Artificial Intelligence in Software Project Management

Liyan Song and Leandro L. Minku

Abstract The success of a software project highly depends on how well the project is managed. This includes crucial activities such as estimating the effort required to develop the software project, creating a software project schedule including allocation of human resources, managing project risks, monitoring progress, etc. Inadequate handling of such activities can thus lead to serious consequences to software companies. However, software project management is also a very challenging task that involves multiple business and human factors, and conflicting objectives. These challenges are exacerbated when dealing with medium to large size software projects. Under such circumstances, artificial intelligence has the potential to play a significant role in supporting software project managers, enabling them to make more informed management decisions. This chapter discusses how artificial intelligence techniques can support software project managers with two key software project management activities: software project scheduling and software effort estimation.

Liyan Song
Department of Computer Science and Engineering, Southern University of Science and Technology, China, e-mail: songly@sustech.edu.cn

Leandro L. Minku
School of Computer Science, University of Birmingham, UK, e-mail: l.l.minku@bham.ac.uk

Version note: this is the final author accepted version, which contains place holders for the references to other sections and chapters of the book, instead of the references themselves.

Software project management is a software engineering task concerned with ensuring that the software is delivered on time and on budget, and in accordance with the requirements of the stakeholder organisations [1]. It includes crucial activities such as estimating the effort required to develop the software project, creating a software project schedule including allocation of human resources, managing project risks, monitoring progress, etc.

Inadequate handling of such activities can lead to serious consequences to software development companies. For instance, if the software project goes over budget, the company will lose profit and may even get a negative profit margin. Depending on how large the monetary loss is, the software company could even go bankrupt. If the software project overruns, the software development company may lose a client organisation or even be in breach of contract.

However, software project management is also a very challenging task that involves multiple business and human factors, and conflicting objectives. These challenges are exacerbated when dealing with medium to large size software projects. Under such circumstances, Artificial Intelligence (AI) has the potential to play a significant role in supporting software project managers, enabling them to make more informed management decisions. This chapter discusses how AI techniques can support software project managers with two key software project management activities: software project scheduling and software effort estimation.

Software project scheduling is the process of organising the work to be done in a software project into different tasks and deciding who will work on which task and when [1]. AI techniques can be used to help software project managers with finding good allocations of employees to tasks with certain objectives in mind, such as minimising the cost and duration of the project. Section 1 introduces software project scheduling and AI approaches that can be used to support it, focusing on the work of Alba and Chicano [2], Minku et al. [3] and Shen et al. [4] as examples.

Software effort estimation is the process of predicting the effort required to develop a software project. AI can be used for software effort estimation as a decision support tool, based on which project managers can justify, criticise or adjust the estimation derived by the experts. Such AI-based estimations are repeatable, objective, efficient, and can often provide better understanding of the estimation process. There have been many AI approaches proposed for software effort estimation, and we will take the effort estimator proposed by Song et al.'s [5] as an example to explain typical procedures of adopting AI for effort estimation. Section 2 introduces software effort estimation and AI techniques to support it.

1 Software Project Scheduling (SPS)

Software Project Scheduling (SPS) consists in organising the work to be done in a software project into different tasks and deciding who will work on which task and when [1]. It requires identifying the tasks to be performed in a software project; the dependencies among the tasks; the employees available to perform tasks, their skills and salaries; estimating the effort required to develop each task; allocating employees to tasks; and ultimately producing charts to communicate the project tasks, their duration and employee allocation. In more traditional software development processes, a detailed schedule is produced at the beginning of the project, and this schedule is then adjusted as the project is developed. In agile software development, the initial schedule is typically more coarse, identifying the different phases of the project. More detailed schedules are then produced for each phase or iteration of the project during its development [1].

An illustrative example of project Gantt chart that could have been produced as part of the SPS process at the beginning of a project is given in Figure 1. In this project, there are nine tasks t_0 to t_8 and three employees e_0 to e_2. Employees can be allocated with different amounts of dedication to tasks (shown in parentheses in the Gantt chart). This dedication corresponds to the percentage of the employee's full time work that is dedicated to each task. For example, if an employee's full time work is 8 hours per day, then a dedication of 50% would correspond to 4 hours per day. Tasks can be allocated to more than one employee at the same time, e.g., e_0 and e_2 are both allocated to t_0. A given employee can also be allocated to more than one task that occur concurrently. For instance, employee e_0 is allocated to both t_1 and t_4, which co-occur from Day 16 to Day 20. At the beginning of t_1, e_0 works with 100% dedication to this task, but at the end of this task e_0 shares their time between t_1 and t_4, dedicating 50% of their time to each of these tasks. Employee e_0 remains working with 50% dedication to t_4 until the end of this task, even though this employee was not allocated to any other concurrent task in this project. It could be, for example, that this employee was dedicating the other 50% of their time on another project.

Typically, schedules are created with certain objectives in mind. For instance, one may be interested in producing a schedule that minimises the cost and duration of the project. For example, the schedule illustrated in Figure 1 is estimated to take 42 days to complete. Possibly, this project could have taken less time to complete if more employees were allocated to it. However, allocating more employees might include some employees with higher salaries, meaning that the cost of the project could potentially increase. The cost of a software project can be calculated based on the salaries of the employees allocated to it plus the cost of any other resources that may need to be used to develop the project.

Creating a software project schedule can be a daunting task especially when the project is large and the company has a large number of employees. The space of possible allocations of employees to tasks can be enormous [6]. The complexity of the allocations can also be high, given that employees can work in parallel on different tasks and each task can be allocated to more than one employee at the same time. Constraints such as employees being unable to work on tasks for which they do

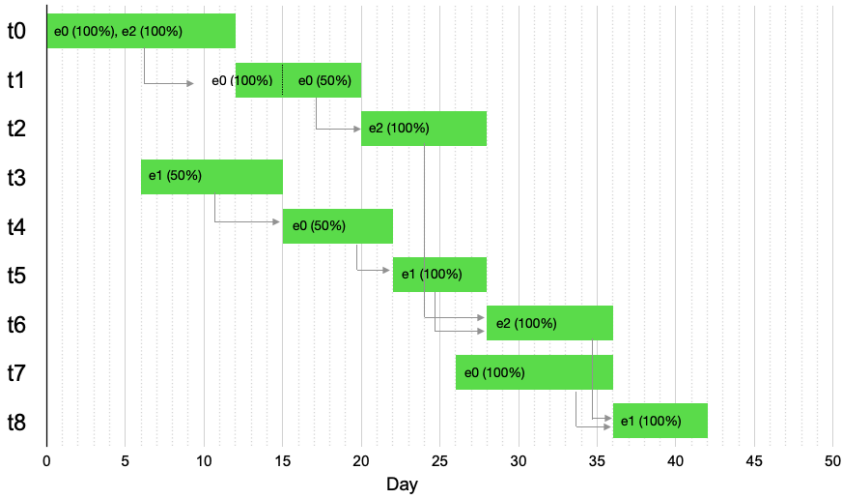

Fig. 1 Illustrative example of project Gantt chart with nine tasks t_0 to t_8 and three employees e_0 to e_2. Arrows represent dependencies between tasks. Percentages in brackets represent the percentage of the employee's full time work dedicated to the corresponding task. Each vertical gray line represents the end of a given working day.

not have the required skills, employees being unable to work more than a maximum number of hours per day, and task dependencies add to the challenge [3]. Moreover, the objectives that one may be interested in optimising when creating a schedule may frequently be conflicting. For instance, a schedule that allocates expensive staff (those with higher salaries) may lead to a more expensive project, but may result in a shorter duration. Producing an optimal schedule manually is thus a difficult problem.

An alternative to producing a schedule completely manually is to adopt AI approaches. These approaches would still require project managers to decide how to split the project into different tasks and provide information such as task dependencies, employees' salaries and skills. However, based on such information, AI could support software managers in creating project schedules with the aim of optimising certain objectives such as project cost and duration. Section 1.1 discusses some of the existing AI approaches for SPS.

1.1 AI Approaches for SPS

Despite SPS being typically a human activity, the objectives that the software manager may be interested in optimising when creating a schedule (e.g., project cost and duration) can be mathematically formulated. Therefore, SPS can be seen as an optimisation problem, i.e., a problem where we are interested in finding a solution that minimises or maximises one or more objective functions. Such formulation

enables the SPS problem to be solved by AI approaches called approximate search algorithms, such as metaheuristics [7] (see Chapter). Solutions generated automatically by AI can then be used to support software managers in making more informed decisions during the SPS process.

Several different formulations of SPS as an optimisation problem have been investigated in the literature. For example, some researchers formulate SPS as the problem of finding the order with which tasks should be undertaken and the assignment of employees to tasks that minimises the duration of the software project [8], or the cost (salaries paid) and the amount of penalties incurred from missing milestone deadlines in the project [9]. Others formulate it as the problem of finding the allocation of employees to tasks that minimises the duration and cost (salaries paid) of the software project, whilst the order with which tasks are performed is calculated by a separate deterministic algorithm based on the allocations and dependencies among tasks [2]. Different problem formulations can also take into account different levels of detail. For example, some formulations take into account different salaries for work in normal hours or overtime [9], the tasks' required skills and employees' skills [9, 3], the level of proficiency of employees on different skills [9, 4], the potential mistakes in the estimations of the effort required to complete tasks [4], the dynamic events that may affect the software project during its development [4], etc.

Section 1.1.1 presents a popular problem formulation that was proposed in [2], and briefly discusses its extension [4] to take into account additional aspects relevant to SPS. Section 1.1.2 briefly discusses the metaheuristic proposed in [3] to solve this problem and its extended version [4]. Even though the proposed problem formulations and metaheuristics have been proposed in [2, 3, 4], this book chapter concentrates on discussing them in a more didactic manner.

1.1.1 An SPS Problem Formulation

Alba and Chicano [2] formulated the SPS problem as the problem of finding an allocation of employees to tasks that minimises the cost and duration of the project. This formulation is a landmark formulation that has inspired other more detailed formulations [4]. As it is a simple formulation that is easy to understand, this section will focus on it, and then briefly explain how it was extended to include more details that are relevant to realistic SPS scenarios. We explain what information this formulation requires software managers to provide about the project and available employees, how a candidate solution to the SPS problem looks like in this formulation, what objectives are intended to be optimised and what constraints a solution must satisfy to be feasible.

Information About The Software Project And Available Employees

Alba and Chicano's problem formulation [2] considers that the following information about the software project and the employees available to work on it is provided by the software manager:

- **Employees –** there are n employees $e_0, e_1, \ldots, e_{n-1}$ available for the project, with salaries $s_0, s_1, \ldots, s_{n-1}$, sets of skills $sk_0, sk_1, \ldots, sk_{n-1}$, number of normal working hours in a day $h_0, h_1, \ldots, h_{n-1}$, and maximum dedication $md_0, md_1, \ldots, md_{n-1}$. Table 1 provides an illustrative example of information that could have been provided by a software manager for $n = 6$ employees.

Table 1 Example of Employees Available For a Given Project

Employee Name	Hourly Salary	Skills	Normal Hours	Maximum Dedication
e_0 = John	s_0 = \$10	sk_0 = { Python, SQL }	h_0 = 8 hours	md_0 = 1
e_1 = Tom	s_1 = \$30	sk_1 = { C++, TCP/IP, HTTP }	h_1 = 8 hours	md_1 = 1
e_2 = Jack	s_2 = \$25	sk_2 = { Java, Javascript, SQL, JSON, Testing }	h_2 = 8 hours	md_2 = 1
e_3 = Claire	s_3 = \$27	sk_3 = { UML, Java, SQL, Testing, Mocks }	h_3 = 8 hours	md_3 = 1
e_4 = Mary	s_4 = \$35	sk_4 = { Data science, Python, SQL, JSON }	h_4 = 8 hours	md_4 = 1
e_5 = Junior	s_5 = \$20	sk_5 = { C++, TCP/IP, HTTP }	h_4 = 8 hours	md_5 = 1

  The maximum dedication can be specified as a percentage of the normal working day. For instance, 1 means 100% of a normal working day, whereas 1.25 means 125% of a normal working day. Assuming that a normal working day has 8 hours as in Table 1, a maximum dedication of 1 would mean $1 \cdot 8 = 8$ hours per day, i.e., the employee is not allowed to work overtime. A dedication of 1.25 would mean $1.25 \cdot 8 = 10$ hours per day, i.e., the employee is allowed to work up to 2 hours overtime.
  In practice, a much larger number of employees may be available than those in the example provided in Table 1. More details about the employees could also be specified by extending this problem formulation. For instance, Shen et al. [4] also considered each employee's overtime salary and level of proficiency on each skill. Such extra information can be important as it would affect the cost and duration of the project. In addition, employees' availabilities may be affected by dynamic events that may occur over time. For instance, an employee may become unavailable to work due to illness, or may leave the company. Therefore Shen et al. [4] also associated each employee to a status indicating whether this employee is currently available or not to work. A change in such status is considered as a critical event that immediately triggers rescheduling of the project, so that the project schedule remains feasible and its cost and duration remains competitive.
- **Tasks –** the project has m tasks $t_0, t_1, \ldots, t_{m-1}$ with required efforts $ef_0, ef_1, \ldots, ef_{m-1}$ and sets of required skills $rsk_0, rsk_1, \ldots, rsk_{m-1}$. Table 2 provides an illustrative example of tasks for a project with $m = 10$ tasks to develop a system to monitor and track cattle in a farm.
  The tasks are determined by the software project manager, and could be either coarser grained tasks (e.g., architecture design, database design, communication protocol design, front end development, back end development, front end testing, etc), or more detailed tasks resulting from the initial design of the system (e.g., develop a specific class, a specific table for the database, etc). The required efforts are also determined by the software manager, potentially with the support of AI

Table 2 Example of Tasks in a Cattle Monitoring and Tracking Project

Task Name	Required Effort	Required Skills
t_0 = Design Database	ef_0 = 24 person-hours	rsk_0 = {SQL}
t_1 = Implement Database	ef_1 = 24 person-hours	rsk_1 = {SQL}
t_2 = Design GUI	ef_2 = 32 person-hours	rsk_2 = {Javascript}
t_3 = Implement GUI	ef_3 = 40 person-hours	rsk_3 = {Javascript}
t_4 = Design Communications Protocol	ef_4 = 80 person-hours	rsk_4 = {TCP/IP}
t_5 = Implement Communications Protocol Class	ef_5 = 40 person-hours	rsk_5 = {TCP/IP, C++}
t_6 = Implement Cattle Class	ef_6 = 40 person-hours	rsk_6 = {Java, Javascript}
t_7 = Implement Monitor Class	ef_7 = 40 person-hours	rsk_7 = {C++}
t_8 = Implement Data Transmission Class	ef_8 = 80 person-hours	rsk_8 = {C++}
t_9 = Test System	ef_9 = 160 person-hours	rsk_9 = {Testing}

approaches for software effort estimation such as the ones presented in Section 2. Coarser task granularity may mean that each task is actually composed of many sub-tasks that have dependencies with each other, potentially making their effort estimation more difficult. Finer granularities may require more detailed design, but their required effort may be easier to estimate.

