INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN, ICED'09
24 - 27 AUGUST 2009, STANFORD UNIVERSITY, STANFORD, CA, USA

IMPROVING EFFECTIVENESS OF AGILE DEVELOPMENT

Shelly Menachem, Yoram Reich
Tel Aviv University

ABSTRACT

Software projects are mostly based on human labor. This causes significant variance and uncertainty in software project planning as human labor is very hard to predict. This article represents a new approach to resource allocation in software development projects that will be demonstrated using software development methodology called "Agile." The reason we selected this methodology is because "Agile" is already considered to be the most effective way of project management in software [1]; therefore, if we can enhance resource allocation efficiently in Agile projects then it could be done in all software projects where Agile is applicable,

Agile software development has evolved in the past two decades into a set of tools for developing software. It is based on "all at once" models that assume that the creation of software is done by simultaneously working on requirements, analysis, design, coding, and testing, then delivering the entire system all at once. Agile main principle is using an iterative and incremental approach to building software. In Agile practice most of the responsibility is on the hands of the team members that need to estimate the work within iteration and commit to it. In order to accomplish this today, team members use their experience, educated advice, and some rules of thumb. We believe that especially in this stressed environment with almost "microscopic" work assignments, there should be a simple and fast mathematical model that assists them in this task. The model should handle dynamic environments and risky situations. This paper presents such a model, called HRA, and demonstrate in on a simple but representative real scenario.

Keywords: Project management, Agile development, Scrum, Software development, Stochastic simulation

1 INTRODUCTION

Agile software development has evolved in the past two decades into a set of tools for developing software. It is based on "all at once" models which assume that the creation of software is done by simultaneously working on requirements, analysis, design, coding, and testing, then delivering the entire system all at once [1, 4]. Agile main principle is using an *iterative* and incremental approach to building software [7, 11]. One particular *agile software project management* method called *Scrum* also gains popularity. Scrum structures product development in cycles of work called *Sprints*. Sprints are iterations of work that are typically 1 to 4 weeks long. The sprints are of fixed duration – they end on a specific date whether the work has been completed or not, and are never extended [2]. In this article, we will be using 2 week sprints, meaning 10 work days.

The scrum starts with a half day planning session, referred to as *Release Plan*, in which the desired features for the release are outlined. Then, this list of features is broken into X pieces that are achievable in one sprint. This is the *Product Backlog***.** The items in the product backlog can vary significantly in size; however, the larger ones will often be broken into smaller pieces during the *Sprint Planning Meeting*, and the smaller ones may be consolidated. One of the myths about Scrum is that it prevents from writing detailed specifications; in reality, it is up to the *Product Owner* and *Team* to decide just how much detail is required, and this may vary from one product backlog item to the next [2].

At the beginning of each sprint, the *Sprint Planning Meeting* takes place. In the first part of the sprint planning meeting, the *Scrum Team* reviews the product backlog, discussing the goals and context for the items on the product backlog. In the second part of the meeting, the Scrum team selects the items committed for completion by the end of the Sprint from the product backlog, starting at its top. In

other words, selection starts with the items that have the highest business value for the product owner (representing the customer). Each item is called *story card* or *card* in short. One software feature can be represented by one or many cards, but each card is managed separately and cards may spread over many sprints. Furthermore, cards could be broken into tasks. During the Sprint, the deliverables do not change [2, 7]. Every day during the Sprint, the team meets and discusses past days work, today's work, and obstacles. This allows all team members to be involved in each other's work and lend a hand to team members in trouble [1]. The team also updates simple charts that orient them to the work remaining and called *burndown chart* [2].

The team in Scrum is "cross-functional" – it includes all the expertise necessary to deliver a potentially shippable product each Sprint; and it is "self-managing," with a very high degree of autonomy and accountability.

One of the key practices in Scrum is that the team decides how much work they will commit to complete, rather than having it assigned to them by the product owner. The team will begin the Sprint planning meeting by estimating the time each member has for Sprint-related work [2]. This leads to a much more reliable commitment; first, because the team is making it, rather than having it "made" for them by someone else; and second, because the team itself is determining how much work will be required, rather than having someone else decides how much "should" be required [2]. The only rule the team is supposed to follow is keeping the priority of items the way the product owner has set them. The team does have the ability to pull in items from further down the product backlog if it makes sense (for example, pulling in a slightly lower priority item that can be quickly completed as part of higher priority work).

