are increased. Consequently, linear programming has been considered by researchers as a suitable technique for helping software project managers in their planning activities also.

Li et al. (2007) used integer linear programming to help software development organizations cope with the pressures of limited resources and decreased time-to-market intervals by proposing two models concerning requirement scheduling and software release planning. Their first model takes into account the precedence dependencies of requirements and the skills of available teams of developers to generate a project schedule for the development of requirements of a new release within the shortest possible make-span, whereas their second model integrates requirements selection and software release planning of a project with a fixed deadline to maximize revenues in addition to providing an on-time delivery schedule. One of the assumptions of this attempt is that requirements are assigned to teams of developers to implement and not to individual developers. Additionally, for testing their proposed approach, the authors used both example and real-world data sets. The authors do point out, however, that a mathematical model cannot stand alone as a project management decision support system since other real-world factors influence the decision-making process, such as psychological, personality and political factors.

Another methodology using linear programming is presented by Otero et al. (2009) and was developed to tackle the issue of project manager subjectivity in human resource allocation. The authors highlight that ineffectual resource allocation can lead to many problems for development organizations, such as "*schedule overruns, decreased customer satisfaction, decreased employee morale, reduced product quality, and negative market reputation*" (Ejnioui et al. 2012). They, therefore, propose the best-fitted resource methodology that works to measure the suitability between the skills required by tasks and the skills possessed by the available resources. Project managers can then use the results from the methodology to decide on the most suitable (optimal) allocation of resources based on their capabilities. To test their approach, the authors provided a small sample resource allocation scenario to 30 subjects consisting of software engineers and project managers from the industry and also computer science students and professors from universities, and asked them to perform a ranking of the available resources based on the developers capabilities in the required skills. The results of this survey were then compared to the results obtained from their methodology, showing that such an approach had potential in allocating developers to tasks.

Otero et al. (2010) presented another similar multicriteria decision-making methodology for software task assignment. Here, they state that there is evidence that ineffective human resource project planning is the main reason that software development projects fail (Tsai et al. 2003). The methodology uses a desirability function as a means of assigning tasks to developers in cases where there are no optimally suitable developers in the existing workforce. It takes into account the capabilities of resources in skills, the required levels of expertise as well as the level of significance of skills required by tasks and task complexities. A significant aspect of this approach is that it can be extended to take into account project-specific factors that a software project manager decides are important according to the needs of the project. An artificial case study was used to demonstrate the methodology, consisting of a scenario where a task needed to be assigned to one of ten candidate developers based on their skill assessment and associated cost with respect to the required skills of the task. On a practical level, the authors state that the approach can be adopted by software project managers using a simple spreadsheet implementation. However, no formal description of a tool is provided. Although it seems sensible to exploit the strengths of developers based on what each task requires, this is only realistically possible if the developer is available to carry out a task. The approach, however, does not address the issue of availability when computing the desirability function and does not deal with human resource scheduling, which often influences or comes hand-in-hand with allocation.

4.2.1.2 Probabilistic Modelling

Probabilistic modelling is a mathematical modelling approach that uses data (usually historical data) to forecast the conditions of different future states of a problem by calculating the probability of certain outcomes. A characteristic of this approach is that one or more of the variables in the model can be random.

Padberg (2001) presented a probabilistic project scheduling model, which focused on using scheduling strategies to help software development organizations to manage their human resources more effectively, arguing that software developers are the most valuable resources and that software project managers need a useful scheduling support tool as opposed to a common cost estimation tool that simply predicts the overall development effort needed to carry out a project. Specifically, in the approach scheduling strategies represent, in quantitative terms, the effect of decisions regarding development costs and duration on the current state of a project. Once a strategy is fixed it is inserted into the model to compute a probability distribution estimating the completion time and cost by using several technical and non-technical factors, such as scheduling constraints, adopted software processes and the complexity of components to be developed, as well as the skills and experience of the human resources. Stochastic optimization techniques are then applied to optimize the expected duration or the cost of the project with regard to the allocated resources. It is important to model the intrinsic uncertainty that is part of the software process regarding the duration of activities and also the events that occur during a project. The author, therefore, claims that using a probabilistic approach can help deal with the fact that events in a project can occur with a particular likelihood. The approach considers a project to be broken down into components to which only one team is assigned at any given time. An advantage to the approach is that it allows a team to interrupt their work on a component in order to rework a previously completed component. In addition, it takes into account the availability of development teams as well as the precedence relationships between components. However, overall this approach can only be applicable in software companies that have predefined teams of developers, with each team possessing the know-how to undertake the development of the component. For small-to-medium sized companies that do not often have such luxury, this could be impractical.

Padberg (2002, 2003) later implemented this previous probabilistic scheduling model as a discrete simulation model for project managers to use as a tool to provide feedback and comparisons among varying strategies and also implemented a variation of the value iteration algorithm to generate optimal scheduling policies in the model (Padberg 2004, 2006). The premise of these works remains the same as in his previous approaches: that uncertainty inherent in the task durations can only allow a software project manager to create a schedule wherein the duration and cost are "*likely*" to be minimized, and so it is vital for software project managers to be able to apply dynamic scheduling policies.

4.2.1.3 Queuing Theory

Queuing theory can be used as a mathematical model to simulate a system providing services to customers (human or otherwise) as they wait in line to be served. In general, this method attempts to minimize the duration and size of delays subject to constraints and, therefore, has practical applications in problems such as scheduling, employee allocation, facility design and management, and traffic flow management.

Antoniol et al. (2004a) used this technique in their approach concerning the allocation of resources in a large software maintenance project. Specifically, the authors made use of stochastic simulations of queuing networks as an instrument to evaluate the probability that the project meets its deadline as the project is being carried out.

Jalote and Jain (2004) implement a critical path/most immediate successor first approach to resource allocation targeting software projects that are to be developed by multiple teams across different geographically distributed time zones. With a rise in the number of organizations adopting global software development, project managers face new communication and coordination issues in addition to technical and managerial problems. Therefore, they suggest a 24-hour software factory model that utilizes project task precedence graphs and available resources to satisfy three types of constraints: operational, skill and resource, in order to generate a near-to-optimal software project schedule with the shortest make-span. Further reading on global software development is available in Chaps. 9 and 10, which discuss in detail various aspects of managing IT projects developing software across the globe and motivating virtual team members involved in global IT projects, respectively.

4.2.1.4 Constraint Satisfaction

Constraint satisfaction is a method that is adopted as a means of modelling and finding solutions to combinatorial problems by imposing conditions on variables in mathematical functions that are all required to be satisfied. They feature in many artificial intelligence fields and other disciplines, including planning, scheduling and logistics. Well-known examples of problems that can be solved using this method include map colouring, job shop scheduling and even Sudoku puzzles. With respect to the software industry, the constraints regarding development projects predominantly concern the budget, the schedule and the quality of the software products. Therefore, this method is adopted in order to attempt to satisfy the restrictions surrounding these issues.

Barreto et al. (2005) proposed the use of constraint satisfaction as an optimization approach to software project staffing, stating that process productivity and product quality are highly associated with the abilities of the available resources. The abilities taken into consideration included skills, knowledge, experience, capabilities and roles, and together with the characteristics of a project's activities and any development organization constraints, various utility functions can be maximized or minimized depending on the project manager's needs. The possible optimizers implemented consisted of most or least qualified team, cheapest team, smallest team, and best partial solution team. It is assumed that tasks are broken down into small units of work to which only one developer can be assigned. Once the software project manager decides what these tasks are, the tool performs optimization in order to locate the developer assignments that best fit the chosen utility function. The approach concentrates solely on the allocation of resources, while the starting and finishing times of tasks are known beforehand.

As an extension to their previous approach, Barreto et al. (2008) incorporated a mechanism to also handle developer productivity. The authors state that the time taken to carry out a task is affected by the developer's level of productivity. Hence, the approach proposes various productivity modifiers computed based on the experience, the profession or the activity itself. A software project manager selects to apply one of these modifiers, and then a new duration for each task is estimated accordingly (either increasing or decreasing it based on the developer assigned). A new utility function was subsequently implemented to enable assignments yielding the fastest team. The ability to factor in productivity is very important for software companies as the accuracy of budgets and schedule estimates can be improved.