In the example shown in Table 2, we assume that a UML diagram for the classes that compose the system has already been designed by the software project manager, leading to a more detailed set of tasks. In practice, a much larger number of tasks may need to be scheduled, especially when dealing with large projects and when a more detailed level of granularity is used. There may also be much more tasks related to software testing, e.g., testing each of the classes separately before an integration test.

Shen et al. [4] extended this problem formulation to also consider that new tasks may be added as a result of changes in the requirements of the project during the software development lifecycle. New tasks can be critical or non-critical tasks. New critical tasks immediately trigger rescheduling of the project, so that they can be incorporated into the project schedule as soon as possible. Regular tasks are accumulated over time and are only incorporated into the project schedule when another critical event occurs or when they need to start so that other existing tasks that depend on it can commence.

- **Task Precedence Graph –** tasks in software projects typically have precedence relations, i.e., some tasks cannot start before other tasks are completed. For instance, the implementation of a GUI cannot start before its design is completed. This could be captured in the form of a task precedence graph, where each node represents a task and an arc going from a given task t_i to t_j means that task t_j depends on the completion of task t_i. Figure 2 shows an illustrative example of task precedence graph that could have been generated by the software manager for our illustrative project.

It is worth noting that a software project may also suffer from other types of uncertainty and dynamic events in addition to changes in employees' status (available or non-available) and new tasks. For instance, there may be changes in task precedence, new employees being hired by the company, removal of tasks from the

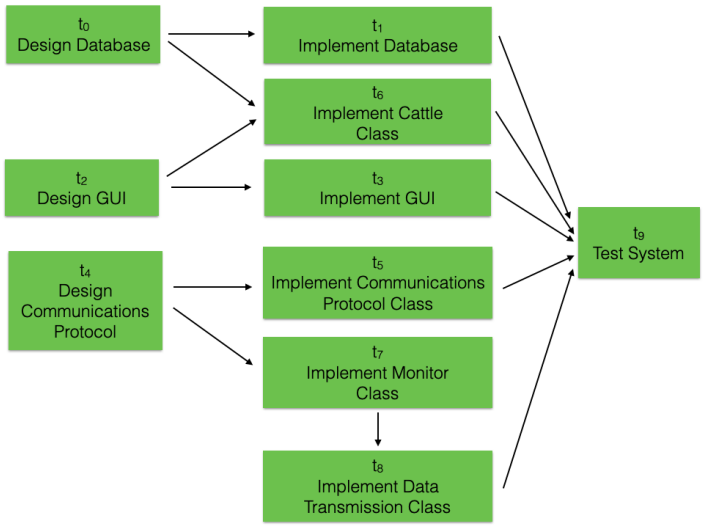

Fig. 2 Example of Task Precedence Graph for a Cattle Monitoring and Tracking Project

project due to changes in requirements, etc. These other dynamic events were not taken into account in Shen et al. [4]'s work, but could potentially be dealt with in a similar way to changes in employees' status and new tasks.

Candidate Solutions and Gantt Charts

A candidate solution to the SPS problem corresponds to a matrix of dedications of employees to tasks. An AI algorithm to solve the SPS problem will thus attempt to automatically find a good allocation of employees to tasks, specifying the dedication of each employee to each task.

An example of possible (not necessarily optimal) solution generated manually for our Cattle Monitoring and Tracking project is shown in Table 3. Each cell $[i, j]$ contains a numeric value corresponding to the percentage of employee e_i's normal working time dedicated to task t_j. For example, employee e_0 will dedicate 100% of their time (i.e., 8 hours per day) to task t_0 when this task is active, whereas employee e_5 will dedicate 50% of their time (i.e., 4 hours per day) to task t_5 when this task is active. The dedications have a pre-defined granularity that prevents assigning employees to tasks with too fine grained dedications, which would be difficult to follow in practice. For instance, we could say that dedications can only take the values of 0, 0.25, 0.50, 0.75 and 1. It is possible for a given employee not to be allocated to any task. For example, employees e_1 and e_4 were not allocated to this project despite being available. This could have been done, for example, because these two employees are the ones with the highest salaries, and the software manager might be trying to reduce the cost of the project.

Based on the project tasks, task precedence graph and a given solution, it is possible to generate a Gantt chart for the project in an automated manner. The Gantt

Table 3 Example of Solution for Scheduling a Cattle Monitoring and Tracking Project. The numbers in each cell $[i, j]$ correspond to the percentage of employee e_i's normal working time dedicated to task t_j.

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
e_0	1	1	0	0	0	0	0	0	0	0
e_1	0	0	0	0	0	0	0	0	0	0
e_2	0	0	1	0.25	0	0	0.5	0	0	1
e_3	0	0	0	0	0	0	0	0	0	1
e_4	0	0	0	0	0	0	0	0	0	0
e_5	0	0	0	0	1	0.5	0	0.5	1	0

chart for the solution in Table 3 is shown in Figure 3. An algorithm that can be used to generate this Gantt chart is presented in [3]. This algorithm iteratively adds tasks that can commence given the task precedence graph to the Gantt chart, with their lengths calculated based on the tasks' required effort and the total amount of dedication of employees allocated to them.

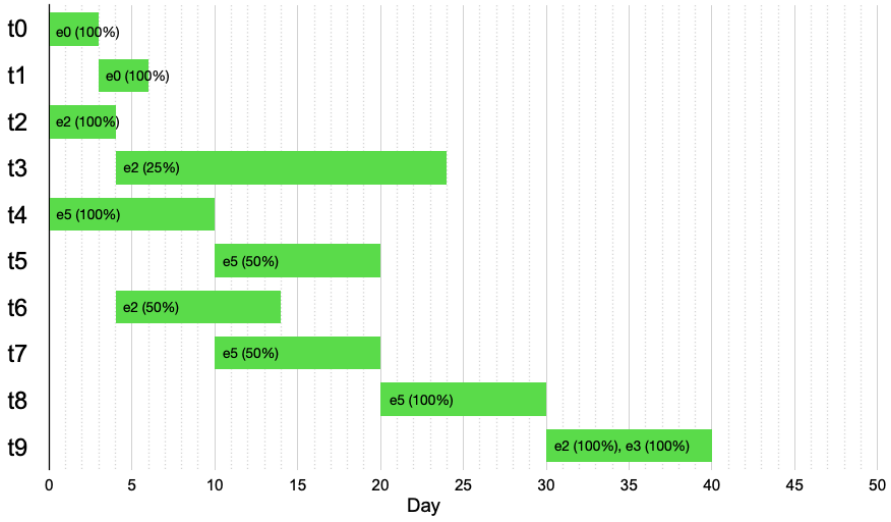

Fig. 3 Gantt Chart for the Solution from Table 3. Task dependencies are illustrated in Figure 2 and are omitted here so that the Gantt Chart is easier to read. The duration of the project is 40 days and the project cost is $12,240 based on this solution.

The time it takes for a task to complete is calculated by dividing the task's required effort by the sum of the dedications of all employees allocated to this task [2]. For instance, in our example project, task t_0 has no dependency from the beginning, and can thus start on Day 1 as shown in Figure 3. This task takes 3 days to complete. This is because it requires 24 person-hours (Table 2) and one employee (e_0) is allocated to it with $1 = 100\%$ of their time (Table 3). So, this task will require $24/1$ hours (=3 days) to be completed. If there were two employees assigned to this task, one with a

dedication of 1 and the other with a dedication of 0.5, then this task would require $24/(1 + 0.5) = 16$ hours (=2 days) to be completed.

Simply dividing the task's required effort by the sum of the dedications of the employees allocated to it means that the time it takes to complete a task decreases linearly with the total amount of dedication allocated it. This means that AI SPS algorithms following this problem formulation could suggest software managers to allocate a very large number of employees to tasks in an attempt to reduce the project duration. However, it is well known that larger teams lead to communication overheads [10]. A larger total dedication resulting from a large number of employees allocated to a task would result in such communication overhead. Shen et al. [4] extended this problem formulation by applying a penalty which increases the task's required effort if the team allocated to it exceeds a maximum size. They also extended the formulation to consider that employees with lower proficiency on the skills required for the task would cause this task to take longer to complete. Such extensions are important because they would prevent AI algorithms from simply allocating an excessively large number of employees to a given task to reduce its duration.

Objectives

Alba and Chicano's formulation [2] considers two objectives to be minimised: project cost and duration. An AI algorithm for the SPS problem based on this formulation would thus help the software manager to find an allocation that minimises the project cost and duration. These objectives are computed as follows:

- **Duration –** the duration of the project can be automatically obtained when the Gantt chart is generated. For example, in Figure 3, we can see that the project will take in total 40 days to be completed, as task t_9 is the last task to be completed and it finishes at the end of Day 40.
- **Cost –** this formulation considers that the cost of the project corresponds to the total amount of salaries paid to the employees allocated to the project. In practice, other costs may also be incurred, depending on whether other resources are required for the development of the project. However, these other costs would normally be unrelated to the project schedule itself, and so are not taken into account in the formulation. The total amount of salaries paid can be determined automatically based on the Gantt chart and the information about the employees' salaries. In particular, for each employee, we need to compute the total number of hours that they will spend on the project. We can then calculate the total salary paid for this employee. For example, for the solution presented in Table 3, employee e_0 works in total 3 days for t_0 and 3 days for t_1 (= 6 days $\cdot$ 8 hours = 48 hours). So, their total payment is $48 \cdot \$10 = \$480$. After computing the salaries paid to each employee, we can sum the salaries paid for all employees to determine the cost of the project. The total cost of the project based on this Gantt chart is \$12,240.

Besides the two objectives above, Shen et al. [4] also considers two additional objectives:

- **Robustness –** the task effort estimations may involve uncertainty, i.e., the actual task efforts may not be equal to the estimated ones. Therefore, the allocation of

employees to tasks should ideally be as robust as possible to potential variations in task effort. The robustness objective evaluates such robustness. In general terms, it represents the average increase in project cost and duration obtained when simulating variations in task effort based on a Gaussian distribution. The higher the increase in cost and duration, the worse the robustness of the candidate solution to task effort uncertainty.

- **Stability –** software projects may suffer changes over time, such as changes in the status (available or unavailable) of employees and new tasks being added. Whenever a critical event occurs, the project needs to be immediately rescheduled so that its cost and duration remains feasible and competitive. To avoid project disruption, the new allocation of employees to tasks should ideally not be too different from the previous one. The stability objective measures the amount of changes in the dedication of employees to tasks that were previously available in the project. The larger the amount of changes, the more disruptive and undesirable the new project schedule is.

Constraints

Alba and Chicano's formulation [2] also takes into account certain constraints. A solution can only be feasible if it satisfies these constraints. If a given solution does not satisfy these constraints, it is considered infeasible and cannot be adopted for the software project being scheduled. An AI algorithm would thus attempt to find a solution that not only minimises the project cost and duration, but also satisfies the constraints. These constraints are explained below:

- **Maximum Dedication –** at any point during the project, the total dedication of a given employee e_i to tasks should not exceed the maximum dedication md_i.
  For example, based on the Gantt chart from Figure 3, employee e_5 works concurrently on tasks t_5 and t_7 from days 11 to 20. As this employee's dedication to each of these tasks is 0.5, this employee is working with a total dedication of $0.5 + 0.5 = 1$ from days 11 to 20. This does not exceed the maximum dedication of this employee ($md_1 = 1$). Therefore, the maximum dedication constraint is satisfied during this period of time. However, had the dedication of this employee to task t_7 been 0.75, this employee's total dedication during this period would have been $0.5 + 0.75 = 1.25$, which exceeds their maximum dedication of $md_1 = 1$. In this case, the maximum dedication constraint would have been violated, i.e., the solution would have been infeasible.
  Minku et al. [3] proposed to fix infeasible solutions by adjusting their corresponding Gantt charts. This means that the AI algorithm itself would not need to try and find a solution that satisfies this constraint. Instead, any solution that violates this constraint would be fixed into a Gantt chart that satisfies this constraint. The strategy to fix the Gantt chart is called "normalisation" and its idea is as follows. Consider the infeasible solution discussed above, where the dedication of employee e_5 to tasks t_5 and t_7 is 0.5 and 0.75, respectively. The total dedication of this employee to tasks from Days 11 to 20 is 1.25. If we divide the dedications to tasks t_5 and t_7 by 1.25 when creating the Gantt chart from days 11 to 20, the

resulting dedications will be $0.5/1.25 = 0.4$ to task t_5 and $0.75/1.25 = 0.6$ to task t_7 from days 11 to 20. Therefore, the constraint would not be violated anymore.
Such fixes are only applied for the periods of time when violations would have occurred. Therefore, an employee may initially have a given dedication to a task, and then decrease this dedication at a later date in order to avoid a violation. For instance, in the Gantt chart from Figure 1, employee e_0 started to work on task t_1 with a dedication of 1 and then reduced it to 0.5 once task t_4 started. Such reduction in the dedication to task t_1 avoided a total dedication larger than 1 from Day 16 to Day 20. It also enabled the project to complete faster, because employee e_0 did not need to work on the whole task t_1 with a dedication of 0.5, only from Day 16 onward. This enabled this task to complete faster than if employee e_1 was working with dedication 0.5 during this whole task.