One of the most important base assumptions of agile development is that it is generally impossible to plan and estimate work that is not in the foreseeable future [3, 5, 10]. The *scrum master* is supposed to protect the team from any outside influence that will affect the deliverables and the work the team has committed to complete. Sometimes, it is impossible to protect the team, for example, when a team member is taken to participate in another high priority tasks, a shift in the sprint plan is unavoidable. Unforeseen technical difficulties, customer disapproving the outcome, or other reasons could also cause sprint plan changes. During the daily scrum meeting, the team is notified about the obstacle and works together to minimize its effect on sprint deliverables [1]. Sometimes the effect goes beyond one sprint and might compromise the whole release. It is the scrum master's duty to alert the product owner and the management about any features in the release that become at risk.

When an obstacle, also called *blocking issue*, occurs, there are some possible reactions that can be employed by the team, for example: (1) the team can push ahead tasks that were planned for later in the iteration very carefully as the business priority of each item is very important to the decision when to do it; (2) do some reshuffling between team members and the tasks they are working on; and (3) sometimes, the only possible reaction would be postponing a task to the next sprint [1, 2].

Beside obstacles that lead to wasting precious time, the opposite situation, where things progress better than anticipated also could lead to wasting time because when cards are about to be finished before schedule, the developer will usually start working slower in order to finish on-time. This is not done because the developers mean to harm the project, but because the only commitment is to finish on time, and as long as the commitment is met, no harm is done. This is actually "waterfall" behavior, when only the final commitment matters. We have observed that cards that were finished before schedule were a rare occasion. These two issues have to be addressed when improving the practice of agile methods.

There are many articles and studies about personal experience and agile implementation in software development teams that used to practice waterfall software project management methodology [e.g., 8, 9], another set of studies is about how to improve Agile implementation using software process improvement models [e.g., 10], much research has also been done on the subject of estimation, especially at the release plan and sprint plan phases [e.g., 10], there is also considerable amount of researches on how to improve time estimates in software projects in general which are inherently uncertain [e.g. 17,18, 19]. However, there are no studies on the following questions: (1) the team is supposed to estimate and commit; once they have committed, how do they decide who does what and when in the sprint scope? (2) How do they react to changes if enforced? (3) How can the team be sure they have made the most *effective* commitment?

Models that optimize allocation of software developers and development tasks were researched and suggested in previous studies [12, 14]. In those models, the developers are represented as the set of

their expertise and the task is a set of its characteristics, for example skill requirements. The biggest problem, in any allocation process with doers that are humans and therefore unique, is to separate the task from the doer, since a task can be assigned to any doer and not necessary the one that strikes as the most fit at first. This article suggests a different view; each developer has several important characteristics besides technical skills that affect an overall "*performance factor*" (PF). The set of developer skills is represented by "*fitness factor*" (FF). When the two factors are combined, they "normalize" the time estimate for a task for a certain developer; then optimization model is employed to create an optimized allocation. The last stage is simulated risk analysis to point out the pitfalls and possible delays.

The model proposed in this article can answer the question: who should do what and when in the scope of one sprint? The model is based on two types of inputs:

1. Tasks – comprised of the task time estimate done by the whole team (not a specific individual), and the skill-set per each task.
2. Team members and their characteristics – such characteristics include set-up-times, fitness to task (based on existing skills), level of motivation etc.; all characteristics are presented numerically.

The output of the model is all possible combinations of people and tasks during one sprint, based on linear programming models. It can also indicate, given the appropriate constraints, what is the most effective combination. Some risks analysis is also provided based on the estimated range of human characteristics (second input type).

The proposed model is demonstrated by a model of an actual agile scrum sprint that records the way the team reacted to a change and the resulting consequences. These results are compared to another reaction that they could have taken if they used the proposed model called the *Human Resource Allocation* (HRA) Model. It is important to emphasize that this model aids in decision making; it should not be treated blindly and it does not replace the need of the team to decide.

The remainder of this article is organized as follows. Section 2 describes the sprint and the event; Section 3 describes how the team reacted to the event and what happened. Section 4 demonstrates feeding event data to the HRA model and the results when using it. Section 5 discusses the stochastic simulations of the HRA model and Section 6 draws conclusions on the procedure and points out future research agenda.