4.2.2 Computational Intelligence Approaches

4.2.2.1 Evolutionary Algorithms

Evolutionary algorithms are a class of population-based algorithms that stem from the theory of natural evolution. They are most widely used to solve search-based problems that require some form of optimization since they are able to explore and exploit a problem's search space more efficiently and effectively to locate the best or near-best solutions. Consequently, evolutionary algorithms have been commonly applied directly or indirectly to the problem of human resource allocation and scheduling in software development projects. In order to find the optimal or near-optimal solutions, evolutionary approaches evaluate each individual in the population, which represents a candidate solution, using an objective function that rates the fitness of solutions and checks whether they satisfy various constraints. Stronger candidates are passed into subsequent generations, whereas weaker ones are discarded, leading to the detection of (near-)optimal solutions. Chapter 15 discusses a number of approaches that adopt such algorithms, though several have been selected to be presented in the remainder of this section.

One of the earliest instances of using evolutionary algorithms for software project allocation and scheduling is found in the work of Chang et al. (1994), who formalized a model for software project management, namely SPMNet, in the mid-1990s. Their approach focuses on the fact that software development organizations fail to assign the right developers to the right tasks due to the difficulties faced by project managers in handling the high level of complexity involved in finding optimal or near-optimal schedules. Their approach employs a single-objective genetic algorithm as a "*schedule optimizer*" aiming to minimize the total duration and cost of a software project through a process of assigning software developers to tasks (Chang et al. 1994, 1998). One of the practical benefits of the formal software management model proposed is that it allows software project managers to track the progress of a project by working together with developers and customers. It also addresses the issue of risk management by enabling the pre-execution of SPMNet and, hence, predicting the future states of a project. Over the years, this model has been significantly extended to support features to deal with additional software project management issues, such as partial assignment of developers to tasks, developer overload and multiple project scheduling (Chang et al. 2001), in addition to developer reassignments, task suspensions and resumptions, learning and task-specific deadlines (Chang et al. 2008).

Ge and Chang (2006) used the schedule optimizer mentioned above to implement a capability-based scheduling framework in which task durations are calculated through system dynamics simulation that focused on the capabilities of the available personnel. The authors state that it is important to consider developers' capabilities because they can influence a team's average productivity, which is determined by factors such as individual productivity, overworking and communication overhead. Being able to simulate the effect of an assignment based on the capabilities of the developer going to carry out a task could provide software project managers vital information at any stage during the project. However, exact details on how system dynamics simulation manages to generate task durations are not provided, which severely limits the assessment of its applicability in real-world settings.

An extension to the capability-based scheduling framework is suggested by Jiang et al. (2007), who incorporate personnel risks based on historical data during the assignment process. This addition is aimed at helping software project managers identify, analyse and monitor possible risk factors arising from human activities (for example, late-in-the-day coding) and allow them to regulate resource assignment. Furthermore, the authors adapt the previous genetic algorithm to a multi-objective schedule optimizer employing a weighted sum method to allow for tradeoff solutions to be generated. Another approach using multi-objective optimization was implemented again by Ge (2009) to provide a framework for scheduling and rescheduling software projects. The approach takes into account the skills and capabilities of available developers and attempts to provide an optimal project schedule based on efficiency (minimum cost and duration) and also stability factors (minimum impact of disruptions caused by rescheduling developers).

Alba and Chicano (2005, 2007) also employed genetic algorithms to develop an automated tool to allocate resources to tasks taking into account duration, resource skills, cost and global complexity. Their research work was centered on the fact that one of the goals of software project managers is to reduce both the cost and the duration of software development projects even though these two goals can be conflicting. Each individual in the population is an assignment matrix representing the allocation of developers to tasks. The quality of each assignment matrix regarding cost and duration is evaluated through two objectives using the salary of each developer, the degree of dedication each developer is permitted to work on each task and the effort required for each task. The project's schedule is constructed directly as a result of the developers allocated to each task. As the algorithm executes, solutions converge to the optimal/near-optimal allocations and schedules. By allowing project managers to adjust weights according to the problem at hand, they have the ability to perform different scenario analyses and make better decisions regarding the software project. This is a significant feature because the importance of each criterion is subject to the software being developed within the project, thus it is reasonable to expect a software project manager in some cases to want to give emphasis on minimizing the cost of the project and in other cases to want to focus on minimizing the project's duration, depending on which criterion he or she considers more important. One drawback to the approach, however, relates to the way that developer skills are handled. Specifically, the skills possessed by developers are treated as Boolean; either a developer possesses a certain skill or does not, and this information is used to evaluate whether the skills required by the project's tasks are satisfied in the form of a constraint. However, in reality, most project managers do not treat skills in such a way but rather take into account that developers possess skills at varying levels. Therefore, the approach would make more sense to address this as part of the evaluation of objectives (i.e., as an additional criterion for assigning developers to a task) rather than as part of the assessment of the constraints. A comparison of several multi-objective evolutionary algorithms using various quality indicators was subsequently performed in Luna et al. (2011) and Chicano et al. (2011) using the same representation, that is, with each solution comprising a series of developers possessing a set of skills, which are matched against the skills required by the project's tasks. None of the experiments in this group of approaches, however, has been tested on real-world software projects. Instead, they have only been applied to a collection of simulated projects, which were created by an instance generator that randomly creates a set of tasks (with associated costs and required skills) and a set of developers (with associated salaries and skills possessed). The randomness of the generated software projects may not always accurately reflect, for example, the correlation between the skill set and salary of a developer where higher-skilled employees are more likely to be paid more.

In the approach proposed by Duggan et al. (2004), project managers supply the complexity of the packages to be developed (using McCabe's (1976) cyclomatic complexity measure) and the proficiency (from novice to expert) of the available software engineers in each of the packages. Using a multi-objective genetic algorithm, the approach aims to find an optimal solution that minimizes the number of defects per unit of complexity and minimizes the duration of the project with a specific assignment of developers. However, software project managers may find it difficult to adopt this approach because it is strictly focused on allocating and scheduling resources regarding implementation tasks of a development project and only if the project is developed using an object-oriented approach.

Kapur et al. (2008) proposed a hybrid approach, which employs integer linear programming in conjunction with genetic algorithms for resource scheduling and allocation, targeting planning product releases. The authors emphasize the fact that software developers have different levels of skills and so their goal is to help project managers assign the most qualified developers to the required tasks in order for them to achieve maximum productivity, which in turn leads to a product release offering features that maximize business value. The optimization carried out using the genetic algorithm helps software companies decide which features should be included in a particular release for its customers. The two-phase method was applied to a real-world project carried out at Chartwell Technology, which specializes in developing online gaming and gambling software, demonstrating how change requests, user requirements and improvements were planned and ordered. This approach, however, can only be used for allocating and scheduling human resources for software projects developed incrementally. Ngo-The and Ruhe (2009) further develop this two-phase approach again aimed at incremental software development. The authors use integer linear programming to fix an upper bound to the maximum possible achievable business value according to stakeholders' satisfaction and then employ a genetic algorithm to evaluate this value and subsequently find an optimal or near-optimal assignment and schedule of developers to tasks in order to plan which features are to be included in each release and which are to be postponed. The approach also allocates non-human resources, such as capital, during the assignment procedure. One of the benefits of the approach, as stated by the authors, is that project managers can replan features and reschedule resources if requirements are changed or new requirements are introduced by simply using the same two-phase approach with modified inputs and parameters.

Several attempts carried out by Antoniol et al. (2004b, 2005) had the sequence of execution of work packages and the assignment of teams to work packages evaluated using a hybrid of queuing simulation and a single-objective genetic algorithm. A shift to a multi-objective genetic algorithm was then made in the approach suggested in Gueorguiev et al. (2009), which highlights the difficulties in constructing project schedules with regard to risk. The main objective of this approach focuses on the conflicting objectives of robustness and completion time, but the approach can be used implicitly for resource usage maximization. Furthermore, the adoption of queuing simulation for task staffing and optimization for scheduling tasks are also part of a later approach in Di Penta et al. (2011), where additional features are implemented to deal with fragmentation, software developer specialization and work package dependencies. Ren et al. (2011) opted for a different approach to optimizing the sequence of execution of work packages and assigning developers to tasks by adopting a cooperative co-evolutionary method, which tries to evolve two populations of individuals simultaneously through collaboration, rather than having individuals in a single population compete against each other.