- **Required Skills –** the team (group of employees) allocated to a given task must, together, have all the skills required to conduct that task. For example, a team composed of one employee who knows Java and one employee who knows SQL, together, holds the skills Java and SQL. This team could thus be allocated to a task that requires Java and SQL, but could not be allocated to a task that requires Java and Javascript. Any solution that assigns a team of employees that do not, together, hold all skills required to develop that task is an infeasible solution.
This definition for the required skills constraint has some problems. In particular, each employee in the team does not need to have all the skills required by the task, so long as the team as a whole has all skills. For instance, it could happen that all employees from Table 1 are allocated to task t_9 of Table 2, even though only employees e_2 and e_3 have testing skills. Employees with no proficiency on the skills required for a given task would be unable to efficiently work on this task, as they would have to learn the skill on the go. Shen et al. [4] extended this problem formulation to take this into account when computing the duration of each task as mentioned in the explanation of how to generate the Gantt chart. Therefore, an AI algorithm to solve the SPS problem based on the extended formulation would only allocate an employee who is not proficient on a given task's required skill only if this would bring a worthwhile benefit to the cost and duration of the project as a whole.
Alternatively, it is possible to replace Alba and Chicano [2]'s required skills constraint by a constraint that states that each and every employee allocated to a given task must have all the skills required to develop that task. For instance, if a given task requires Testing and SQL in our example project, the only employees that could be assigned to this task would be e_3 and e_4.

1.1.2 An AI Algorithm for Solving SPS

Given a specific SPS problem formulation, AI algorithms can be adopted to solve it. Different approximate search algorithms have been applied to the SPS problem [7], among which evolutionary algorithms are among the most popular. Evolutionary algorithms are explained in Section . The current section explains an example of

evolutionary algorithm design for SPS. This algorithm design was proposed by Minku et al. [3] and is based on the problem formulation introduced by Alba and Chicano [2], but adopting the normalisation strategy to fix the Gantt chart to avoid violations of the maximum dedication constraint as explained in Section 1.1.1. This strategy, together with some other design choices, enabled this algorithm to significantly outperform other existing algorithms designed for the same problem formulation.

As explained in Section 1.1.1, this problem formulation does not take into account certain details that are important when scheduling software projects in practice. Therefore, it was extended in subsequent work such as the work of Shen et al. [4]. However, the more detailed problem formulation proposed by Shen et al. requires a more complex evolutionary algorithm to be solved, which in turn requires advanced knowledge of evolutionary algorithms to understand. Therefore, this section focuses on Minku et al.'s algorithm [3]. In practice, software managers would not need to understand in detail how different evolutionary algorithms for SPS work to be able to adopt them, as they would only need to interact with a software tool that implements such algorithms by inputting information about the employees and tasks in order to obtain an allocation.

The evolutionary algorithm proposed by Minku et al. [3] for SPS is depicted in Algorithm 1. It requires us to pre-define certain parameters, namely the population size μ, number of parents λ, probability of crossover P_c and maximum number of iterations *MaxIt*. The best values for these parameters may vary depending on the software project in hands. However, default values of $\mu = 64$, $\lambda = 64$, $P_c = 0.75$, $MaxIt = 69$ led to good results in experiments with several simulated software projects with different features in [3]. Therefore, if the software manager is not familiar with evolutionary algorithms, we recommend the use of these default parameter values.

Algorithm 1 Evolutionary Algorithm for SPS

Parameters: population size μ, number of parents to be selected λ, probability of crossover P_c, maximum number of iterations *MaxIt*.

1: Initialise population P with μ candidate solutions.
2: **repeat**
3: Select λ parents from P using binary tournament selection.
4: **for** each pair of parents A and B **do**
5: With probability P_c, apply crossover between A and B to generate A' and B'.
6: Otherwise, $A' \leftarrow A$ and $B' \leftarrow B$
7: Apply mutation to each cell of A' and B' using probability $P_m = 1/(mn)$.
8: $P \leftarrow P \cup \{A', B'\}$.
9: **end for**
10: Select the μ best candidate solutions from P to survive for the next generation.
11: **until** maximum number of iterations *MaxIt* is reached

The algorithm first initialises a population of candidate solutions P with μ candidate solutions (Line 1). Then, a loop is repeated until the maximum number of iterations *MaxIt* is reached. In this loop, λ parent solutions are selected based on

binary tournament selection (Line 3). For each pair of parents A and B, crossover is applied with probability P_c to generate children A' and B' (Line 5). If crossover is not applied, the children A' and B' are clones of the parents (Line 6). Mutation is then applied to the children with a probability $P_m = 1/(mn)$ (Line 7), where m is the number of tasks and n is the number of available employees. The children are then added to the population P (Line 8). The μ best candidate solutions (i.e., those with the best fitness values) from the population P are then selected to survive for the next generation.

As mentioned in Section , some of the key design choices of evolutionary algorithms involve deciding on an appropriate representation, crossover and mutation operators, fitness function and strategy to deal with constraints. The algorithm proposed by Minku et al. [3] makes use of a direct representation of the candidate solutions, i.e., a matrix of dedications of employees to tasks with a pre-determined granularity as explained in Section 1.1.1 and illustrated in Table 3.

Crossover between two parent solutions A and B is applied with a pre-defined probability P_c. When crossover is applied, one of the following strategies is randomly used to generate two children solutions A' and B':

- Exchange Rows – for each employee, select its corresponding dedications to tasks from one randomly chosen parent to compose child A', and from the other parent to compose child B'. This can be seen as exchanging rows of the matrix of dedications between the parents A and B. An example of crossover by exchanging rows is given in Figure 4 for a project with three employees and four tasks.

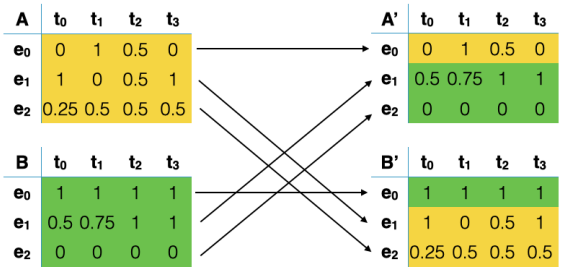

Fig. 4 Example of crossover by exchanging rows being applied to two parents A and B shown (left), leading to two children A' and B' (right). The rows used to compose child A' are picked randomly from parent A or B, and the other rows are used to compose child B'.

- Exchange Columns – for each task, select its corresponding employees' dedications from one randomly chosen parent to compose child A', and from the other parent to compose child B'. This can be seen as exchanging columns of the matrix of dedications. An example of crossover by exchanging columns is given in Figure 5 for a project with three employees and four tasks.

Mutation is applied independently to each cell of a child with probability $P_m = 1/(mn)$, where n is the number of employees and m is the number of tasks. This

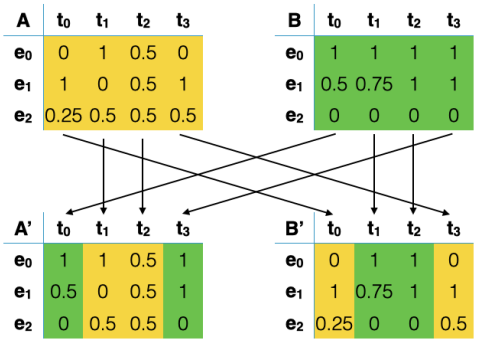

Fig. 5 Example of crossover by exchanging columns being applied to two parents A and B (top), leading to two children A' and B' (bottom). The columns used to compose child A' are picked randomly from parent A or B, and the other columns are used to compose child B'.

means that, on average, one cell of the matrix of dedications gets mutated for each child. The mutation consists in replacing the current dedication value of the cell by any other possible dedication value. For example, consider the dedication granularity of 0, 0.25, 0.5, 0.75, 1 and a given cell with value 0.75. If this cell is mutated, another value is picked randomly from 0, 0.25, 0.5, 1, e.g., 0. An example of that is shown in Figure 6.

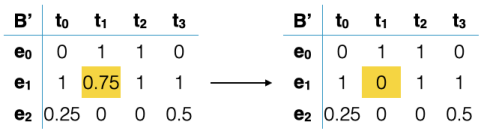

Fig. 6 Example of mutation being applied to child B' (left), leading to a mutated child (right). Each cell has a probability of $1/(3 \cdot 4) \approx 0.08$ of being mutated to a new randomly picked dedication value. In this example, this resulted in the dedication of employee e_1 to task t_1 being mutated from 0.75 to 0.

Another important component of an evolutionary algorithm is its fitness function and its strategy to deal with constraints. The fitness function adopted in [3] is the weighted average between the project cost and duration incurred by the solution, as shown below:

$$f(X) = w_{cost} \cdot cost(X) + w_{duration} \cdot duration(X) \quad (1)$$

where *cost*(X) and *duration*(X) are the cost and duration of solution X, and w_{cost} and $w_{duration}$ are pre-defined positive real values used to control how much emphasis the software manager would like to place on the cost and duration of the project, respectively. These weights can also work to make the magnitude of the cost values

similar to that of the duration values. For example, if we know the project cost would be in the region of ten thousands and the duration would be in the region of hundreds, we could set the weights as $w_{cost} = 0.001$ and $w_{duration} = 0.1$ so that the cost does not dominate the computation of fitness values. Setting the weights as $w_{cost} = 0.01$ and $w_{duration} = 0.1$ would increase the importance of cost when scheduling this project. This would guide the algorithm towards finding an allocation that leads to a cheaper, but possibly longer project.

Good solutions have smaller fitness values according to this function. To deal with the required skills constraint, the $cost(X)$ and $duration(X)$ of the fitness function is forced into a very large value which makes the fitness of any infeasible solution worse than that of any feasible solution. If Alba and Chicano [2]'s original required skills constraint is adopted, this value gets worse when larger numbers of skills are missing in the teams allocated to the tasks. Similarly, if the alternative required skills constraint that states that each and every employee of a team needs to have all skills required by the task is adopted, this value gets worse when larger number of tasks are allocated inadequate teams. This helps guiding the algorithm to find feasible solutions.

The maximum dedication constraint is automatically dealt with by fixing the Gantt chart of infeasible solutions by using the normalisation strategy explained in Section 1.1.1.

1.2 Running an AI Algorithm for SPS

This section gives an example of how to run an AI algorithm for SPS. The tool used here implements Minku et al. [3]'s evolutionary algorithm for SPS and also includes some variations of it. The variation that we will experiment with in this section is the one that adopts the alternative required skills constraint that requires each and every employee allocated to a given task to have all the skills required for that task.

The tool was implemented for research purposes, making use of text files to collect information about the project and available employees, and for retrieving the solution in the form of an allocation of employees to tasks. The tool can either be run from the command line or via a user interface, but this user interface is mainly used for the purpose of setting up the evolutionary algorithm to be adopted and its parameters.

In practice, an intelligent SPS tool would have a user interface for collecting information about the project (such as the information from Table 2 and Figure 2) and available employees (as in Table 1), and for displaying the solution in the format of an allocation of employees to tasks (as in Table 3) and its corresponding Gantt chart (as in Figure 3). The setting of the evolutionary algorithm itself would be considered as an advanced setting that software managers would not need to change unless they are familiar with evolutionary algorithms and would like to investigate whether different settings would lead to better solutions.

The tool is implemented in Java and is available at [11] under GNU GLP 3.0 license. It makes use of the Opt4j framework for meta-heuristic optimisation [12], which supports the implementation of different evolutionary algorithms and is available under the MIT license. Once you download the tool, you will see that there is a folder called *problem-instance-examples*. This folder contains nine different examples of toy software projects to be scheduled. One of them is called *instance_sample_book.txt*, which we will use as a running example in this book. This example corresponds to the illustrative project discussed in Section 1.1.2. In particular, the employees are the ones listed in Table 1, the tasks are the ones in Table 2 and the task precedence graph is the one in Figure 2.

The format of this file follows the format of the generator introduced by Alba and Chicano [2]. Comment lines can be added by starting a line with #. When opening this file, you will note that the first lines contain comments listing the numeric identifiers that are being used to represent each skill. The numeric identifiers used for the employees and tasks are the same as the ones used in Tables 1 and 2.

After the commented lines, the file contains a number of statements that define the information from Tables 1 and 2 and Figure 2, i.e., the information about the employees, tasks and task precedence graph, respectively. The order of the statements in the file does not matter. The file requires the following statements:

- employee.number – number of employees n.
- task.number – number of tasks m.
- skill.number – number of skills.
- graph.arc.number – number of arcs in the task precedence graph (e.g., the graph from Figure 2 has 12 arcs).
- employee.i.skill.number – number of skills for employee e_i.
- employee.i.salary – salary of employee e_i.
- employee.i.skill.j – skill sk_j of employee i. The number j varies from 0 to employee.i.skill.number-1 and is used just to list employee.i.skill.number different skills for the employee. The skills themselves (e.g., Java, SQL, etc) are represented by other numeric identifiers which have been listed in the first commented lines of the file for our information.
- task.i.skill.number – number of skills required by task t_i.
- task.i.cost – required effort of task t_i.
- task.i.skill.j – skill rsk_j required by task t_i. The number j varies from 0 to task.i.skill.number-1 and is used just to list the different skills required by the task. The skills themselves (e.g., Java, SQL, etc) are identified by other numeric values which have been listed in the first commented lines of the file for our information.
- graph.arc.i – definition of arc i of the task precedence graph. The number i varies from 0 to graph.arc.number-1 and is used just to list the different arcs that define the task precedence graph. A line "graph.arc.i=j k" means an arc going from task t_j to task t_k, meaning that task t_k depends on task t_j.

This implementation assumes that all employees always have a maximum dedication of 1. In addition, the unit used for the employees' salaries must be consistent

with the unit of required effort for the tasks. For instance, if the required effort is in person-hours, the salary must be an hourly rate. The monetary unit (e.g., USD) of the salary itself does not matter, so long as all salaries are using the same monetary unit. The project cost will be calculated using the same monetary unit as the salaries. The number of hours of a normal working day is not used by this implementation, as it computes the Gantt chart using the same time unit as the required effort. For instance, if the required effort is in person-hours, the x-axis of the Gantt chart is in hours, rather than days.