2 DESCRIPTION OF THE SPRINT AND THE EVENT

The time frame of the example is one sprint, which is two weeks or 10 work days long. The unit of measurement for work time is one work day and its fractions.

The *cards* to be worked on in this sprint are selected according to their priorities and time estimation (see introduction section) during the sprint plan meeting; subsequently, they are broken into tasks. Acceptable time frame for a task is in the range of 0.5 days to 2 days of work [10]. The cards are displayed in Table 1.

Table 1: Cards of an example sprint

Story Card	Skill1	Skill2	Skill3	Skill4	Skill5	Time estimation (High risk, -20%)	Time estimation (Normal risk)	Time estimation (Low risk, +20%)
1				+	+	8	10	12
2	+			+	+	8	10	12
3	+	+	+	+	+	4	5	6
4			+		+	4	5	6
5		+	+		+	4	5	6
6		+			+	4	5	6
7	+	+			+	4	5	6
8	+		+	+	+	4	5	6

Each card requires several skills from the list of five skills defined for this project (e.g., Knowledge of a coding language such as Java; expertise in specific coding topics such as SQL queries; and knowledge of a specific development tool such as MSDN). The number of 5 skills is merely an example; for another project, a larger skill set might be required. We think that managing up to 10 skills is reasonable. Table 1 lists for each card the required skill-set and the time the team has

estimated it would take to complete this card with high, normal, and low risk. Some of the cards are dependent on other cards; the work on a card could begin only when its predecessor card has completed. Card 4 is dependent on the completion of card 3, card 6 on the completion of card 5 and card 8 on the completion of card 7. In the table, the dependent cards have been colored with the same color.[1]

The list of cards in Table 1 represents the team commitment for this sprint that was created during the sprint plan [1, 2, 3, and 4]. We do not describe how they reached this commitment; the team is supposed to sit in the room and stay there until they come up with such a sprint plan.[2] Task sequencing is not discussed here explicitly, although it is implicitly considered; the sequence between tasks must be taken into consideration in the optimization and simulation part for the HRA model.

Table 2 represents the burndown chart of the sprint. Each column represents a day. There are 5 team members. The table lists the developer(s) that is (are) expected to work on the particular card at that day, e.g., DEV4 in column 1 row 7, refers to developer number 4 that works on card 7 on the first day. Each day the finished work is erased from the chart, indicating how much work is left and if the team is working in the correct pace. From the table we see developers 1, 4, and 5, work each on 2 cards during this sprint and developers 2 and 3 picked longer cards that take the whole sprint to complete. QA (testers) are not included in this example for simplicity reasons. It is very common for a certain card to have both QA and development tasks. All tasks require commitment from team members whether they are development or QA.

Table 2: Planned card assignment

Story Card \ Day	1	2	3	4	5	6	7	8	9	10
1	DEV2	DEV2	DEV2	DEV2	DEV2	DEV2	DEV2	DEV2	DEV2	DEV2
2	DEV3	DEV3	DEV3	DEV3	DEV3	DEV3	DEV3	DEV3	DEV3	DEV3
3	DEV1	DEV1	DEV1	DEV1	DEV1					
4						DEV1	DEV1	DEV1	DEV1	DEV1
5	DEV5	DEV5	DEV5	DEV5	DEV5					
6						DEV5	DEV5	DEV5	DEV5	DEV5
7	DEV4	DEV4	DEV4	DEV4	DEV4					
8						DEV4	DEV4	DEV4	DEV4	DEV4

3. CHANGE HANDLING AND ITS CONSEQUENCES

After one week, all the work was done as estimated up to this point of time. The burndown chart at that point is represented in Table 3. At that point (6th working day, morning), developer 4 got sick and was supposed to stay home for 3 days. Since she did not start the work on card 8, 3 days of work got lost within the sprint.

It is important to note the difference between two scenarios, early and late delivery compared to planned commitment. When a developer ends his work earlier then the original estimate, she can pick another card from the project backlog [1, 2]. On the other hand, when a developer misses the estimated schedule, this might also affect future activities such as testing of the code since it is not possible to test what was not yet coded.

After it was confirmed that developer 4 has to stay home for 3 days, the scrum master announced in the scrum meeting that card 8 will be passed to the next sprint. Developer 4 will resume work on this card when she is back but it will not be finished in this sprint and not counted in the sprint completed tasks (agile does not accept partial completion).