In an alternative approach, Yannibelli and Amandi (2011) proposed a knowledge-based genetic algorithm to aid project managers at the early stages of scheduling to staff software projects with the most effective employees. Specifically, the approach uses available knowledge about employees' previous participation in projects to evaluate how effective a set of resources will be if assigned to a specific activity and how effective each individual in that set will be. With this knowledge, the algorithm attempts to find feasible and optimal project schedules satisfying the precedence relationships between the activities and the human resource requirements. An important aspect of this approach is that allocations are based not only on the skills of developers but also on the level of effectivity that is the result of two or more developers working together on the same task. This is an attempt to reflect real-world practices since a software project manager may be hesitant to assign a task to a pair of developers when he or she is aware that the pair is less effective working together, even though individually the developers possess a higher level of skills than another pair of developers. It might be preferable to allocate two developers who are less skilled, but more effective working together in order to be more productive. What the authors do not make clear, however, is whether the duration of a task is specified knowing the exact number of developers to be assigned to it. What would be more flexible, if this is not the case, is having the duration of a task to actually shorten or stretch depending on the final level of effectivity resulting from the developers assigned.

Some of the approaches mentioned in this section are revisited in Chap. 15, which provides an overview of how different areas and problems of software project management have been reformulated to be solved with computational search and optimization techniques.

4.2.2.2 Swarm Intelligence

Swarm intelligence algorithms are a specific group of computational intelligence methods inspired by the behaviour of biological systems found in nature, such as the flocking of birds and the schooling of fish. The aim of swarm intelligence algorithms is to mimic how each individual in the swarm acts and interacts with other individuals in its environment to achieve a common goal shared by all individuals. In particular, swarm intelligence algorithms, such as ant colony optimization and particle swarm optimization, work similarly to evolutionary algorithms by assessing the quality of the solution that each individual in the swarm represents. In the case of human resource allocation and scheduling for software development, these types of algorithms are only just now beginning to be applied, though the general goals of the approaches still focus on minimizing the cost and duration of software projects in a similar fashion to evolutionary algorithms.

Chen and Zhang (2013) recently proposed a model that combines an event-based scheduler with ant colony optimization, aiming to provide solutions consisting of reduced project costs and more stable workload assignments. Essentially, the model considers that developer allocations are affected by specific events: the starting time of the project, the time when developers join or leave the project, and the time when developers are released from completed tasks. When one of these events occurs during the project, the event-based scheduler modifies the allocation of developers based on the priority given to tasks, the skill proficiency of developers and the current availability of the developers. Then, the method goes on to construct a new schedule by using ant colony optimization, where artificial ants are iteratively dispatched to build project plans. The practical benefits with this method are that it allows a software project manager to have the flexibility to pre-empt tasks, but also to be able to handle and avoid resource conflicts. Experiments were carried out on 80 artificial projects and 3 real-world business software projects of a departmental store, and the results demonstrated that the combination of event-based scheduling with ant colony optimization was effective in yielding solutions with the lowest project cost.

Xiao et al. (2013) also presented an approach using swarm optimization to allocate and schedule developers in a software project. The authors adopted a similar approach as Alba and Chicano (2005), but instead of using a genetic algorithm to generate solutions with optimal developer assignments and project schedules, they adopted ant colony optimization. They used the same objectives, that is, to minimize cost and duration, subject to the precedence relationships of tasks and skills of developers. The authors show through the results of optimization on 30 randomly generated project instances that this approach outperformed the original.

A particle swarm optimization algorithm was used by Gerasimou et al. (2012) in an initial investigation in maximizing human resource usage. The proposed approach aims to assign tasks to software developers based on their experience in the skills required by the tasks and, simultaneously, to generate the shortest project make-span by scheduling tasks with respect to task dependencies and developer availability. A software project manager gives each objective a weight denoting its level of importance so that the particles, which represent the start days of tasks and the developers allocated to it, can converge to an optimal as possible solution that balances the two objectives according to the weights provided. The reasoning behind this approach is that, ideally, a software project manager would want to assign a task to the developer most suitable in terms of skill. However, if the most skilful developer is assigned conflictingly to two tasks that are scheduled to execute in parallel, a software project manager is faced with the dilemma of how to handle such a conflict. Does he or she allocate a different, possibly less skilled, developer to one of the tasks but keeps the schedule the same? Or does he or she leave the assignments as they are and sets one of the tasks to start as soon as the developer becomes available again, possibly increasing the duration of the project? The challenge with implementing such an approach in real-world projects is that the cost of the project is not taken into consideration, which could also affect the allocation criteria. Furthermore, project managers may not be able to quantify the experience a developer has in particular skills easily. It is, however, critical to take into account the non-interchangeable nature of software developers considering that each developer possesses a different skill set with different levels of proficiency.

4.2.2.3 Fuzzy Logic

Fuzzy logic is regarded as a control system for solving problems based on information that is imprecise, ambiguous, uncertain or even missing, and is used to imitate the human decision-making process on a linguistic (descriptive) rather than a numeric basis. The goal is to model the vagueness of variables that do not possess a clear and crisp distinction between its possible values. Instead, it divides the variable into (usually) overlapping fuzzy sets and with the use of membership functions determines the degree to which a specific value falls into each set. It has been applied in many disciplines, such as robotics, medicine and management, where it has helped overcome the subjectivity of the decision maker.

One attempt at using fuzzy logic for software project scheduling was proposed as a decision support system by Hapke et al. (1994), who claim that, due to the uncertainty of time parameters, software project managers can only approximate the durations of development activities. The fuzzy project scheduling system proposed, therefore, creates intervals representing possible durations of tasks and aims to assign software engineers to development phases taking into consideration the completion time and maximum lateness of a software project. The time criterion is cut into lower and upper bounds generating a set of optimistic and pessimistic scenarios, which are then optimized using priority heuristic rules. Because the approach only handles the minimization of the duration of projects, its applicability in the industry may be limited. The fact, however, that human resources are considered renewable resources severely increases its limitations since it does not accurately reflect the impact that developers' capabilities can have on allocation and scheduling.

Fuzzy logic was also employed as a means for project scheduling by Callegari and Bastos (2009) in order to handle the difficulties present in pure mathematical models, for example, "*the partial loss in meaning in terms of knowledge representation.*" The multi-criteria resource selection method proposed employs multi-valued logic and a set of inference rules to rank available resources according to their suitability to specific tasks, thus allowing project managers to assign resources to tasks. Specifically, a fuzzy rule matrix is constructed that stores how suitable an assignment is based on the skill level expected by a task and the current skill level possessed by the assigned developers. If-then rules then help software project managers allocate developers in order to meet the requirements of each task. One advantage of this approach is that the rules can help avoid poor utilization of developers, which is considerably important for software development companies as highly experienced developers are not wasted on tasks requiring low levels of skills. However, one criticism is its inability to handle the scheduling of developers simultaneously. This is one of the few approaches that also demonstrate a prototype tool to show how a software project manager can adopt the approach in the industry.

The approaches discussed in Sect. 4.2 are summarized in Table 4.1. They are grouped by the method/technique adopted in each proposed human resource scheduling and allocation attempt explored. As can be seen, the majority of attempts employ computational intelligence methods as a form of optimization, with the most popular technique being evolutionary algorithms.

4.2.3 Discussion

Not getting the right people to do the right job at the right time can be detrimental to the success of a software project. Various techniques borrowed from several fields have been used to help avoid this through different approaches allocating and scheduling human resources in software projects. But despite the evolution over the years, the problem still remains unsolved largely because there is no consensus on the criteria that these research approaches need to target to create a successful human resource allocation and scheduling tool. There are several notable points regarding the approaches that need to be addressed.

Firstly, even though there have been many approaches proposed, their ability to be applied in real-world environments is not always clear. First and foremost, any approach should be accompanied with some sort of tool to show exactly how the approach could be adopted by software project managers and not provide only a description of the underlying mechanisms. Additionally, the information needed to

Table 4.1 Summary of human resource allocation and staffing approaches

Technique/Method	Research Attempt	Goals/Objectives	Constraints	Data Used
Linear Programming	– Li et al. (2007)	– Minimize project duration – Maximize revenues	– Requirement dependencies – Team availability	– Simulated – Real-world
	– Otero et al. (2009) – Otero et al. (2010)	– Maximize suitability of developers	– Skill/expertise requirements – Resource requirements	– Simulated
Probabilistic Modelling	– Padberg (2001; 2002; 2003; 2004; 2006)	– Minimize project duration	– Skill/expertise requirements	– Simulated
Queuing Theory	– Antoniol, Cimitile et al. (2004)	– Minimize risk of delay	– N/A	– Real-world
	– Jalote and Jain (2004)	– Minimize project duration	– Task dependencies – Skill/expertise requirements – Resource requirements	– Simulated – Real-world
Constraint Satisfaction	– Barreto et al. (2005; 2008)	– Minimize project cost – Maximize/minimize team quality – Minimize team size – Minimize project duration	– Resource requirements – Developer availability	– Simulated
Evolutionary Algorithms	– Chang et al. (1994) – Chang et al. (1998) – Chang et al. (2001) – Chang et al. (2008)	– Minimize project cost – Minimize project duration – Minimize amount of over-time	– Developer availability – Developer overtime limit – Hard deadlines – Resource requirements	– Simulated
	– Ge and Chang (2006)	– Minimize project cost	– Developer availability – Developer overtime limit – Task dependencies	– Simulated