The file *GA.xml* contains the setup of the evolutionary algorithm explained in Section 1.1.2. The other files *OnePlusOneEA.xml* and *RLS.xml* contain other algorithms that have also been analysed by Minku et al. [3]. Most information in this file would not need to be set by the software manager, except for the following:

- Weights w_{cost} and $w_{duration}$ explained in Section 1.1.2. These weights can be set in the following lines of the GA.xml file, where `wCost` refers to w_{cost} and `wDuration` refers to $w_{duration}$:

```
<property name="wCost">0.01</property>
<property name="wDuration">0.1</property>
```

- The granularity of the dedications of employees to tasks. The granularity is set so that the dedications can assume values $0, 1/\mathtt{k}, 2/\mathtt{k}, \ldots, 1$. This is specified through the `k` value in the following line:

```
<property name="k">4</property>
```

- Name of the file containing information about the project to be scheduled. In our example, this is the file `instance_sample_book.txt`. This is specified in the following line:

```
<property name="pspInstanceFileName">
instance_sample_book.txt</property>
```

- Name of the output log file where the solution will be saved. This is specified in the following line:

```
<property name="filename">outputGALog.csv</property>
```

Should the software manager wish to further modify this file to adopt different parameters for the evolutionary algorithm, or to change to a different evolutionary algorithm, further instructions are given in the file *readme.pdf* found with the code.

To run the approach, the following command can be used:

```
java -cp pspea.jar:opt4j-2.4.jar:junit.jar \
org.opt4j.start.Opt4JStarter GA.xml
```

The output log file obtained when running the command line above is given in the file *example-output-log/outputGALog-wcost0.01.csv*. Each row corresponds to a different iteration (generation) of the evolutionary algorithm and logs information about:

- Iteration – this is the number of the iteration whose information is being recorded.
- Evaluations – this is the number of fitness evaluations performed up to the point when the information is recorded.
- Runtime[s] – the runtime elapsed until then.
- COSTDUR[MIN] – the value of the fitness function for the best solution (i.e., the solution with the smallest fitness value) of this iteration.
- Cost – the cost of the project based on the best solution of the population in this iteration.
- Duration – the duration of the project based on the best solution of the population in this iteration.
- Undt – number of tasks for which at least one employee in the team does not have *all* the skills required by the task. For example, if a given task t_j requires skills 0, 1 and 2, and at least one employee allocated to this task does not have all these three skills, this task will count as one of such tasks. When undt > 0, this means that the required skills constraint is violated. This column is only used when the alternative required skills constraint that states that each and every employee allocated to a task must have all skills required by that task is adopted.
- Reqsk – this is unused for the problem formulation discussed in this section.
- Overwork – this is unused for the problem formulation discussed in this section, as the normalisation strategy is adopted to deal with this constraint when creating the Gantt chart associated to the project. So, there is never overwork under this problem formulation.
- PhenotypeBeforeNormalisation – columns from this one onward contain the dedications of employees to tasks for the best solution of this iteration. This is printed based on the code below, where allocation is the matrix of dedications of employees to tasks, n is the number of employees and m is the number of tasks:

```
for (e=0; e<n; ++e)
      for (t=0; t<m; ++t)
            print(allocation[e][t] + ",");
```

As a research tool, this implementation does not produce a plot containing the Gantt chart for the solution, even though it computes the Gantt chart internally in order to calculate the fitness value. Instead of plotting the Gantt chart, this implementation only records the solution itself. In practice, a decision support tool implementing this algorithm would also produce a plot with the Gantt chart.

In matrix format, the solution retrieved by the algorithm based on $w_{cost} = 0.01$ and $w_{duration} = 0.1$ is shown in Table 4. This solution places more emphasis on cost than duration, but still allows duration to play a significant role. The project cost resulting from this configuration was $13,066.67 and the duration was 237.33 hours (29.66 days, if we assume that each working day has 8 hours). This is 25.82% (10.33 days) shorter than the duration based on the manual solution from Figure 3, with a cost just 6.75% ($826.67) higher.

When compared to the manual solution presented in Table 3, we can see that this allocation includes employee e_1 with some dedication to tasks t_4, t_5 and t_8. This enabled such large speed ups. As the dedications were not high, the cost of the project

Table 4 Solution for the Running Example when Using $w_{cost} = 0.01$ and $w_{duration} = 0.1$.

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
e_0	0.75	1	0	0	0	0	0	0	0	0
e_1	0	0	0	0	0.5	0.5	0	0	0.25	0
e_2	0	0	1	1	0	0	1	0	0	1
e_3	0	0	0	0	0	0	0	0	0	1
e_4	0	0	0	0	0	0	0	0	0	0
e_5	0	0	0	0	1	0	0	1	1	0

did not increase much. Moreover, the evolutionary algorithm did not recommend to allocate employee e_4 to any task. This is because adding this employee would cause an increase in the cost of the project without reducing its duration.

The Gantt chart corresponding to this solution is shown in Figure 7. This Gantt chart figure was not outputted by the research tool adopted in this section. It was generated manually for the purpose of this book. However, in practice, an SPS tool would have generated such Gantt chart as part of the output. From the Gantt chart, we can see that the algorithm adopted the normalisation strategy. In particular, employee e_2 is the only employee who can work on tasks t_3 and t_6, as these tasks require the Javascript skill. These tasks co-occur from around Day 6 to Day 11. The normalisation strategy enabled this employee to be allocated full time (dedication of 1) to both tasks without leading to overwork. In particular, it adjusted the Gantt chart so that employee e_2 starts the work on task t_3 with dedication of 1, then switches to dedication of 0.5 to task t_3 and 0.5 to task t_6 when these tasks co-occur, then switches to dedication of 1 to task t_6 when task t_3 finishes.

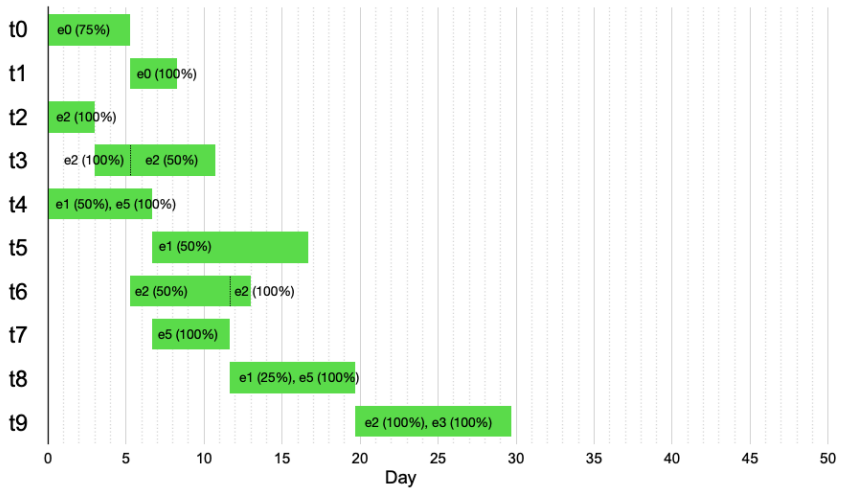

Fig. 7 Gantt Chart for the Running Example when Using $w_{cost} = 0.01$ and $w_{duration} = 0.1$. This project schedule has a duration of 29.66 days and a cost of $13,066.67.

If we assume a more extreme scenario where the software manager would like to save cost as much as possible no matter the increases that this could cause to the duration of the project, we could adopt an even higher weight for the cost (e.g., $w_{cost} = 1$). This can be achieved by replacing the value of the property *wCost* in *GA.xml* by 1. In this case, the tool would help the software manager to identify that employees e_1, e_3 and e_4 do not need to be allocated to this project as their skills are covered by other employees, and that employee e_2 should be allocated to task t_9, as this is the cheapest employee who can do this task. This would lead to a cost of $12,080 and a duration of 400 hours (=50 days). The output generated by the tool for such scenario is available in the file *example-output-log/outputGALog-wcost1.csv*.

If we assume a scenario where the software manager would like to give a more balanced importance to cost and duration, we could adopt a weight $w_{cost} = 0.003$ instead of 0.01 or 1. In matrix format, the solution retrieved by the algorithm based on this weight is shown in Table 5. The algorithm produced a solution where the project cost was $13,440 (9.8% more expensive than in the manual solution presented in Table 3) but with a duration of just 200 hours (25 days, 37.5% shorter than in the manual solution).

Table 5 Solution for the Running Example when Using $w_{cost} = 0.003$ and $w_{duration} = 0.1$.

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
e_0	0.75	1	0	0	0	0	0	0	0	0
e_1	0	0	0	0	1	1	0	1	1	0
e_2	0	0	1	0.5	0	0	0.5	0	0	1
e_3	0	0	0	0	0	0	0	0	0	1
e_4	0	0	0	0	0	0	0	0	0	0
e_5	0	0	0	0	1	0	0	1	1	0

This was achieved by allocating more employees to each task, but only allocating more expensive employees when this was really useful to reduce the duration, as the project cost still plays a significant role in the fitness calculation. Therefore, employees e_3 and e_4 were still not allocated to the project, whereas employee e_1 was allocated full time to tasks t_4, t_5, t_7 and t_8. From the Gantt chart presented in Figure 8, we can see that the allocation of employee e_1 was very helpful to reduce the duration of the project by working together with the cheaper employee e_5 on tasks t_4, t_7 and t_8, which have a direct impact on the duration of the project due to their dependencies to each other. In terms of task t_5, one might have thought that it would have been better to swap the allocations of employees e_1 and e_5, so that task t_5 is performed by the cheaper employee e_5. However this would not really have reduced the cost of the project. Indeed both employees e_1 and e_5 are working full time from Day 1 to Day 15, meaning that swapping the roles of e_1 and e_5 would not help.

The illustrative project being scheduled here is a small project, not being too challenging to allocate manually. However, even this way, a significant amount of reflections need to be done to decide who is worth allocating or not. With larger projects in companies with a larger number of employees, this challenge increases. By using an AI tool, software managers would be able to get different suggestions

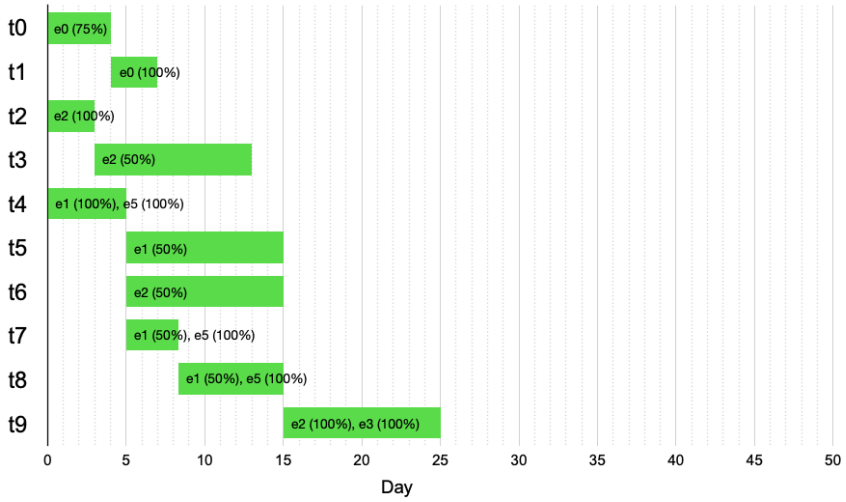

Fig. 8 Gantt Chart for the Running Example when Using $w_{cost} = 0.003$ and $w_{duration} = 0.1$. This project has a duration of 25 days and a cost of $13,440.

with different trade-offs between cost and duration (based on different weights w_{cost} and $w_{duration}$), helping them to decide on a good schedule to adopt.

Shen et al.'s algorithm [4] also considers that the project may suffer disruptions during its execution. For example, a given employee may need a sick leave. This algorithm enables project managers to enter this information during the project, and then produces an alternative schedule that takes such disruptions into account while not causing too many changes to the original schedule. This can be very helpful to maintain the cost and duration of the project as similar to the original one as possible, while not changing the original schedule too much.

Even though the evolutionary algorithm proposed by Shen et al. [4] is more complex than the ones proposed by Minku et al. [3], the way to run the two algorithms would be similar, i.e., the tool would require software managers to input information about the employees, tasks and task precedence, and would output an allocation of employees to tasks.

2 Software Effort Estimation (SEE)

Software Effort Estimation (SEE) is the process of predicting the effort (e.g. in person-months) required to develop a software project. It is typically the first step of the software development process and can contribute to a successful software project management, constituting the basis for the subsequent steps such as planning, estimating cost, managing and reducing project risks [13, 14, 15].

When estimating the effort of a software project as a whole in the traditional waterfall-like software development, software requirements of the customers are elicited at the beginning of the project and typically remain the same or do not change much throughout the software development process. Therefore, SEE often takes place in the early stages of software development, and based on it project managers make important decisions such as the budget, the bidding price and the subsequent planning and control [16]. The estimation is typically made based on information regarding the project and the team intended to work on the project, such as functional size (a metric estimating the size of the software to be developed), software development type (new development, enhancement, re-development) and development team skills (low, medium, high) [17, 18].

When adopting agile software development processes, software is typically developed incrementally in iterations, catering for feedback and requirements from the customers [19]. Specifically, at the end of each developing iteration, developers would release the software at the current status to the customer requiring for feedback such as new functionalities and changes to implemented functionalities. Developers and customers then negotiate and agree on the requirements to be developed in the next iteration. In this scenario, the effort for each iteration (not for the remaining iterations of the project) is typically estimated right before starting to develop this iteration. Therefore, SEE often takes place multiple times throughout the entire process [20, 21]. Similar to the traditional development process, the estimation of an agile project to be developed is typically made based on information regarding the project's iteration and the team assigned to work it, such as task size in this iteration, development team's skills (low, medium, high) and development team's experience (with / without prior experience on the task) [22, 20, 23].

Good effort estimation is important for software project management. Both over- and under-estimation can cause serious problems to the organisation. Over-estimation may result in a company losing bids for contracts or wasting resources. Under-estimation may lead to poor product quality, unsatisfied customers, delayed or even incomplete software systems [14, 15]. For example, NASA had to shut down its incomplete Checkout & Launch Control System project after it massively exceeded the budget [24]. Nevertheless, providing accurate effort estimations is very challenging. Project managers have mainly relied on expert judgement to produce estimations. As a result, the estimation quality is highly affected by the experience and the capability of the experts [25, 26]. Such expert-based estimation typically suffers from a number of issues: they can be costly, the prediction process is not explicit, they are not repeatable, they can be influenced by irrelevant factors such as being sensitive to political pressure, and they can have personal bias [25, 27].

Artificial Intelligence (AI) can be used for SEE as a decision support tool. Based on estimations provided by AI models, software project managers can potentially justify, criticise or adjust the estimation derived by the experts. Such model-based estimation can potentially circumvent problems faced by the experts: they are repeatable, objective, efficient, and can often provide better understanding of the estimation process [15]. Therefore, AI approaches could potentially provide a more justifiable effort estimation than those decided based on 'gut feeling'. They could also be used to investigate "what–if" scenarios, where required efforts when using different development resources, different development teams and so on could be compared to each other, enabling practitioners to make more informed planning decisions. Section 2.1 explains the main process of applying AI for SEE, and Section 2.2 presents an example of how to run an AI approach for SEE.

2.1 AI Approaches for SEE

Many AI approaches have been investigated for SEE. They are typically machine learning approaches. Examples of machine learning approaches applied to waterfall-like software development processes include k-nearest neighbors [28, 29, 30], linear regression [31, 32], regression tree [33], linear programming [34] and ensembles of learning machines [35, 36, 37, 38]. Examples of machine learning approaches applied to agile software development processes are similar to that in the traditional development scenario, including decision tree, support vector machine, neural networks and ensemble of learning machines [20, 21]. There are also approaches making use of natural language processing and deep learning for effort estimation in agile software development [39, 40].

This section takes the approach proposed by Song et al. [5] as an example to explain the typical procedures of using machine learning techniques for SEE. To build an AI approach for SEE, one needs to extract data features describing previously completed software projects (or agile iterations) and their project team(s), and to collect the actual effort spent to fulfil these projects. This can form a training set, based on which an SEE model can be constructed, following a machine learning algorithm. After that, software project managers can use the SEE model to estimate the effort to develop a new software project (or agile iteration). Figure 9 illustrates the typical process of adopting machine learning approaches to solve a given problem. Each of the steps in this process is described in the context of SEE in the rest of this section. For an explanation of machine learning approaches in general, please refer to Chapter .

2.1.1 Collecting the Training Set

Features to Describe Software Projects or Agile Iterations

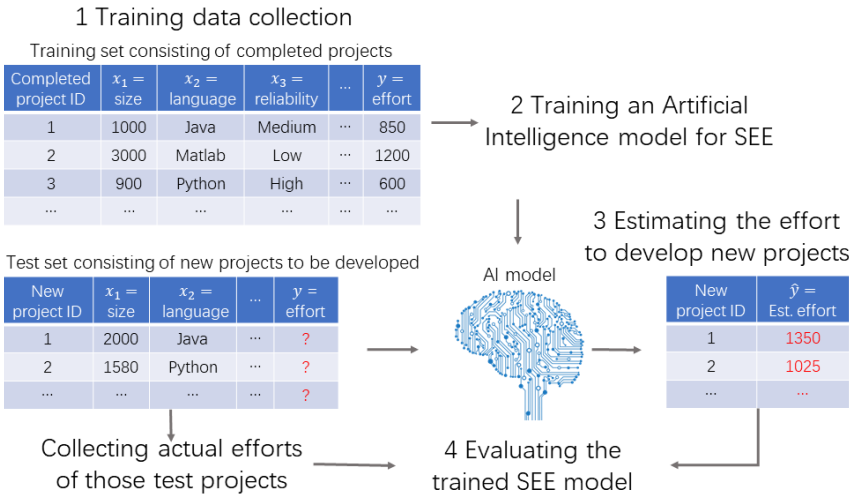

Fig. 9 Typical procedures of SEE from the AI viewpoint. Model adaptation is not illustrated here.

Table 6 COCOMO-format features of software projects in Nasa93 [41].

feature	description	type	value
loc	line of codes	numerical	continuous
dev	develop type	categorical	organic, embedded semidetached
rely	required software reliability	ordinal	1 = very low 2 = low 3 = normal 4 = high 5 = very high 6 = extra high
data	data base size	ordinal	
cplx	process complexity	ordinal	
time	time constraint for CPU	ordinal	
stor	main memory constraint	ordinal	
virt	machine volatility	ordinal	
turn	turnaround time	ordinal	
acap	analysts capability	ordinal	
aexp	application experience	ordinal	
pcap	programmers capability	ordinal	
vexp	virtual machine experience	ordinal	
lexp	language experience	ordinal	
modp	modern programming practices	ordinal	
tool	use of software tools	ordinal	
sced	schedule constraint	ordinal	

To use AI approaches for SEE, one needs to adopt some quantitative and qualitative metrics, as data features, to describe the software projects and their allocated development teams. Based on them, the effort required to develop the projects can be estimated. Table 6 lists the features of projects in Nasa93, a popular open-source data set for SEE in a waterfall-like development process [42]. It follows the famous COCOMO data format using characteristics such as the (estimated) size of the code, constraints that the software must satisfy, and information about the development

team allocated to the project [43]. Other features could also be adopted for SEE. For example, Kitchenham, another popular data set in SEE, uses features such as adjusted functional size, project type (A, C, D, P, Pr, U) and client code (C1, ..., C6) to describe software projects [44].

As pointed out in Usman et al.'s [20] and Marta et al.'s [21], there has been little consensus on the best features to describe software projects in different agile contexts. Nevertheless, development team skills (low, medium, high), size of the task to be estimated (a numerical value) and team's prior experience (with/without prior experience) are often highlighted as playing an important role in the estimation process in the agile context. Besides that, some work uses the textual description of user stories or issue reports as input features [39, 40].

Unit of Required Effort

To adopt AI techniques for SEE, one also needs to decide on the unit of the required effort. Required effort is typically a positive real value, measured in (e.g.) person-months. Alternatively, some agile SEE approaches measure the effort to develop user stories or issue reports in story points [39, 40].

Training Set

To build an SEE model based on machine learning, a training set containing examples of completed projects described by the input features and their actual required efforts is necessary. This is illustrated in Step 1 of Figure 9. Each project described by its input features and actual required effort is referred to as a training example. Typically, SEE data sets have from around 20 to 200 examples, though larger data sets would be desirable. Such data sets usually contain projects developed by the single company for which estimations are to be provided. This means that procedures need to be put in place for this company to collect features describing its software projects and the actual effort that the development team had spent on them. However, some specific SEE approaches have demonstrated success when adopting mixed data coming from different companies [45, 46, 47].

2.1.2 Building SEE Models Based on the Training Set

As shown in Step 2 of Figure 9, an SEE model can be constructed based on the training set by using a machine learning approach. Song et al. [5]'s study uses Relevance Vector Machines (RVMs) for this purpose, as they have shown good estimation accuracy compared to other approaches in the context of SEE [48]. An explanation of RVMs is provided in Section .

Typically, machine learning approaches for SEE are used to provide a point estimation, i.e., to estimate a single value representing the required effort to develop a software project [15, 49]. However, as uncertainty is inherent to SEE [50, 51], point estimates make it difficult to manage risks associated to SEE, potentially leading project managers to wrong decision-making. Song et al. [5] proposed a modification to RVMs enabling them to provide not only a point estimation, but also an interval of values, within which one would have a high confidence that the actual

Algorithm 2 Training algorithm for SynB-RVM.

Inputs: number of Bootstrap bags M, degree of synthetic displacement ρ.
Output: M trained RVM models and their training errors.

1: *Bootstrap training bag construction:* create M Bootstrap training bags from the training set using Bootstrap re-sampling with replacement.
2: **repeat**
3: **for** each Bootstrap training bag **do**
4: *Synthetic project generation:* replace the repeated training projects with their synthetic counterparts. The degree of the dispersion of the synthetic project from the original one is determined by the hyperparameter ρ.
5: *RVM training:* train the RVM model based on the revised Bootstrap training set.
6: Calculate training error of the RVM model according to some performance metric.
7: **end for**
8: **until** the maximum number of Bootstrap training bags is reached

required effort will fall. This allows for risk management, helping project managers to make better-informed decisions. For example, when bidding for a project, if the competition is very fierce the project manager can report a lower price within the interval to enhance the winning chances; when the competition is less fierce, he/she can propose a higher price. Moreover, such estimation can be a more reasonable representation of reality that embraces the fact that effort estimates are probabilistic assessments of a future condition [52].

Song et al. [5]'s approach showed competitive performance compared to others able to provide prediction intervals [5]. The rest of this section thus concentrates on explaining how Song et al. [5] extended RVMs to provide prediction intervals in addition to point estimates. If the reader is not interested in learning more about the techniques behind this extension, it is possible to jump to Section 2.1.3.

The extension of RVMs is a new approach named *Synthetic Bootstrap ensemble of Relevance Vector Machines (SynB-RVM)*. The basic idea of SynB-RVM is to create several SEE RVM models by training them on different samples of the training set. The predictions given by these SEE models can be aggregated to provide a point estimate corresponding to the most likely required effort value, and to create an interval of other possible values around it. The training algorithm of SynB-RVM is depicted in Algorithm 2, consisting of four steps: 1) Bootstrap training bag construction, 2) synthetic project displacement, 3) training RVMs and 4) calculating the training error for each RVM model. Each of these steps is explained below.

Bootstrap Training Bag construction

SynB-RVM first creates M different samples ('bags') of the training set based on a procedure called Bootstrap re-sampling with replacement. Each bag has the same size as the original training set, meaning that some training examples from the original training set will be missing and some will appear multiple times in the bag. The number of bags M is a hyperparameter that needs to be chosen beforehand. This can be done based on classical hyperparameter tuning techniques such as k-fold cross-validation. For a discussion of existing hyperparameter tuning techniques, please see Section .

Synthetic Project Displacement

Due to sampling with replacement, each Bootstrap training bag contains duplicated training examples, which would cause invertibility problem of the kernel matrix when training RVMs [53]. To this end, the training algorithm has a procedure to generate synthetic counterparts for these repeated training examples by displacing some of their features. The effectiveness of this displacement technique was verified in [5].

Figure 10 illustrates the mechanism of synthetic displacement where there is only one input feature and the output effort composing two dimensions of the vector space of projects. Given a duplicated project P_1, the synthetic project P_* is generated by displacing it along a different direction, influenced by a different project P_2 from the original training set. Specifically, the synthetic project is created based on the parallelogram formed in the vector space, as shown in this figure.

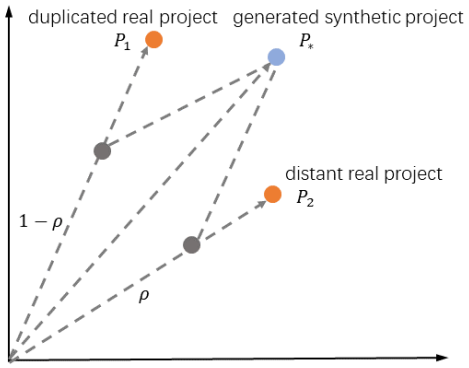

Fig. 10 Illustration of synthetic project generation for a duplicated real project P_1 in a 2-dimensional data space (one for the input feature and the other for the output effort). The synthetic project P_* is generated as a linear summation following *Parallelogram Law* in the 2-D vector space based on real projects P_1 and P_2 that are scaled according to the degree of synthetic displacement ρ.

Project P_2 is chosen as the most distant project from P_1 in the training set. The reason for choosing the furthest project are two-fold: 1) this leads to a more diverse Bootstrap bag which is more likely to relieve the invertibility problem and 2) disturbance along a real project will avoid the synthetic project to be too far away from the real one. For a project that has more than one replication, the learning algorithm will repeat this procedure multiple times. In this situation, rather than choosing the furthest project, the algorithm would choose the second or the third furthest project and so on.

RVM Training and Computation of Their Training Error

Based on the revised Bootstrap training bags, the training algorithm constructs M RVM models, each corresponding to one Bootstrap bag, according to RVM's learning algorithm explained in Section . Practically, these RVM models can be

constructed in parallel to speed up the training process. The training error of each RVM is also computed by comparing the actual and the estimated effort values across the training set. Each RVM has one particular training error, which is passed on to the prediction algorithm explained in Section 2.1.3.

2.1.3 Estimating the Effort for New Projects to Be Developed

After getting a trained SEE model, we can use it to estimate the effort required to develop new software projects, for which only input features are known. The aim is to predict their effort values. Figure 9 illustrates a test set containing a number of projects whose effort is to be estimated. Once the actual effort used for such projects becomes known, one can evaluate the predictive performance of the SEE model.

Given a new project to be developed, the project manager needs to first produce the values of the input features describing this project. These input features need to be the same features used for the training set. For instance, let's assume that these features are (code size=2000, language=Java, reliability = 'high', ...) as illustrated in Figure 9. Then, the project manager can feed these feature values to the SEE model that was built based on the procedure explained in Section 2.1.2, so that it provides an effort estimation for this project. The estimation will use the same unit (e.g., person-months) as defined at the data collection step.

Song et al.'s approach [5] provides a prediction interval associated to a confidence level. The center of this interval corresponds to the most likely effort value. For instance, the model may predict the interval [60, 100] person-months with a confidence level of 95%, together with the most likely required effort of 80 person-months. An approach that only provides point estimates would provide a single value (e.g., 80 person-months). The rest of this section concentrates on explaining how SynB-RVM produces both a point prediction and a prediction interval with a certain confidence level. If the reader is not interested in learning more about the machine learning technique behind SynB-RVM, it is possible to jump to Section 2.1.4. Algorithm 3 demonstrates the SynB-RVM procedure of producing an effort estimation for a new project to be developed [5]. Each step of this algorithm is explained below.