Technique/Method	Research Attempt	Goals/Objectives	Constraints	Data Used
Evolutionary Algorithms (continued)	– Jiang et al. (2007)	– Minimize project cost – Minimize project risk	– Developer availability – Developer overtime limit – Task dependencies	– N/A
	– Ge (2009)	– Maximize efficiency – Maximize stability	– Developer availability – Developer overtime limit – Task dependencies	– Simulated
	– Alba and Chicano (2005; 2007) – Luna et al. (2011) – Chicano et al. (2011)	– Minimize project cost – Minimize project duration	– Developer overtime limit – Task dependencies – Resource requirements – Skill/expertise requirements	– Simulated
	– Duggan et al. (2004)	– Maximize project duration – Minimize software defects	– Package dependencies – Team utilization – Communication overhead	– Simulated
	– Kapur et al. (2008) – Ngo-The and Ruhe (2009)	– Maximize business value	– Feature dependencies – Task dependencies – Developer availability – Release deadlines – Feature release requirements – Resource requirements	– Simulated – Real-world
	– Antoniol, Di Penta et al. (2004; 2005)	– Minimize project duration	– N/A	– Real-world
	– Gueorguiev et al. (2009)	– Minimize project duration – Minimize project overruns	– Work package dependencies	– Real-world

(continued)

Table 4.1 (continued)

Technique/Method	Research Attempt	Goals/Objectives	Constraints	Data Used
Evolutionary Algorithms (continued)	– Di Penta et al. (2011)	– Minimize project duration – Minimize schedule fragmentation	– Work package dependencies – Work package assignment – Skill/expertise requirements	– Real-world
	– Ren et al. (2011)	– Minimize project duration	– Work package dependencies – Resource requirements	– Real-world
	– Yannibelli and Amandi (2011)	– Maximize effectivity levels of teams	– Task dependencies – Resource requirements	– Simulated
Swarm Intelligence	– Chen and Zhang (2013)	– Minimize project cost	– Task dependencies – Developer overtime limit – Resource requirements	– Simulated
	– Xiao et al. (2013)	– Minimize project cost – Minimize project duration	– Developer overtime limit – Task dependencies – Resource requirements – Skill/expertise requirements	– Simulated
	– Gerasimou et al. (2012)	– Minimize project duration – Maximize suitability of developers	– Developer availability – Task dependencies – Resource requirements – Skill/expertise requirements	– Simulated
Fuzzy Logic	– Hapke et al. (1994)	– Minimize project duration	– Task dependencies – Developer availability	– Real-world
	– Callegari and Bastos (2009)	– Maximize suitability of developers	– N/A	– Simulated

execute any approach should be easily obtainable and measurable where necessary by software project managers, such as the dependency relationships between tasks in order to validate the feasibility of a schedule. But, for example, things like units of complexity may not be able to be provided by a software project manager, especially at the initial stages of the project. Also, some attempts have put their approach to the test using simulated or artificial projects only, without obtaining results from experiments on real-world cases. This may negatively influence a project manager's perception of the practicality of the approach.

Secondly, the majority of the research works approach the problem as a (multi-) optimization problem in that they aim to minimize/maximize several objectives, with genetic algorithms being the most prevalent of approaches. The most popular objectives involve the cost and duration of the project—two of the three dimensions/constraints of software project success—through allocating and scheduling developers in such a way that the assignments yield a balance between the two.

Thirdly, some approaches consider software developers as interchangeable resources, especially when it comes to dealing with the skills required by tasks and the skills possessed by developers. Just because two developers possess the same skill, it does not mean that they will carry out a task in the same way or within the same time. Software developers are knowledge workers, and it is with this knowledge that software is built. The varying levels of skill proficiency and experience between developers can be directly related to the salary of developers as well as to the time it takes to carry out a task. Therefore, in approaches trying to allocate and schedule developers, it is important for software project managers to be able to factor in the variance caused by different levels of performance and productivity of developers.

Skill proficiency and experience levels are not the only things that differentiate developers. Performing human resource allocation and scheduling using only these technical aspects of software development means that other, non-technical aspects are neglected. Amrit (2005) argues that approaches that are based strictly on skills and experience may be inadequate for project managers to help them handle issues like interpersonal relationships among developers. Such human, social and cultural aspects are strongly exhibited in software development companies, especially as they become more reliant on teamwork and collaboration and the emergence of distributed development. For this reason, more and more research work is being carried out that tries to incorporate non-technical aspects, especially human-centric factors, involved in software development. The following section discusses one of the current directions in this trend and, in particular, the impact of the personality type of software developers. There have been many studies surrounding this human-centric factor, and additionally, various approaches have been proposed attempting to incorporate it into human resource management either as part of team formation strategies or as part of allocation and scheduling activities.

4.3 The Implication of Software Development Personality Types

4.3.1 Personality and Type Assessment

Personality psychology is the area of psychology that examines the person—the human individual. Every individual has a set of characteristics, both organized and dynamic, that come into action or that are expressed in certain situations regarding a person's cognitive, motivational and behavioural patterns. Over the years, many categories of personality theories have been developed, including trait theories, type theories, humanistic theories and behaviourist theories, all of which aim either to understand an individual's distinctive personality features or to identify general rules applying to different individuals. During the twentieth century, with the rapid growth of the field of personality psychology, there was an equal interest in the field of personality testing. The intensive research in the field has led to many additions and modifications of personality assessment instruments, both in approach and application.

One of the most widely administered personality tests is the Myers-Briggs Type Indicator (MBTI) (Briggs Myers et al. 1998), which scores individual preferences based on the works of Carl Gustav Jung (1923). Individuals answer a psychometric questionnaire that assesses preferences relating to four dichotomies: extraversion/introversion, sensing/intuition, thinking/feeling and judging/perceiving. The personality type of an individual is determined by which alternative of each dichotomy is preferred by answering a series of forced-choice questions. It should be noted that preference of one option does not mean that the other is never used—it is simply less preferred. As a result, there are 16 possible combinations of personality types.

Another well-known personality test is the Keirsey Temperament Sorter (Keirsey and Bates 1984), which is closely related to the MBTI but, instead, uses temperaments rather than attitudes and functions. The four temperaments (artisan, guardian, idealist and rational) can be subsequently broken down into roles and further into role variants based on an individual's preference towards behaviours that are concrete or abstract, cooperative or pragmatic, directive or informative and assertive or responsive. An individual's temperament and character type is measured using a 70-item forced-answer questionnaire.

The Revised NEO-Personality Inventory (NEO-PI-R) (Costa and McCrae 1992) was introduced to measure the Five-Factor Model (FFM) personality traits of individuals (Tupes and Christal 1961). The assessment determines emotional, interpersonal, experiential, attitudinal and motivational styles represented by the five domains—neuroticism, extraversion, openness to experience, agreeableness and conscientiousness—and their subdomains (facets). The 240 items comprising this psychological personality inventory contain descriptions of behaviours that are answered using a five-point scale.

Personality tests have been used extensively in a number of academic and application disciplines that have required the adoption of personality measures. More importantly, personality tests may also be utilized for career and personnel assessment. Even as far back as the early twentieth century, tests were used to investigate the desirable psychological abilities and traits an employee was required to have. For example, Münsterberg's (1913) theories and research work in the field of industrial/organizational psychology led to the widespread development and adoption of a personality test to measure and evaluate candidate employees. His test was put into practice by the Boston Elevated Company for selecting conductors as well as by the American Tobacco Company for choosing travelling salesmen. After World War II, the use of personality testing shifted its focus on assessing employees most suited for managerial and executive positions, and many companies, including IBM, started developing their own employee personality tests. Eventually, during the 1960s, testing restarted being practised at all levels and over a wider variety of occupations (Cox 2003).

Investigating whether or not a specific type of job can be executed by a particular personality type, especially for the heavily people-oriented field of software development, is very appealing to many organizations. It can help supervisors decide on issues such as pay rises and promotions or, in a negative light, disciplinary action and dismissal. As a result, several studies have been carried out to investigate whether software development professionals possess a specific type of personality. The outcomes of these investigations can shed light on the type(s) of personality that are drawn towards a career in software development. It also enables to explore whether different professions within the industry appeal to different personality types.