Multiple Probabilistic Prediction

The output of an RVM model when estimating the effort of a new project comes in the format of a Probability Density Function (PDF) of the possible effort values to develop the new project. Specifically, it follows a Gaussian distribution [53]. Figure 11 shows a probabilistic (Gaussian) effort estimation for a new project. The x-axis represents the effort and the y-axis represents the relative likelihood of observing different effort values. The corresponding point estimation is the mean of the Gaussian (1000 person-months), as it is the most likely effort value. When using SynB-RVM for effort estimation, we would get M predicted Gaussian PDFs, based on which the prediction interval with a certain confidence level is produced.

Model Pruning

Algorithm 3 Prediction algorithm of SynB-RVM.

Inputs: confidence level α, pruning rate $\tau \in [0, 1]$.
Output: prediction interval with respect to the given α.

1: **repeat**
2: Produce the values of input features describing test projects.
3: **for** each test project **do**
4: *Multiple probabilistic prediction:* employ the M RVM models constructed by the training algorithm to produce M Gaussian PDF predictions.
5: *Model pruning*: prune these RVMs with (a) negative estimated mean values and (b) unsatisfactory training performance according the pruning rate τ.
6: *Integrating the remaining estimates:* integrate the prediction of the remaining RVMs into a unified prediction result.
7: *Construct prediction interval in line with α:* derive the required prediction interval and the most likely point estimate based on the integrated prediction.
8: **end for**
9: **until** all test project are predicted

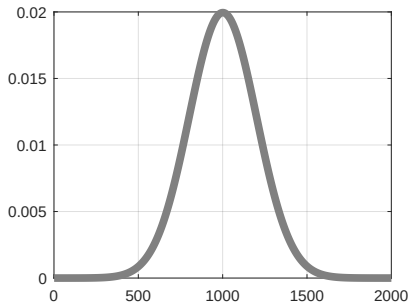

Fig. 11 Illustration of a Gaussian distribution from RVM as the estimations of the possible effort required to develop a software project. The y-axis represents the relative likelihood of effort values.

The mean values of the Gaussian PDFs produced by some RVMs may be improperly negative, whereas effort values cannot be negative. This happens because each RVM itself is a weak model, which may not perform so well. SynB-RVM will combine such weak models together to produce a strong, better model. However, it is prudent to eliminate RVMs that produce negative means when estimating a given project. The remaining RVMs are ranked based on their training predictive performance, and the worst τ percentage models are also pruned. This pruning prevents poor models from contributing towards the final estimation given by SynB-RVM.

Project managers can choose the pruning performance metric that best reflects their preference, based on the practical meaning of those metrics. Mean absolute error was recommended as the default performance metric for being symmetric and having no bias against over-/under-estimation [5].

Combining Uncertain Predictions

To generate the final estimation, we need to combine the PDFs provided by the remaining M' RVMs into a unified one. As Gaussian distribution is uniquely

determined by its mean and variance, this issue can be simplified into combining M' pairs of mean-variance. Song et al. [5] proposed three slightly different ways to derive the final probabilistic prediction on the test project, namely *empirical mean*, *uni-variate empirical PDFs* and *bi-variate empirical PDFs*. We discuss the version of applying *bi-variate empirical PDFs* in this book chapter as it can usually produce better prediction intervals when one has got an adequately large number of training projects. For a full description, please refer to [5].

To combine the PDFs based on the bi-variate empirical PDF method, the algorithm first creates a 2-dimensional frequency histogram $f(i, j)$ of the mean and standard deviation of the Gaussians outputted by all remaining RVM models. This can be implemented by applying MATLAB function *histcounts2()*, by which the numbers of bins are automatically determined by the binning algorithm to cover the data range and reveal the shape of the underlying distribution. Then, the geometric middle point $(y(i), \sigma(j))$ of the mean and standard deviations within each rectangle bin of the histogram is computed. The frequency $f(i, j)$ of each bin is also computed. Finally, the sum of the middle points multiplied by the frequencies is used as the mean y and standard deviation σ of the combined PDF, which is calculated as

$$(y, \sigma) = \sum\nolimits_{i,j} (y(i), \sigma(j)) \cdot f(i, j)$$

Construct Prediction Interval

Denote $\widehat{y}$ as the final estimated mean and $\widehat{\sigma}^2$ as the final estimated variance of the Gaussian PDF. Prediction intervals with respect to confidence levels of 68%, 95% and 99.7% can be easily derived as

$$[max(0, \widehat{y} - j\widehat{\sigma}), \widehat{y} + j\widehat{\sigma}],$$

according to the "68-95-99.7" rule of Gaussian distribution [54], where $j = 1, 2, 3$ corresponds to confidence levels 68%, 95% and 99.7%, respectively. For example, if the predicted final mean is 80 person-months, the most likely effort value is 80 person-months. If the variance is 5^2, the project manager would have 95% confidence level that the actual effort of the test project falls within [70, 90] person-months.

Deriving prediction intervals with respect to other confidence levels is a bit more complex. An explanation can be found in [5]. It is worth noting that the whole training and prediction process of machine learning algorithms can be automated, i.e., the derivation of the prediction intervals does not need to be done manually by the software project manager as will be shown in Section 2.2.

2.1.4 Evaluating the Predictive Performance of an SEE Model

Once software projects are completed, their actual required effort can become known, so long as the software developers working on the project are recording information on how much effort they have spent on the project. Projects that have not been used to train the predictive model can be used to evaluate this model once their

actual required efforts become known. Evaluating the model is important so that the software project manager has some idea of how good the estimations given by this model are expected to be. In particular, before starting to use a given model in practice, it is worth evaluating it on a number of completed projects that have not been used for training. This section discusses some popular performance metrics for evaluating models that produce point estimations and prediction intervals.

Metrics for Point Effort Estimation

There are several performance metrics for evaluation of point effort estimation. Popular examples are mean/median magnitude of relative error, mean/median absolute error, and percentage of estimates within $p\%$ of actual values.

Magnitude of Relative Error (MRE) [28, 55] measures the error ratio between an actual effort y_i and its corresponding estimated point effort $\widehat{y}_i$ as

$$MRE_i = |\widehat{y}_i - y_i|/y_i.$$

The smaller the MRE_i, the better the prediction performance of the SEE model performing on this project. For example, if an SEE method predicts that the development team will take 120 person-months to develop a project and it turns out that the team has spent 100 person-months in reality, MRE is $|120 - 100|/100 = 20\%$ meaning that this prediction is 20% away from the actual effort in magnitude.

When computing $\{MRE_i\}$ for several projects, one can compute the Mean MRE (MMRE) or Median MRE (MdMRE) to get an idea of the typical relative error of the model. The advantage of these metrics is that they are interpretable as a percentage. However, they can be biased towards predictive models that underestimate effort, i.e., a model that underestimates effort has a lower MMRE or MdMRE than another model that overestimates effort by the same amount [56, 55, 57, 58].

Absolute Error (AE) measures the difference in magnitude between an actual effort y_i and its estimated point effort $\widehat{y}_i$ as

$$AE_i = |\widehat{y}_i - y_i|.$$

The smaller the AE_i, the better the prediction performance of the SEE model performing on this project. For the same example having the predicted and actual efforts of 120 and 100 person-months, AE is $|120 - 100| = 20$ person-months, meaning that this prediction has a difference of 20 person-months from the actual effort.

When computing $\{AE_i\}$ for several projects, one can compute the Mean AE (MAE) or Median AE (MdAE) to get an idea of the typical absolute error of the model. MAE has been recommended by Shepperd and MacDonell for SEE studies, for being a symmetric metric and not biased towards under or overestimation [58]. For instance, "MAE=250 person-months" means that on average, the predicted effort would be larger / smaller than the actual value by a magnitude of 250 person-months. MdAE has shown to be less sensitive than MAE to occasional projects with very large efforts and is thus a useful addition to MAE [55]. But MdAE is less straightforward than MAE to be interpreted. One of the disadvantages of MAE and MdAE is that they cannot be interpreted as a percentage.

In summary, different performance metrics emphasize different factors and can behave differently in evaluating point effort estimation models [38]. It is highly unlikely to exist a single, simple-to-use and universal goodness-of-fit performance metric for SEE [55]. In practice, practitioners need to choose the performance metrics according to their particular emphasis and preferences.

Metrics for Uncertain Effort Estimation

Prediction intervals should be wide enough to capture the actual effort and at the same time sufficiently narrow to be informative for practical use. Therefore, metrics to evaluate the quality of prediction intervals need to take this into account. The following two metrics are typically used for that purpose.

Hit rate is the most common evaluation metric for prediction intervals [59, 60, 59]. The idea is as follows. If prediction intervals with confidence level α are evaluated based on N software projects, it is expected that around $\alpha \cdot N$ projects have actual efforts falling within the corresponding prediction intervals. Therefore, the hit rate can be calculated by first counting the number of projects whose estimated efforts are within the prediction intervals, and then dividing that by the total number of projects.

When the number of test examples is sufficiently large, the obtained hit rate should be around the chosen confidence level. When the hit rate is higher, the estimated prediction intervals are too wide; otherwise, the estimated prediction intervals are too narrow. For example, consider that there are plenty of test projects to evaluate an SEE model for a confidence level of 90%. If the hit rate is as low as 60%, this means that only 60%, rather than the desired 90%, of the actual efforts were within the prediction interval. This indicates that the model is not performing well for having too narrow prediction intervals or/and cannot achieve good point estimations. It is worth noting that due to the small SEE data sets, we usually do not have sufficient test examples. Hence, hit rate may deviate from its corresponding confidence level although the two values could be very close in essence.

Relative width is another useful evaluation metric for prediction intervals [61]. From two sets of prediction intervals with similar hit rates, the set with the narrower intervals is more informative. For example, given that two SEE methods can produce a similar hit rate of 90%, method A has an average relative width of 1.5 and method B has an average relative width of 2.2, method A can be considered better than method B for providing narrower (more informative) prediction intervals.

Given a prediction interval of a project, the relative width can be calculated by first computing the "width" of the prediction interval by subtracting the lower bound from the upper one, and then dividing that by the estimated point estimation that is the most likely to happen. For example, given a prediction interval of [500, 1000] person-months, if the most likely effort is 750 person-months, the relative width can be calculated as $(1000 - 500)/750 \approx 0.67$. This means that the width of the prediction interval was around 67% of the point estimation. This quantifies how informative the prediction interval is. One can compute the average relative width for the prediction intervals estimated for a given test set as a way to quantify the average amount of information of such prediction intervals.

It is important to note, though, that wider intervals may have better hit rates, whereas narrower intervals may have lower hit rates. Therefore, if two methods have different hit rates, the relative widths of their prediction intervals are not comparable. It is only possible to compare the relative widths of the intervals provided by two estimation approaches if their hit rates are similar.

2.1.5 Updating the SEE Model Based on New Incoming Data

Whenever a new project is completed and information on its actual effort is collected, a new example can be created. As mentioned in Section 2.1.4, these examples can be used to evaluate the predictive performance of the SEE model. Some approaches have also been proposed to further update this SEE model based on new examples, after such examples are used for evaluation purposes. In particular, the approaches by Minku et al. [45] have been proposed to adapt SEE models to changes suffered by software companies and that may affect such models.

SynB-RVM is a typical offline effort estimator. To be able to incorporate new training examples into it, it needs to be re-trained from scratch with a new training set that includes the new completed projects.

2.2 Running an AI Algorithm for SEE

This section gives an example of how to run an AI algorithm for SEE. Specifically, we employ SynB-RVM proposed by Song et al. [5] as an example to demonstrate how to produce effort estimations for software projects.

The tool is implemented in Matlab and is available at [62] under GNU GLP 3.0 license. Once the tool is downloaded, there is a folder called *matlab*, containing the codes implementing this tool. Another folder called *data_example* contains the edited software projects for SEE produced based on Nasa93 [42].

To use the tool, one needs to run the script *config.m* or typing in the command window the following line:

```
>> config()
```

This function configures the directories between code scripts and the data set and among all scripts, so that the scripts can call each other and load the data set as if they are in a single one-layer directory. An example of running the overall implementation of SynB-RVM is given in *example_run_SynB_RVM.m* in the directory "*matlab/examples/*". This example will be explained in the next subsections.

2.2.1 Preparing the Training and Test Projects

In the real-world, practitioners need to collect information of completed software projects (e.g., the features in Table 6) and the efforts spent to complete the development in their own organisations, making up the training set. The SEE model is then constructed based on this training set. Later on, when there comes a new project to be developed, the practitioner describes this project by the same features and applies the trained model to estimate the effort. This corresponds to the training and prediction processes of SEE as described in Sections 2.1.2 and 2.1.3.

Our tool was implemented for research purposes, making use of text files to collect information about the features describing software projects and their actual efforts. In our running example, we use nasa93, a data set from the open-source repository SEACRAFT [17] (`https://zenodo.org/record/268419#.YLRQyKgzYUE`), to demonstrate the whole process.

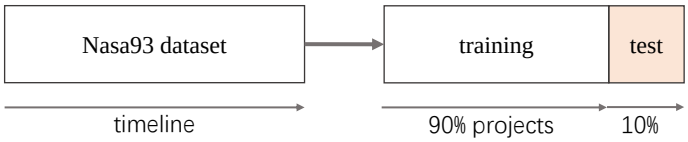

Fig. 12 Split Nasa93 into the training and test sets according to the project timeline, described as 'year of development' in the title file.

To simulate a real-world SEE scenario, the original Nasa93 is split into a training and a test set according to the time order of the software projects. As shown in Figure 12, 90% projects that completed earlier are used as the training data and the remaining 10% projects are used as the test examples to showcase point and interval predictions produced by SynB-RVM. The 17 features used to describe projects of Nasa93 are shown in Table 6 and are in the tool's folder `data_example/nasa93_edited`. The suffix `edited` means that the data set has been revised to be used with Matlab. For example, the feature 'stor' (main memory constraint) with the values of `vl`, `l`, `n`, `h`, `vh` and `xh` are converted to the values of 1, 2, 3, 4, 5 and 6, respectively.

To obtain the data division into training and test sets in one step, one can run *get_nasa93.m* or type in the command window the following line:

```
>> [X_train, y_train, X_test, y_test] = get_nasa93()
```

where `X_train` and `X_test` contain the input features of the training and test examples, respectively, and `y_train` and `y_test` contain their actual required efforts in person-months. Through *get_nasa93.m*, the training and test examples are pre-processed to prune out three outliers and to conduct feature normalisation. If one would like to run the pre-processing separately, this process is implemented in *data_preprocess.m* and can be run in the command window by typing the following line:

```
>> [years, X, y] = data_preprocess(Data);
```

where `Data` denotes the original projects of Nasa93, `years` denotes the time of completing each project, and `X` and `y` are their features and actual effort for development.

2.2.2 Using the Training and Prediction Algorithms of SynB-RVM

The training and prediction algorithms of SynB-RVM (Algorithms 2 and 3, respectively) are implemented in *SynB_RVM.m* in the directory "matlab/train-prediction/". Script *example_run_SynB_RVM.m* in the directory "matlab/examples/" exemplifies the usage of the training and prediction algorithms, producing point and interval predictions with a confidence level of 85% for the projects in the test set created in Section 2.2.1. Type the following command to run this script:

```
>> example_run_SynB_RVM()
```

The predicted point and interval prediction may improve or deteriorate with different hyperparameter settings. Section 2.2.5 shows how to tune hyperparameters. The hyperparameter setting of SynB-RVM adopted in this example is shown in the script *example_run_SynB_RVM.m* as

```
m_bags = 10;        %  #(Bootstrap Bags)
pru_m = 0.1;        % prune rate
tho = 0.01;         % synthetic displacement
```

This script prints the results below:

```
Prediction intervals with CL 0.85 of test projects in Nasa93 are:
   id    actual   point estimate   prediction interval    hit/not
    1    210.00       233.33        109.37 -   357.29         1
    2     48.00       133.33          1.38 -   265.29         1
    3     50.00        55.00          0.00 -   162.96         1
    4     60.00        65.00          0.00 -   172.96         1
    5     42.00       122.22          6.26 -   238.18         1
    6     60.00       144.44         28.48 -   260.41         1
    7    444.00       516.67        408.70 -   624.63         1
    8     42.00       144.44         28.48 -   260.41         1
    9    114.00       183.33         67.37 -   299.30         1
```

The second column presents actual efforts of test projects. These values would not have been available in practice at the prediction stage. They would become available only after the project finishes, if effort information is collected during the development process. The third column reports the point effort estimations. The fourth column reports the prediction intervals with a confidence level of 85%. The last column reports whether or not the prediction interval covers the actual effort for each test project. Again, this last column would not be available in practice at prediction time, only after the project is completed.

Let's take the seventh test project as an example of project being estimated. Its most probable point effort is 516.67 person-months. With 85% confidence, the actual effort falls within the interval of 408~625 person-months. This interval captures

reasonably well the actual required effort for this project. Some interval predictions catch the lowest bound of the effort value (0 person-month), such as the third and the fourth test projects. In such cases, this often indicates that the actual effort values has higher probability to be a smaller value rather than a larger value.

2.2.3 Evaluating the Performance of a SynB-RVM Model

In practice, completed projects with known effort that have not been used for training purposes can be used to evaluate SEE models. Such performance evaluation for the point estimations and interval predictions are implemented in *eval_point_pf.m* and *eval_PI_pf.m* in the directory "matlab/evaluate/", respectively. They will make use of the test set created in Section 2.2.1 for evaluation purposes. The evaluation of point predictions can be run with the following command:

```
>> eval_point_pf(y_true, y_pred)
```

The inputs `y_true` and `y_pred` are column vectors consisting of the true and predicted efforts of test projects, respectively. The estimated point efforts are obtained from the function *example_run_SynB_RVM()* as discussed in Section 2.2.2, and the actual efforts are collected when the projects finish and the effort spent on the development has been recorded. This function prints point prediction performance of SynB-RVM as shown below. Some of these performance metrics have been explained in Section 2.1.4. For a description of the other metrics, we refer the reader to [55].

```
ans = structures consisting of
       mae: 57.8214
      mdae: 82.3365
     mlgae: 3.5704
    mdlgae: 4.4108
      mmre: 0.9730
     mdmre: 0.8201
    pred25: 44.4444
    pred15: 44.4444
    pred10: 22.2222
      coor: 0.9536
       lsd: 0.6875
      rmse: 69.2337
     rmdse: 82.3365
        sa: 0.4656
```

The results mean that, the point estimations were around 57.82 person-months away from the actual required efforts on average (MAE), and that the point estimations were 97.30% away from the actual effort in magnitude on average (MMRE).

To evaluate the prediction intervals measured in *hit rate* and *relative width* as explained in Section 2.1.4, the following command can be used:

```
>> [hit_rate, relative_width] = eval_PI_pf(PI, y_pre_mean, y_true)
```

The input `PI` is a data matrix where each row represents the prediction interval of a test project and `y_pre_mean` is a column vector consisting of the point effort estimations. The output includes the information below:

```
hit_rate = 1
relative_width = 1.6957
```

The results mean that all prediction intervals covered the actual efforts, and that the widths of prediction intervals were around 170% of the value of the point estimates on average.

2.2.4 Using SynB-RVM in What-If Scenarios

Software managers could adopt AI SEE methods such as SynB-RVM to investigate what-if scenarios helping them to make more informed decisions. For example, assume a scenario where the project manager is considering whether to develop a software application that is as much space- and time-efficient as possible. However, they are unsure on whether this should be done, given the increase in effort that this could result in. As it is challenging to estimate the extent with which the effort would increase, artificial intelligence SEE models may be particularly helpful.

To investigate projects with higher computational and storage constraints, we can increase the values of the resource related features of test projects in Nasa93, including *data base size constraint*, *time constraint for cpu* and *main memory constraint*. For example, we can increase them to the highest value *xh* (extremely high) while retaining other features. The generation of such test projects is implemented in *get_nasa93_resource.m* in the directory "matlab/examples/", and can be run by typing the command below:

```
>> get_nasa93_resource('highest')
```

After that, we can observe SynB-RVM's effort estimations for these modified projects. The experiment is implemented in *experiment2_resource.m* in the same directory. One can run it by typing the following command:

```
>> experiment2_resource('highest')
```

where the input `highest` means that the highest resource constraint is evoked. Point and interval predictions of SynB-RVM are shown below:

```
=======================================
SynB-RVM_ht2d's prediction on the test projects in Nasa93 that
are reconstructed with the highest resource constraint.
Prediction intervals are with confidence level of 0.85.
   id predicted   prediction  interval
    1    805.56    689.59 -    921.52
    2    472.22    372.25 -    572.19
    3    516.67    424.70 -    608.64
    4    516.67    424.70 -    608.64
    5    638.89    522.93 -    754.85
    6    527.78    411.82 -    643.74
```

```
7    750.00    642.04 -    857.96
8    527.78    411.82 -    643.74
9    527.78    411.82 -    643.74
```

Comparing this against the previous estimations from *example_run_SynB_RVM.m*, SynB-RVM generally predicts that much higher efforts are required to develop a project with extremely high computational and storage constraints than to develop a similar project with less constraints. These estimations can help the software manager in deciding whether it is worth going ahead with such extreme constraints, given the increase in effort. Note that these projects have not been developed with such extreme constraints in practice. So, we cannot determine how accurate these estimations are.

If we assume another extreme scenario where the project manager is considering to have very light computational and storage constraints. In this scenario, we would take the lowest constraints on the database size, time and memory constraint features of Nasa93 projects. This can be implemented by running the function below:

```
>> get_nasa93_resource('lowest')
```

We can then use SynB-RVM to estimate the efforts required to develop those projects with the least computational and storage constraints, by running the command below:

```
>> experiment2_resource('lowest')
```

where the input `lowest` means that the lowest constraint of the development resource is evoked. The point and interval predictions of SynB-RVM are shown below:

```
=====================================
SynB-RVM_ht2d's prediction on the test projects in Nasa93
that are reconstructed with the lowest resource constraint.
Prediction intervals are with confidence level of 0.85.
   id predicted   prediction interval
    1    166.67     50.70 -    282.63
    2    128.57     10.32 -    246.82
    3     55.00      0.00 -    162.96
    4     65.00      0.00 -    172.96
    5    125.00      8.04 -    241.96
    6    144.44     36.48 -    252.41
    7    455.56    347.59 -    563.52
    8    125.00     17.04 -    232.96
    9    144.44     36.48 -    252.41
```

Comparing this against the previous estimations from *example_run_SynB_RVM.m*, SynB-RVM generally predicts that lower efforts are required to develop a project with light computational and storage constraints than to develop a similar project with higher constraints when the same development team is present. These estimations can help the project manager in deciding whether it is worth adopting lighter constraints.

Now assume a scenario where the project manager would like to give medium (nominal) computational and storage constraints. This can be achieved by replacing the resource related features of the projects in Nasa93 with the *nominal* value. We can implement this by running:

```
>> get_nasa93_resource('balanced')
```

We can then use SynB-RVM to estimate the efforts required to develop those projects with the balanced constraint by running the below command:

```
>> experiment2_resource('balanced')
```

where the input `balanced` means that a nominal constraints are evoked. Point and interval predictions of SynB-RVM are shown below:

```
=====================================
SynB-RVM_ht2d's prediction on the test projects in Nasa93
that are reconstructed with the balanced resource constraint.
Prediction intervals are with confidence level of 0.85.
   id predicted   prediction interval
    1    433.33    301.38 -    565.29
    2    188.89     72.93 -    304.85
    3    277.78    169.81 -    385.74
    4    300.00    192.04 -    407.96
    5    383.33    267.37 -    499.30
    6    283.33    167.37 -    399.30
    7    616.67    508.70 -    724.63
    8    300.00    192.04 -    407.96
    9    316.67    200.70 -    432.63
```

Comparing this against the estimations produced by *experiment2_resource('highest')* and *experiment2_resource('lowest')*, SynB-RVM generally predicts that the required efforts to develop a project with nominal resource constraint are smaller (larger) than that to develop a similar project with lowest (highest) constraint on the resources when the same development tea is present.

2.2.5 Choose Hyperparameter Values for SynB-SVM

To use SynB-RVM, one needs to pre-define three hyperparameters: the number of Bootstrap bags M, the degree of synthetic displacement ρ in the training algorithm (Algorithm 2), and the pruning rate τ in the prediction algorithm (Algorithm 3).

In the previous explanation on how to use the tool, we showed experimental results by running SynB-RVM with the default hyperparameter setting for Nasa93. Those values were decided based on our experience and preliminary experiments. In practice, practitioners may be unaware of the theoretical mechanisms behind the approach and find it hard to decide what hyperparameter values to adopt in order to obtain good effort estimations. This section aims to demonstrate how to tune the hyperparameters of SynB-RVM. Choosing the *kernel width*, a hyperparameter specific to RVM, is not included in this process to save computational resources. However, it can be tuned based on a similar procedure to the one shown below.

The hyperparameter tuning process is implemented in the directory */matlab/example-para_tune/*. Hyperparameter values investigated in *experiment_para_tune.m* are in Table 7. Their default values are emphasised in bold and correspond to the hyperparameters that can usually lead to good predictive performance according to the theoretical meaning of these hyperparameters and also to our experiences of using this approach. Hyperparameter settings are composed by enumerating all values for

each hyperparameter with all the others set to their default values. Thus, we have 9 hyperparameter settings in total. We believe that the values shown above form a good range of each of the hyperparameters. We run SynB-RVM with all the hyperparameter settings to calculate the performance on the validation set, which is a set of completed projects with known required efforts and that have not been used for training the predictive model. From that we can determine the best hyperparameter setting in terms of point prediction.

Table 7 Hyperparameter Values Investigated in *experiment_para_tune.m*.

Hyperparameter	Values	Description
M	**10**, 20, 30	the number of Bootstrap bags in the training algorithm
ρ	**0.01**, 0.05, 0.1	the degree of synthetic displacement in the training algorithm
τ	**0.1**, 0.2, 0.4	the pruning rate in the prediction algorithm

Given a combination of hyperparameter values, in this example, we will randomly select 90% of the projects in the training set to construct the SynB-RVM model, and the remaining 10% to compose the *validation* set on which this hyperparameter configuration will be evaluated. From that, we can evaluate the (point) predictive performance regarding this specific hyperparameter setting. This process is conducted ten times to cancel out the randomness of the process of splitting the training set into a training and a validation set. The average (point) predictive performance is used as the indicator of how good this hyperparameter configuration is. Finally, the hyperparameter configuration that leads to the best predictive performance on the validation set is selected. The ultimate SynB-RVM model to be adopted in the next prediction process is produced based on the entire training set with the chosen hyperparameter configuration.

To run the approach, the following command can be used:

```
>> experiment_para_tune()
```

The best hyperparameter configuration decided by this experiment, the point and interval predictions of SynB-RVM with the chosen hyperparameter setting and their predictive performance are shown below:

```
=======================================
The best hyperparameter setting based on this experiment is as below:
     the best number of Bootstrap bags is 30
     the best degree of synthetic displacement is 0.01
     the best pruning rate is 0.1

=======================================
Predictive performance of "Nasa93" with SynB_RVM_ht2d.
---------------------------------------
Overall predictive performance is as below:
     mae = 64.1
     hit_rate = 1.00
     relative width = 1.88
---------------------------------------
```

```
Prediction intervals of CL=0.85 of all test projects are as:
   id    actual  predicted   prediction interval     hit/not
    1    210.00     244.00    101.49 -    386.51                    1
    2     48.00     166.67     17.57 -    315.76                    1
    3     50.00      56.11      0.00 -    182.74                    1
    4     60.00      67.78      0.00 -    194.40                    1
    5     42.00     137.50      0.00 -    275.46                    1
    6     60.00     146.15      7.74 -    284.57                    1
    7    444.00     492.59    360.64 -    624.55                    1
    8     42.00     140.00      0.37 -    279.63                    1
    9    114.00     196.30     53.68 -    338.92                    1
```

Further Considerations

This section presents further considerations when adopting a SEE approach such as SynB-RVM:

1. The data quality of the organisation is the upmost important factor for producing a good software effort estimation. If the data is of low quality (e.g., large amounts of noise) and not representative for the current status of software development process (e.g., the data is very much obsolete), it is highly unlikely to attain good effort estimations regardless of how intelligent an approach could be.
2. SynB-RVM is suitable to be adopted for interval prediction only when it is found to achieve good point prediction performance. Otherwise, the prediction intervals provided by this approach would be less reliable. This is valid to most approaches that can provide uncertain effort estimation as the preciseness of the prediction intervals typically depend on the point prediction, being an interval around it.
3. Among the hyperparameters of SynB-RVM, the kernel width of RVM is typically the most important. If the computational resources for hyperparameter tuning are limited, the value for this hyperparameter can be decided independent of the others and can be processed beforehand, by adopting the same procedures discussed in this section.
4. A larger number of Bootstrap bags M probably leads to more computational cost for training the predictive model, as more base learners need to be learned (though they can be trained in parallel to save computational time).
5. The assignment of degree of pruning rate τ should be related to the number of Bootstrap bags M. Given a large M, one can assign τ to as big as 0.4 because there are potentially more RVMs to be pruned out. For a small M, practitioners are suggested to confine τ to be less than 0.2 for retaining enough base learners.