4.3.2 Personality Types of Software Development Professionals

One of the earliest studies of personnel in software engineering-related occupations was performed by Moore (1991). The study was based on the Sixteen Personality Factor Questionnaire (16PF), a popular measurement tool in personality-occupation studies and extensively used to assemble personality profiles for people in various occupations (Cattell et al. 1993). In the study, the author compiled the 16PF questionnaire for four software development occupation categories—application programmers, systems analysts, technical programmers and data processing managers—in an attempt to answer the question "*Do these groups of information systems professionals share a common personality profile, or are there significant differences?*" After multiple analyses, the authors found that managers and application programmers were most similar in that they are more inclined to experiment and think freely, thus allowing them to use their imagination more, while at the same time being more outspoken and comfortable with whatever happens.

In contrast with application programmers, however, managers are more likely to be laid-back and spontaneous, more forceful and competitive, and more capable of abstract thinking. Another finding showed that systems analysts and technical programmers have a tendency to be more practical, careful and conservative than data processing managers because their work is often highly visible, not only within data processing but throughout the company. Mistakes can be costly but also embarrassing. Additionally, the study also identified managers as "*less concerned with social rules than most people*" and more likely to pursue their own desires.

Wynekoop and Walz (1998) attempted to explore the differences between information systems professionals in order to determine whether or not differences existed in personality characteristics with the rest of the general population. They surveyed three oil and gas companies with a total of 114 programmer analysts, systems analysts and project managers and administered the California Psychological Inventory Adjective Check List (ACL) (Gough and Heilbrum 1983) on the employees. The results showed that managers and systems analysts are more similar to each other than to programmers. In addition, managers and systems analysts differ from the general population on more scales than programmers but also on different scales. Another finding was that managers tend to be more logical, compliant and with more confidence than the general population, whereas analysts are more willing to keep friendly relationships with others. Generally, the study shows that IT professionals have more leadership skills, are more ambitious and reasonable and have more self-esteem and can be more disciplined than other professionals. Similarly, Smith (1989) also carried out research on IT professionals, though his work concentrated only on the personality types of systems analysts. Based on MBTI type tests, the author concluded that a high majority of systems analysts tend to prefer sensing and thinking in addition to being more introvert rather than extrovert.

Capretz (2003) attempts to provide a personality profile of software engineering employees by distributing the MBTI instrument to 100 software engineers working for the government or for private companies and students of private or public universities and comparing the results to the distribution of MBTI types of the general US adult population. The motivation behind the author's research is the fact that the majority of software engineering professionals are typecast as "*nerds*"—introverts working alone in a corner and with no intentions to interact with others. However, over the years, software development has become more complex and has given rise to specialization within the profession (such as systems analysts, designers, programmers, testers, etc.), and as a result, each role requires a corresponding personality type. Furthermore, at the time of the study, there had been very little research carried out on the degree of job satisfaction among software professionals; any profile of the software engineer constructed may have been modified due to the growth of the field's popularity. The results of the author's survey showed that the majority of software engineers are technically oriented and prefer working with facts and reason rather than with people. It was also noted that systems analysts possessed a personality type that preferred to communicate with other people and to use their enhanced thinking ability to solve organizational problems. On the other hand, programmers exhibit a personality type that excels at spotting the centre of a problem and seem to find practical solutions. Conversely, some have a high need to achieve although a low drive to socialize with other people. It is a fact that the software development field is dominated by introverts, who typically have difficulty in communicating with end users. The greatest difference, according to the author, between software engineers and the general population is that the majority of software engineers take action based on what they think rather than what somebody else feels. This, however, does not help bring software developers closer to the users.

A more recent comprehensive investigation can be found in an analysis by Varona et al. (2012), which surveys existing studies that try to profile software development professions, in order to properly understand the human resources working in the software industry, as well as to spot possible trends and changes.

4.3.3 Allocating Developers to Tasks Based on Personality Types

Even if a specific personality type can be distinguished for each software development profession, the most important question is how to make use of this information in practice when trying to allocate and schedule human resources or form software development teams. There has been a gradual rise in the number of approaches aiming to help answer this question, and this section presents some of these approaches.

The personality type of a developer can play a significant role in determining which tasks he or she is assigned to because particular individual traits can help certain developers to be more adept in coping with the requirements and characteristics of a specific task. Furthermore, a more suitable personality type assigned to a task can have a direct influence on individual performance and team efficacy (Peeters et al. 2006; Capretz and Ahmed 2010b) and group conflict and team cohesion (Karn et al. 2007), as well as contribute to the overall quality of the final software product (Fernández-Sanz and Misra 2011). In addition, when a developer is assigned to a task that suits his or her personality, then his or her level of job satisfaction can increase leading to higher productivity (Acuña et al. 2009).

Dafoulas and Macaulay’s (2001) approach to assigning developers to tasks uses dynamic role allocation to maximize productivity and performance and takes into account certain role criteria (such as the goals and objectives, skills and knowledge, as well as any personality and culture requirements) so that project managers can assign/reassign roles or activities to team members according to their suitability.

Acuña and Juristo (2004) also consider roles and human capabilities in their attempts. Their proposed model first determines the intra-personal, organizational, interpersonal and management capabilities of team members and then performs role assignment to team members based on the capabilities required by the roles and the capabilities of the available resources. Each capability is allotted a number of personality traits required to be possessed using the 16PF test as a psychometric instrument. The goal is to assign those employees possessing the 16PF personality traits nearest to the 16PF personality traits required by the role (Acuña et al. 2006).

Similarly, André et al. (2011) developed a formal model for human resource allocation focusing on the assignment of developers to roles. In this approach, rules are generated to undertake the team formation process based on the roles and competencies of developers assessed through psychological tests. These team formation rules were converted into a formal model comprising four objective functions (competence, team compatibilities, availability and distance cost) and 12 constraint types to perform human resource assignment to roles by employing heuristic algorithms (random restart hill climbing, simulated annealing, tabu search and various other combinations of heuristic approaches).

Capretz and Ahmed (2010a, b) presented an attempt at human resource allocation suggesting a mapping of job requirements and skills to personality characteristics of employees, stating that the diversity of psychological types improves effectiveness and fulfillment of software developers. Because employees are more likely to perform better if they are assigned roles that their personality traits are best suited to, the authors associate hard skills (in the form of job requirements) to soft skills (in the form of personality requirements) for various software professionals: systems analysts, designers, programmers, testers and maintenance staff. The soft skills are then matched with specific personality characteristics based on MBTI personality types, and this can allow project managers to select team members with the same personality types and assign them to the roles required in the project.

Stylianou and Andreou (2012) employed a multi-objective genetic algorithm to simultaneously allocate and schedule software developers to tasks based on the technical skills and their personality types. One of the assumptions in this approach is that the schedule of tasks is fixed, and so the allocation of developers is constrained by the time that each task has been set to execute. This approach was recently developed into a prototype intelligent decision support tool in Stylianou et al. (2012) and was extended to also accommodate situations where a software project manager wishes to allocate and schedule software developers without having the project tasks scheduled beforehand. The results obtained in these two approaches appear highly promising and demonstrate the significance of human-centric developer assignment.

4.3.4 Allocate Developers to Team Based on Personality Types

While some may argue the importance of getting a developer to work on the right task, others may argue the significance of getting developers to work right together. The approaches mentioned in Sect. 4.3.3 all focus on the relationship between developers and tasks. However, what about the relationship between developers themselves? After all, software projects are undertaken by teams and require collaboration, coordination and communication between members. The answer to this question has been explored by several groups of researchers, all trying to identify how the personality type of developers influences various facets of team work and investigate whether certain combinations of personality types improve aspects such as performance, productivity and even quality. From a software project manager's perspective, this could help him or her to understand and exploit this underlying factor effectively when deciding on allocating developers to tasks.

One area of study concerns the heterogeneity of personality types, that is, the diversity of traits possessed by developers. Rutherfoord (2001) examined the impact of diversity by comparing teams comprising different personality types with teams composed of the same personality type using the Keirsey Temperament Sorter. The results showed that groups with members of the same personality type were having more personal problems, rather than technical. The surveys revealed that members seemed to want to elaborate the project by themselves and had problems with members that did not have much of a sharing discipline. On the other hand, groups with members of different personality types seemed to have more problems at a technical level. It was also noticed that groups where all members possessed a "*supervisor*" personality type were spending too much time discussing on how tasks will be assigned, despite this matter having already been decided previously. Groups where all members possessed an "*inspector*" personality type were very quiet, and interaction between them did not seem to exist. These groups appeared, however, much more focused and responsible. Groups with different personality types among their members were very active, had robust discussions and provided different kinds of ideas. The authors also noticed that groups with "*supervisor*" personality types were very opinionated and preferred to "*follow a traditional path.*" Research by Neuman et al. (1999) investigated the relationship between work team effectiveness and two other factors: team personality elevation (TPE), defined as "*the average level of a given trait within a team,*" and team personality diversity (TPD), described as "*the variability or differences in personality traits found within a team.*" Predicting job performance using personality has conventionally been based only on the elevation, or magnitude, of traits within the group, and this has been the foundation of selection and placement strategies. Nevertheless, the authors claim that team-based designs may also require taking into account the diversity, or variability, of traits within the group in order to find correlations between personality and job performance. The research used the FFM to examine the relationship between team personality composition and work team performance. Based on the authors' interpretation of the results, teams perform better when members differ in terms of extraversion and emotional stability rather than when members are similar in terms of these traits. Conversely, team performance is likely to increase if team members possess similarly high levels of traits regarding conscientiousness, agreeableness and openness to experience. Therefore, project management decisions on employee selection can be supported by taking into account the similarity of certain traits and the dissimilarity of others within a team.

Interestingly, a large number of research studies concentrate on the effects of personality in agile methodologies, which is in itself a relatively new development approach in the field of software engineering. Project management for agile methodologies is explored in Chap. 11, which describes how agile methodologies transform the way in which communication, collaboration and coordination practices in software development projects are carried out towards a more "*people-oriented*" approach where software teams are self-managing and share the decision-making. In this chapter, the discussion focuses particularly on pair programming and how personality types are implicated. This activity involves two developers working together on one task as they alternate between the roles of "*driver*"—the developer who codes—and "*navigator*"—the developer who reviews the code. Immediately, there is a need for social interaction (in the form of communication, collaboration and cooperation) among the developers in order to reach a common goal of delivering the unit produced on time and with the required quality. Hence, this is the reason why, especially in the past several years, studies have been carried out to investigate the impact that personality types have on the performance and productivity of the pairs.

Sfetsos et al. (2006) concentrated on the diversity of personality traits and came to the conclusion that pairs with heterogeneous personalities and temperaments exhibit better performance and collaboration-viability than pairs with similar personality traits. Software project managers, therefore, can take into account personality types when allocating developers to tasks and try to match developers so as to optimize the pair's effectiveness. Similarly, Choi et al. (2008) investigated which combination of personality types yields higher pair productivity. Specifically, they tested pairs of developers with alike, opposite and diverse combinations of personality types and found that the latter combination outperformed, in terms of code productivity, the other two.

In practice, a software development company may find it easier and cheaper if developers are left to team up by themselves. Oftentimes, however, pairs will be formed based on friendships and common interests and not on optimizing productivity. If personality types are taken into account, a software project manager can assign tasks to developers yielding maximum effect with relatively little time and cost.

Acuña et al. (2009) explored the relationship between each of the five factors of the FFM and job satisfaction, performance, team cohesion, task conflict and quality in agile settings. Their quasi-experiment produced a variety of results. Firstly, they observed that the quality of the end product is positively correlated to the preferred interpersonal style of the developers. This means that teams with a high average level of extraversion will enjoy the social interaction that is promoted through agile methodologies, and all members share the same goal of making the project a success. They also noted that developers with positive attitudinal and motivational styles are also more likely to be satisfied with their job. Developers in a team that share the same high level of agreeableness and conscientiousness feel more content with their career. Staying with the factors of the FFM, Salleh et al. (2010) explored how they especially affected pair programming. The main findings here were that pairings of developers with high levels of traits relating to openness to experience were conducive to the effectiveness of the pairings. Hannay et al. (2010) provide a comprehensive survey of the research investigating the effects of personality on pair programming and its ability to predict job performance.

4.3.5 Discussion

There are two schools of thought concerning the inclusion of information regarding the personality types of developers for human resource allocation and scheduling activities in software development. On the one hand, there is a view that a developer should be assigned to a task that he or she is more suitable based on the requirements of the task and the personality type of the developer. The claim is that each software development task has a set of characteristics and requirements that can be associated to a desired set of personality traits. For example, requirements elicitation tasks involve a high level of social engagement and the ability to identify with clients to understand their needs. Therefore, an introverted individual may struggle to perform these tasks as they are more reserved and prefer working alone rather than in environments requiring high social interaction. The research does not claim that a developer cannot carry out a task if he or she does not have the right personality type; it claims that he or she would not prefer to carry out the task. Consequently, a better task-fit for a developer would result not only in better performance but also in higher job satisfaction. The more fulfilled a developer is while working on a task that he or she is suited to, the more productive and efficient he or she is. Of course, this can only work if the developer is capable of carrying out the task in the first place with regard to technical skills, knowledge and expertise, so as not to jeopardize the quality of the software being developed.

On the other hand, there is the standpoint that a developer should be assigned with other developers so that the resulting combination of personality types leads to increased performance and effectiveness. The claim here is that there are certain combinations of personality traits that can improve the productivity of the team and increase the probability of success. Some traits, such as conscientiousness, should be present in all team members, while other traits, such as extraversion, should be diverse. Contrariwise, if several developers are assigned to work together on a task, their combination of personality types may not foster the most efficient and productive working environment. This does not mean that the job cannot get done; it may just mean that a more appropriate mixture of developers in terms of personality type may be able get the job done with improved levels of communication, collaboration and coordination, which are governed by an individual's personality type. Inevitably, if this personality type blend is not "*effectual*" there will be several knock-on effects, such as lowered productivity, job satisfaction and, ultimately, software quality.

Overall, there are a limited number of approaches that attempt to incorporate personality types of software developers in order to assign them to tasks, which is expected as this is still a relatively new direction. Some do not treat the allocation and scheduling of developers as an operational research problem and, therefore, do not employ the specialized techniques and methods as the approaches presented in Sect. 4.2 do. Those that do, attempt to optimize the allocation so that the developer whose personality type is closest to a desired profile is assigned. Interestingly, all but one approach overlook dealing with the problem of resource/task scheduling altogether, which as previously mentioned, is tough to separate from allocation as both activities are affected by developer availability constraints. Hence, the ability of approaches to provide an integrated tool may be considered limited unless they are able to accommodate scheduling also.

4.4 Further Research Trends and Challenges

Incorporating aspects of personality types in allocation and scheduling is still at a young and exploratory stage, and so the applicability of approaches lacks the backup of empirical evidence demonstrating their practical benefits in order to promote their adoption by real-world software development companies. A systematic evaluation of the effect is still required to be carried out to gather such evidences, and if these continue to indicate promising results, only then can a significant evolution in team formation, as well as allocation and scheduling strategies, occur.

The desired personality types of roles, tasks or activities that form the basis of assigning suitable developers in a number of approaches are not always justified empirically. It is important that the desired personality type of a task is correctly identified in order to allocate a suitable developer, but this may pose a challenge given the different personality measures and frameworks available to assess personality types and preferences. There is currently no consensus as to which personality instrument is the most capable of providing a task's desired personality accurately.

There is a difference of opinion with respect to how personality types can be utilized—for assigning tasks or for staffing teams. Either way, the emphasis remains on gathering evidence whether taking into account personality types of developers constitutes a legitimate way forward to help software project managers make staffing decisions aiming to increase the probability of success. Ideally, future approaches will be able to support both these valid research viewpoints.

Considering the use of personality types does not aim to single out developers or discriminate against them. Instead, it is supposed to provide software project managers with additional and complementary information to help them in the allocation and scheduling of resources or, in general, task-independent team formation. Additionally, it should not substitute or force to disregard important technical factors such as knowledge, skills and experience. However, some developers may still consider such an approach intrusive, so it is therefore important to provide reassurances that the goal is to utilize this human-centric factor to achieve maximum resource usage through the strengths of developers. Ultimately, the goals and objectives of any approach are to eliminate those risks in software project management preventing development organizations from delivering their products on time, within budget and with the required level of quality.

One of the other biggest challenges for the research community is trying to find a way of blending or "*marrying*" the two styles of solution. The interdisciplinary nature of the area requires many fields to come together to provide adequate and practical solutions for the software development industry. On the one hand, optimizing technical project criteria, such as cost, duration and number of defects, is attempted to be solved as an operational research problem, whereas the human-centric approaches tend to be handled as team formation strategies. Therefore, the ideal direction for research would be to concentrate on providing a hybrid of the two solution styles in a unified software project allocation and scheduling framework, on the one hand taking advantage of the benefits of underlying techniques (mathematical modelling or computational intelligence) and on the other hand targeting technical as well as non-technical, human-centric criteria. One of the obstacles to achieving this is quantifying and measuring human-centric criteria.