3 Conclusion

SPS and SEE are tasks that play important roles in the software project management process, for which AI can provide a useful tool to support project managers for better decision-making. This chapter provided an introduction to these tasks and to AI approaches that can be used to support software managers in carrying them out.

We explained how the SPS task can be formulated so that AI approaches can be used to solve it, based on the work of Alba and Chicano [2] and Shen et al. [4]. Alba and Chicano [2] provide a simpler formulation which provides a good platform to start learning about the subject. Shen et al. [4] provide a more detailed problem formulation that takes into account realistic aspects of software projects such as uncertainties in the task required efforts and changes such as new tasks or employee's leaves. We then introduced the algorithm proposed by Minku et al. [3] as an example of AI algorithm able to solve SPS based on Alba and Chicano [2]'s problem formulation. Shen et al. [4] proposes another AI algorithm able to cope with uncertainties and changes that may occur during software development, being a good next approach to learn after understanding the approach explained in this chapter.

We took the approach proposed by Song et al. [5] as an example to explain the typical procedures for using machine learning techniques for SEE, consisting of four procedures as 1) collecting training set, 2) building prediction models based on the training set, 3) estimating the effort for new project, and 4) evaluating predictive performance of the SEE model. One of the distinctive characteristics of Song et al. [5]'s approach is its ability to provide not only point estimates of required effort, but also a prediction interval, within which one would have a high confidence that the actual required effort will fall. This kind of approach can help software managers to make better-informed decisions. This approach can also be used to investigate the "what-if" scenarios as discussed in this chapter. When choosing an AI-based SEE approach to adopt in a software development company, we recommend evaluating the predictive performance of a number of different AI-based SEE approaches, to check which of them is better suited to the specific context of this company. For an overview of other AI-based SEE approaches, we refer the reader to [63].

References

1. I. Sommerville. *Software Engineering*. Pearson, USA, 10 edition, 2016.
2. E. Alba and J.F. Chicano. Software project management with GAs. *Information Sciences*, 177:2380–2401, 2007.
3. L.L. Minku, D. Sudholt, and X. Yao. Improved evolutionary algorithm design for the project scheduling problem based on runtime analysis. *IEEE Transactions on Software Engineering*, 40(1):83–102, 2014.
4. X. Shen, L.L. Minku, R. Bahsoon, and X. Yao. Dynamic software project scheduling through a proactive-rescheduling method. *IEEE Transactions on Software Engineering*, 42(7):658–686, 2016.
5. L. Song, L. L. Minku, and X. Yao. Software effort interval prediction via Bayesian inference and synthetic Bootstrap resampling. *ACM Transactions on Software Engineering Methodology*, 28(1):1–46, 2019.
6. M. Di Penta, M. Harman, and G. Antoniol. The use of search-based optimization techniques to schedule and staff software projects: An approach and an empirical study. *Software: Practice and Experience*, 41(5):495–519, 2011.
7. F. Ferrucci, M. Harman, and F. Sarro. Search-based software project management. In Günther Ruhe and Claes Wohlin, editors, *Software Project Management in a Changing World*, pages 373–399. Springer Berlin Heidelberg, 2014.
8. G. Antoniol, M. Di Penta, and M. Harman. A robust search-based approach to project management in the presence of abandonment, rework, error and uncertainty. In *International Symposium on the Software Metrics*, pages 172–183, 2004.
9. W.-N. Chen and J. Zhang. Ant colony optimization for software project scheduling and staffing with an event-based scheduler. *IEEE Transactions on Software Engineering*, 39(1):1–17, 2013.
10. F. P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison Wesley, Boston, 1995.
11. L. L. Minku. minkull/SoftwareProjectScheduling, 2022. URL: `https://doi.org/10.5281/zenodo.6308397`.
12. M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich. Opt4J - A Modular Framework for Meta-heuristic Optimization. In *Proceedings of the Genetic and Evolutionary Computing Conference (GECCO 2011)*, pages 1723–1730, Dublin, Ireland, 2011.
13. A. Trendowica and R. Jeffery. *Software Project Effort Estimation: Foundations and Best Practice Guidelines for Success*. Springer, 2014.
14. J. Wen, S. Li, Z. Lin, Y. Hu, and C. Huang. Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology (IST)*, 54(1):41–59, 2012.
15. K. Dejaeger, W. Verbeke, D. Martens, and B. Baesens. Data mining techniques for software effort estimation: A comparative study. *IEEE Transactions on Software Engineering (TSE)*, 38(2):375–397, 2012.
16. B. Baskeles, B. Turhan, and A. Bener. Software effort estimation using machine learning methods. In *International symposium on Computer and Information Sciences*, pages 1–6, 2007.
17. T. Menzies, R. Krishna, and D. Pryor. The SEACRAFT repository of empirical software engineering data. `https://zenodo.org/communities/seacraft`, 2017.
18. K. Maxwell. *Applied Statistics for Software Managers*. Prentice Hall PTR, 2002.
19. Pekka Abrahamsson, Raimund Moser, Witold Pedrycz, Alberto Sillitti, and Giancarlo Succi. Effort prediction in iterative software development processes – incremental versus global prediction models. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 344–353, 2007.
20. M. Usman, E. Mendes, F. Weidt, and R. Britto. Effort estimation in agile software development: A systematic literature review. In *International Conference on Predictive Models and Data Analysis in Software Engineering (PROMISE)*, pages 82–91, 2014.

21. M. Fernandez-Diego, E. R. Mendez, F. Gonzalez-Ladron-De-Guevara, S. Abrahao, and E. Insfran. An update on effort estimation in agile software development: A systematic literature review. *IEEE Access*, 8:166768–166800, 2020.
22. S. Keaveney and K. Conboy. Cost estimation in agile development projects. In *European Conference on Information Systems(ECIS)*, 2006.
23. N.C. Haugen. An empirical study of using planning poker for user story estimation. In *AGILE Conference*, pages 21–31, 2006.
24. Spareref. NASA to shut down checkout & launch control system. `http://bit.ly/eiYxlf`, August 2002.
25. M. Jørgensen. A review of studies on expert estimation of software development effort. *Journal of Systems and Software (JSS)*, 70(1):37 – 60, 2004.
26. M. Jørgensen. Forecasting of software development work effort: Evidence on expert judgement and formal models. *International Journal of Forecasting*, 23(3):449 – 462, 2007.
27. M. Jorgensen and S. Grimstad. The impact of irrelevant and misleading information on software development effort estimates: A randomized controlled field experiment. *IEEE Transactions on Software Engineering (TSE)*, 37(5):695–707, 2011.
28. M. Shepperd and C. Schofield. Estimating software project effort using analogies. *IEEE Transactions on Software Engineering (TSE)*, 23(12):736–743, 1997.
29. Y. Li, M. Xie, and T. Goh. A study of project selection and feature weighting for analogy based software cost estimation. *Journal of Systems and Software (JSS)*, 82(2):241 – 252, 2009.
30. E. Kocaguneli and T.J. Menzies. Exploiting the essential assumptions of analogy-based effort estimation. *IEEE Transactions on Software Engineering (TSE)*, 38(2):425–438, 2012.
31. P. A. Whigham, C. A. Owen, and S. G. Macdonell. A baseline model for software effort estimation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 24(3):20:1–20:11, 2015.
32. B. Kitchenham and E. Mendes. Why comparative effort prediction studies may be invalid. In *International Conference on Predictor Models in Software Engineering (PROMISE)*, pages 4:1–4:5, 2009.
33. C. Briand, E. Emam, D. Surmann, I. Wieczorek, and D. Maxwell. An assessment and comparison of common software cost estimation modelling techniques. In *ICSE*, pages 313–322, New York, USA, 1999.
34. F. Sarro and A. Petrozziello. Linear programming as a baseline for software effort estimation. *ACM TOSEM*, 27(3):12.1–12.28, 2018.
35. P. Braga, A. Oliveira, G. Ribeiro, and S. Meira. Bagging predictors for estimation of software project effort. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1595–1600, 2007.
36. L. L. Minku and X. Yao. Ensembles and locality: Insight on improving software effort estimation. *Information and Software Technology (IST)*, 55(8):1512–1528, 2012.
37. E. Kocaguneli, T. Menzies, and J. Keung. On the value of ensemble effort estimation. *IEEE Transactions on Software Engineering (TSE)*, 38:1403–1416, 2012.
38. L. L. Minku and X. Yao. Software effort estimation as a multi-objective learning problem. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 22, 2013.
39. R. G. F. Soares. Effort estimation via text classification and autoencoders. In *International Joint Conference on Neural Networks*, pages 1–8, 2018.
40. M. Choetkiertikul, H. K. Dam, T. Tran, T. Pham, A. Ghose, and T. Menzies. A deep learning model for estimating story points. *IEEE Transactions on Software Engineering*, 45(7):637–656, 2019.
41. B. W. Boehm. Software engineering economics. *IEEE Transactions on Software Engineering (TSE)*, 10(1):4–21, 1984.
42. Nasa93 doi. `https://doi.org/10.5281/zenodo.268419`, 2008.
43. Barry W. Boehm. *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs, NJ, 1981.
44. B. Kitchenham, S. L. Pfleeger, B. McColl, and S. Eagan. An empirical study of maintenance and development estimation accuracy. *Journal of Systems and Software (JSS)*, 64(1):57–77, 2002.

45. L. L. Minku and X. Yao. How to make best use of cross-company data in software effort estimation? In *International Conference on Software Engineering (ICSE)*, pages 446–456, New York, NY, USA, 2014.
46. L. L. Minku, F. Sarro, E. Mendes, and F. Ferrucci. How to make best use of cross-company data for web effort estimation? In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–10, 2015.
47. E. Kocaguneli, B. Cukic, T. Menzies, and H. Lu. Building a second opinion: Learning cross-company data. In *International Conference on Predictor Models in Software Engineering (PROMISE)*, Baltimore, USA, 2013.
48. L. Song, L. Minku, and X. Yao. The potential benefit of relevance vector machine to software effort estimation. In *International Conference on Predictor Models in Software Engineering (PROMISE)*, pages 52–61, Turin, Italy, 2014.
49. L. Song, L. Minku, and X. Yao. A novel automated approach for software effort estimation based on data augmentation. In *ACM Symposium on the Foundations of Software Engineering (FSE)*, Lake Buena Vista, Florida, USA, 2018.
50. B. Kitchenham, L. M. Pickard, S. Linkman, and P. W. Jones. Modeling software bidding risks. *IEEE Transactions on Software Engineering (TSE)*, 29(6):542–554, 2003.
51. M. Jorgensen. Realism in assessment of effort estimation uncertainty: It matters how you ask. *IEEE Transactions on Software Engineering (TSE)*, 30(4):209–217, 2004.
52. M. Klas, A. Trendowicz, A. Wickenkamp, J. Munch, N. Kikuchi, and Y. Ishigai. The use of simulation techniques for hybrid software cost estimation and risk analysis. In *Advances in computers*, volume 74 of *Software Development*, pages 115–174. Academic Press, 2008.
53. M. Tipping. Sparse Bayesian learning and the relevance vector machine. *Journal of Machine Learning*, 1:211–244, 2001.
54. B. Wlodzimierz. *The Normal Distribution: Characterizations with Applications*. Springer-Verlag, 1995.
55. T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit. A simulation study of the model evaluation criterion MMRE. *IEEE Transactions on Software Engineering (TSE)*, 29:985–995, 2003.
56. B. Kitchenham, L. Pickard, S. MacDonell, and M. Shepperd. What accuracy statistics really measure. *IEE Proceedings - Software Engineering*, 148(3):81–85, 2001.
57. I. Myrtveit, E. Stensrud, and M. Shepperd. Reliability and validity in comparative studies of software prediction models. *IEEE Transactions on Software Engineering (TSE)*, 31(5):380–391, 2005.
58. M. Shepperd and S. McDonell. Evaluating prediction systems in software project estimation. *Information and Software Technology (IST)*, 54:820–827, 2012.
59. M. Klas, A. Trendowizc, Y. Ishigai, and H. Nakao. Handling estimation uncertainty with bootstrapping: Empirical evaluation in the context of hybrid prediction methods. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 245–254, 2011.
60. M. Jørgensen. Evidence-based guidelines for assessment of software development cost uncertainty. *IEEE Transactions on Software Engineering (TSE)*, 32(11):942–954, 2005.
61. M. Jørgensen, K. H. Teigen, and K. Molokken. Better sure than safe? Overconfidence in judgement based software development effort prediction intervals. *Journal of Systems and Software (JSS)*, 70:79–93, 2004.
62. L. Song. sunnysong14/SoftwareEffortEstimation, 2022. URL: `https://github.com/sunnysong14/SoftwareEffortEstimation`.
63. A. Ali and C. Gravino. A systematic literature review of software effort predictionusing machine learning methods. *Journal of Software: Evolution and Process*, 31(10), 2019.