4.5 Concluding Remarks

The purpose of this chapter was to give the reader an insight of the most recent research approaches to human resource allocation and scheduling in software projects. What is observed is that there is a shift from the traditional operational research approach of allocating and scheduling developers based on conventional technical criteria, such as cost and duration, towards focusing on team formation and allocation strategies, aiming to make use of both the technical abilities of developers (e.g., skills and experience) as well as their personality types for improving other criteria, such as performance, productivity and software quality.

The traditional approaches to human resource allocation and scheduling consider the attempt to solve the problem as an optimization problem and make use of mathematical modelling techniques, such as linear programming and probabilistic modelling, in addition to computational intelligence methods, such as evolutionary algorithms and swarm intelligence. In order to determine the best allocation and scheduling plan, a software project manager would have to exhaustively evaluate all the possible permutations given the project's tasks, their durations, their dependency relationships and the skills they require, in addition to the available developers, their cost and the skills they possess. Especially with larger-sized software projects, this may prove overwhelming and time-consuming for a project manager. Approaches that adopt these methods therefore attempt to provide a quicker and easier alternative.

Allocating and scheduling human resources has started to move into a more human-centric direction, with a growth in the research area investigating the addition of non-technical aspects of software development to help software project managers increase the rate of software project success. One such aspect involves the use of personality types in allocation and scheduling. Research approaches, however, are still currently limited, but as this trend is becoming more popular, evidence is accumulating showing that personality types could indeed be used to help assess how well a developer would perform certain jobs and tasks and also how effective and/or productive he or she will be with other developers. Also, some studies have concluded that caution must be given in forming diverse software development teams as these appear to generally perform better than less heterogeneous teams. Overall, this particular area of research is very promising as it contributes to dealing with the important issue of helping software projects succeed by focusing on the most important, if not the only, resource involved in software development.

References

Acuña ST, Juristo N (2004) Assigning people to roles in software projects. Softw Pract Exp 34 (7):675–696

Acuña ST, Juristo N, Moreno AM (2006) Emphasizing human capabilities in software development. IEEE Softw 23(2):94–101

Acuña ST, Gómez M, Juristo N (2009) How do personality, team processes and task characteristics relate to job satisfaction and software quality? Inf Softw Technol 51(3):627–639

Alba E, Chicano JF (2005) Management of software projects with GAs. Paper presented at the 6th metaheuristics international conference, Vienna, Austria, 22–26 August, 2005

Alba E, Chicano JF (2007) Software project management with GAs. Inf Sci 177(11):2380–2401

Amrit C (2005) Coordination in software development: the problem of task allocation. Paper presented at the 27th international conference on software engineering, St. Louis, MO, 15–21 May, 2005

André M, Baldoquín MG, Acuña ST (2011) Formal model for assigning human resources to teams in software projects. Inf Softw Technol 53(3):259–275

Antoniol G, Cimitile A, Di Lucca GA, Di Penta M (2004a) Assessing staffing needs for a software maintenance project through queuing simulation. IEEE Trans Softw Eng 30(1):43–58

Antoniol G, Di Penta M, Harman M (2004) Search-based techniques for optimizing software project resource allocation. Paper presented at the 2004 genetic and evolutionary computation conference, Seattle, WA, 26–30 Jun 2004

Antoniol G, Di Penta M, Harman M (2005) Search–based techniques applied to optimization of project planning for a massive maintenance project. Paper presented at the 21st IEEE international conference on software maintenance, Budapest, Hungary, 26–29 Sept 2005

Barreto A, Barros MO, Werner CML (2005) Staffing a software project: a constraint satisfaction approach. ACM SIGSOFT Softw Eng Notes 30(4):1–5

Barreto A, Barros MO, Werner CML (2008) Staffing a software project: a constraint satisfaction and optimization-based approach. Comput Oper Res 35(10):3073–3089

Briggs Myers I, McCaulley MH, Quenk NL, Hammer AL (1998) MBTI® Manual: a guide to the development and the use of the Myers-Briggs type indicator®, 3rd edn. Consulting Psychologists, Mountain View, CA

Callegari DA, Bastos RM (2009) A multi-criteria resource selection method for software projects using fuzzy logic. Paper presented at the 11th international conference on enterprise information systems, Milan, Italy, 6–10 May 2009

Capretz LF (2003) Personality types in software engineering. Int J Hum Comput Stud 58(2):207–214

Capretz LF, Ahmed F (2010a) Making sense of software development and personality types. IT Prof 12(1):6–13

Capretz LF, Ahmed F (2010b) Why do we need personality diversity in software engineering? ACM SIGSOFT Softw Eng Notes 35(2):1–11

Cattell RB, Cattell AK, Cattell HEP (1993) 16PF fifth edition questionnaire. Institute for Personality and Ability Testing, Champaign, IL

Chang CK, Chao C, Hsieh S et al (1994) SPMNet: a formal methodology for software management. Paper presented at the 18th annual international computer software and applications conference, Taipei, Taiwan, 9–11 Nov 1994

Chang CK, Chao C, Nguyen TT, Christensen MJ (1998) Software project management net: a new methodology on software management. Paper presented at the 22nd annual international computer software and applications conference, Vienna, Austria, 19–21 Aug 1998

Chang CK, Christensen MJ, Zhang T (2001) Genetic algorithms for project management. Ann Softw Eng 11(1):107–139

Chang CK, Jiang H, Di Y et al (2008) Time-line based model for software project scheduling with genetic algorithms. Inf Softw Technol 50(11):1142–1154

Chen W, Zhang J (2013) Ant colony optimization for software project scheduling and staffing with an event-based scheduler. IEEE Trans Softw Eng 39(1):1–17

Chicano F, Luna F, Nebro AJ et al (2011) Using multi-objective metaheuristics to solve the software project scheduling problem. Paper presented at the 13th annual conference on genetic and evolutionary computation, Dublin, Ireland, 12–16 Jul 2011

Choi KS, Deek FP, Im I (2008) Exploring the underlying aspects of pair programming: the impact of personality. Inf Softw Technol 50(11):1114–1126

Costa PT Jr, McCrae RR (1992) NEO inventories professional manual. Psychological Assessment Resources, Inc., Odessa, TX

Cox AM (2003) I am never lonely: a brief history of employee personality testing. Stay Free! 21:22–24

Dafoulas GA, Macaulay LA (2001) Facilitating group formation and role allocation in software engineering groups. Paper presented at the 2001 ACS/IEEE international conference on computer systems and applications, Beirut, Lebanon, 25–29 Jun 2001

Di Penta M, Harman M, Antoniol G (2011) The use of search-based optimization techniques to schedule and staff software projects: an approach and an empirical study. Softw Pract Exp 41 (5):495–519

Duggan J, Byrne J, Lyons GJ (2004) A task allocation optimizer for software construction. IEEE Softw 21(3):76–82

Ejnioui A, Otero CE, Otero LD (2012) A multi-attribute decision making approach for resource allocation in software projects. In: Arabnia HR, Reza H, Xiong J (eds) Proceedings of the 2012 international conference on software engineering research and practice, Las Vegas, 12–16 June 2012

Fernández-Sanz L, Misra S (2011) Influence of human factors in software quality and productivity. Paper presented at the 2011 international conference on computational science and its applications, Santander, Spain, 20–23 Jun 2011

Ge Y (2009) Software project rescheduling with genetic algorithms. Paper presented at the 2009 international conference on artificial intelligence and computational intelligence, Shanghai, China, 7–8 Nov 2009

Ge Y, Chang CK (2006) Capability-based project scheduling with genetic algorithms. Paper presented at the 2006 international conference on computational intelligence for modelling, control and automation and international conference on intelligent agents web technologies and international commerce, Sydney, Australia, 28 Nov–1 Dec 2006

Gerasimou S, Stylianou C, Andreou AS (2012) An investigation of optimal project scheduling and team staffing in software development using particle swarm optimization. Paper presented at the 14th international conference on enterprise information systems, Wrocław, Poland, 28 Jun–1 Jul 2012

Gough HG, Heilbrun AB Jr (1983) The adjective checklist manual. Consulting Psychologists Press, Inc., Palo Alto, CA

Gueorguiev S, Harman M, Antoniol, G (2009) Software project planning for robustness and completion time in the presence of uncertainty using multi objective search based software engineering. Paper presented at the 11th annual conference on genetic and evolutionary computation, Montréal, Canada, 8–12 Jul 2009

Hannay JE, Arisholm E, Engvik H, Sjoberg DIK (2010) Effects of personality on pair programming. IEEE Trans Softw Eng 36(1):61–80

Hapke M, Jaszkiewicz A, Slowinski R (1994) Fuzzy project scheduling system for software development. Fuzzy Sets Syst 67(1):101–117

Heiat A (2002) Comparison of artificial neural network and regression models for estimating software development effort. Inf Softw Technol 44(15):911–922

Jalote P, Jain G (2004) Assigning tasks in a 24-hour software development model. Paper presented at the 11th Asia-Pacific software engineering conference, Busan, Korea, 30 Nov–3 Dec 2004

Jiang H, Chang CK, Xia J, Cheng S (2007) A history-based automatic scheduling model for personnel risk management. Paper presented at the 31st annual international computer software and applications conference, Beijing, China, 24–27 Jul 2007

Jung CG (1923) Psychological types (H. Godwin Baines Trans.). London, England: Routledge; Kegan Paul Ltd

Kantorovich LV (1940) A new method of solving some classes of extremal problems. Doklady Akad Sci USSR 28:211–214

Kapur P, Ngo-The A, Ruhe G et al (2008) Optimized staffing for product releases and its application at Chartwell technology. J Softw Maint Evol R 20(5):365–386

Karn JS, Syed-Abdullah S, Cowling AJ, Holcombe M (2007) A study into the effects of personality type and methodology on cohesion in software engineering teams. Behav Inf Technol 26 (2):99–111

Keirsey D, Bates M (1984) Please understand me: character and temperament Types. Prometheus Nemesis Book Company, Del Mar, CA

Khoshgoftaar TM, Seliya N (2004) Comparative assessment of software quality classification techniques: an empirical case study. Empir Softw Eng 9(3):229–257

Li C, van den Akker J. M., Brinkkemper S, Diepen G (2007) Integrated requirement selection and scheduling for the release planning of a software product. Paper presented at the 13th international working conference on requirements engineering: foundation for software quality, Trondheim, Norway, 11–12 Jun 2007

Luna F, Gonzalez-Alvarez DL, Chicano F, Vega-Rodriquez MA (2011) On the scalability of multi-objective metaheuristics for the software scheduling problem. Paper presented at the 11th international conference on intelligent systems design and applications, Córdoba, Spain, 22–24 Nov 2011

McBride T (2008) The mechanisms of project management of software development. J Syst Softw 81(12):2386–2395

McCabe TJ (1976) A complexity measure. IEEE Trans Softw Eng 2(4):308–320

Michael CC, McGraw GE, Schatz MA, Walton CC (1997) Genetic algorithms for dynamic test data generation. Paper presented at the 12th international conference on automated software engineering, Lake Tahoe, 1–5 Nov 1997

Moore JE (1991) Personality characteristics of information systems professionals. Paper presented at the 1991 ACM SIGCPR conference on computer personnel research, Athens, GA, 8–9 Apr 1991

Münsterberg H (1913) Psychology and industrial efficiency. The Riverside Press, Cambridge, USA

Neuman GA, Wagner SH, Christiansen ND (1999) The relationship between work-team personality composition and the job performance of teams. Group Organ Manag 24(1):28–45

Ngo-The A, Ruhe G (2009) Optimized resource allocation for software release planning. IEEE Trans Softw Eng 35(1):109–123

Otero LD, Centeno G, Ruiz-Torres AJ, Otero CE (2009) A systematic approach for resource allocation in software projects. Comput Ind Eng 56(4):1333–1339

Otero CE, Otero LD, Weissberger I, Qureshi A (2010) A multi-criteria decision making approach for resource allocation in software engineering. Paper presented at the 12th international conference on computer modelling and simulation, Cambridge, England, 24–26 Mar 2010

Padberg F (2001) scheduling software projects to minimize the development time and cost with a given staff. In: Anonymous eighth Asia-Pacific software engineering conference (APSEC 2001), Macao, China, 4–7 Dec 2001. IEEE Computer Science Press, Los Alamitos, CA, pp 187–194

Padberg F (2002) Using process simulation to compare scheduling strategies for software projects. Paper presented at the 9th Asia-Pacific software engineering conference, Gold Coast, Australia, 4–6 Dec 2002

Padberg F (2003) A software process scheduling simulator. Paper presented at the 25th international conference on software engineering, Portland, OR, 3–10 May 2003

Padberg F (2004) Computing optimal scheduling policies for software projects. Paper presented at the 11th Asia-Pacific software engineering conference, Busan, Korea, 30 Nov–3 Dec 2004

Padberg F (2006) A study on optimal scheduling for software projects. Softw Process Improv Pract 11(1):77–91

Peeters MAG, van Tuijl HFJM, Rutte CG, Reymen IMMJ (2006) Personality and team performance: a meta-analysis. Eur J Pers 20(5):377–396

Raymond L, Bergeron F (2008) Project management information systems an empirical study of their impact on project managers and project success. Int J Proj Manag 26(2):213–220

Ren J, Harman M, Di Penta M (2011) Cooperative co-evolutionary optimization of software project staff assignments and job scheduling. Paper presented at the 2011 international symposium on search based software engineering, Szeged, Hungary, 10–12 Sept 2011

Rutherfoord RH (2001) Using personality inventories to help form teams for software engineering class projects. ACM SIGCSE Bull 33(3):73–76

Salleh N, Mendes E, Grundy J, St. John Burch G (2010) An empirical study of the effects of conscientiousness in pair programming using the five-factor personality model. Paper presented at the 32nd ACM/IEEE international conference on software engineering, Cape Town, South Africa, 2–8 May 2010

Sfetsos P, Stamelos I, Angelis L, Deligiannis I (2006) Investigating the impact of personality types on communication and collaboration-viability in pair programming – an empirical study. Paper presented at the 7th international conference on extreme programming and agile processes in software engineering, Oulu, Finland, 17–22 Jun 2006

Smith DC (1989) The personality of the systems analysts: an investigation. ACM SIGCPR Comput Pers 12(2):12–14

Stylianou C, Andreou AS (2007) A hybrid software component clustering and retrieval scheme using an entropy-based fuzzy k-modes algorithm. Paper presented at the 19th IEEE international conference on tools with artificial intelligence, Patras, Greece, 29–31 Oct 2007

Stylianou C, Andreou AS (2012) A multi-objective genetic algorithm for software development team staffing based on personality types. Paper presented at the 8th IFIP WG 12.5 international conference on artificial intelligence applications and innovations, Halkidiki, Greece, 27–30 Sept 2012

Stylianou C, Gerasimou S, Andreou, AS (2012) A novel prototype tool for intelligent software project scheduling and staffing enhanced with personality factors. Paper presented at the 24th IEEE international conference on tools with artificial intelligence, Athens, Greece, 7–9 Nov 2012

Tsai H, Moskowitz H, Lee L (2003) Human resource selection for software development projects using Taguchi's parameter design. Eur J Oper Res 151(1):167–180

Tupes EC, Christal RE (1961) Recurrent personality factors based on trait ratings. J Pers 60 (2):225–251

Varona D, Capretz LF, Piñero Y et al (2012) Evolution of software engineers' personality profile. SIGSOFT Softw Eng Notes 37(1):1–5

Wynekoop JL, Walz DB (1998) Revisiting the perennial question: are IS people different? ACM SIGMIS Database 29(3):62–72

Xiao J, Ao X, Tang Y (2013) Solving software project scheduling problems with ant colony optimization. Comput Oper Res 40(1):33–46

Yannibelli V, Amandi A (2011) A knowledge-based evolutionary assistant to software development project scheduling. Expert Syst Appl 38(7):8403–8413

Biography Constantinos Stylianou is a Ph.D. student at the Department of Computer Science of the University of Cyprus. His research focuses on aspects of software project management and specifically on the use of intelligent techniques for human resource scheduling and allocation in addition to human-centric factors in software development. He is also a research member of the Software Engineering and Intelligent Information Systems Research Lab of the Cyprus University of Technology.

Andreas S. Andreou is an Associate Professor and Vice-Chair of the Department of Electrical Engineering/Computer Engineering and Informatics of the Cyprus University of Technology. He is the Director of the Software Engineering and Intelligent Information Systems Research Lab, where his areas of interest include Software Engineering, Web Engineering, Electronic and Mobile Commerce and Intelligent Information Systems.