[1] Such dependency could be modeled by e.g., DSM and the sequence of tasks could be ordered in some optimal manner. In the present study, such sequence is not created automatically and it is unclear that such automated sequencing would benefit the process in a noticeable way because the number of cards is quite small.

[2] This could be the subject of a future study. The sequencing all the tasks with their priorities might benefit the complete process; e.g., given a long list of cards/tasks, create batches of cards (similar to modular subsets) that have low dependency between the batches and high dependency between tasks in each batch. Some rationale could be given to an opposite strategy. An interesting topic would be to find the best strategy to do such subdivision.

Table 3: Story cards to-do after 1 week of work

Story Card \ Day	1	2	3	4	5	6	7	8	9	10
1						DEV2	DEV2	DEV2	DEV2	DEV2
2						DEV3	DEV3	DEV3	DEV3	DEV3
3										
4						DEV1	DEV1	DEV1	DEV1	DEV1
5										
6						DEV5	DEV5	DEV5	DEV5	DEV5
7										
8						DEV4	DEV4	DEV4	DEV4	DEV4

4. RESULTS WHEN IMPLEMENTING THE HRA MODEL

In order to address the aforementioned change systematically, we need to have better knowledge about the specific characteristics of team members. Table 4 shows several general (non-task related) characteristics of an individual like level of motivation, level of productivity, etc. Bringing all the qualities together using accumulation formula results in the *productivity factor* describing how productive is a certain person compared to others in performing each task. The grades for each factor were given by the direct manager/s. These grades are based on familiarity and experience and can be taken, for example from the periodical performance evaluations that are common practice in software companies. These factors may be correlated; for example, it is common that a person with high level of motivation is usually also very productive. However, in this model, the correction is not mandatory and a person that is highly motivated could also be working slowly. In this case, we may assume that his set-up time will be very high too.

The scale for each numeric characteristic is a *grade* from 1 to 5, where 1 means the best (high quality, fast work, etc.) and 5 means the worst (low quality, slow work, etc.), the scale is *relative* between the 5 developers in the team–it is a representation of a comparison of the team members to each other.[3] We treat this scale as a ratio scale, allowing performing some numeric calculations. For example, we define the *productivity factor* (PF) as the ratio between the group estimation to complete a card which is an "objective" estimate per card [4] and the actual time it will take a *certain* developer working on this card.

The formula used to calculate this factor is:

$$PF = (LOM/4 + LOP/4 + LOQ/4 + SUT/4) / 4 \quad (1)$$

We assume that each parameter has an equal effect on the productivity factor; therefore, the calculation is a simple average. With this data, we can explore different allocations than the team originally decided. While this equation is simplified, we believe it provides a reasonable representation of the productivity factor for specific a developer. Further work on this model including empirical corroboration is required to support it or its refinement.

Table 4: Characteristics of a team member (non task related qualities member). LOM: Level of Motivation; LOP: Level of Productivity; SUT: Set-up Times; LOE: Level of Education

Developer	LOM	LOP	LOQ	SUT	**PF**
1	1	1	3	1	**0.38**
2	3	3	5	4	**0.94**
3	3	5	3	3	**0.88**
4	3	4	5	3	**0.94**
5	4	2	4	2	**0.75**

In addition to the productivity factor, we must take into account the skill-set of each person. We do that by ranking up to 5 skills that are project related. Each developer is supposed to have at least one required skill for which she gets the rank of 1.20; each additional skill reduces 0.05 to the grade, down to 1.00 which represents having all the skill set for the project. This is also a simple calculation, which

[3] Such ordering could be created by using methods such as AHP to minimize the subjectivity involved.

we believe provides a reasonable representation of the fitness factor for a specific developer. Again, further work on this model including empirical corroboration is required to support it or its refinement. The skills that we check are the same 5 skills that were described in Table 1. The grade each developer gets indicates how fit from skill-set perspective she is for a certain task; this is the *fitness factor* (FF). In summary, the model steps are:

1. Compute *productivity factor* for all the developers based on the equation (1).
2. Computer *fitness factor* based on skills required and skills existing.
3. Multiply the result of steps 1 and 2 with the raw estimate done by the team in Table 1.
4. Find the most efficient path using optimization algorithms.

The result of steps 1 and 2 is represented in Table 6. For example, in order to get the first cell we have multiplied the time estimate for Task 1 with the productivity factor of DEV1 and fitness factor (combine task 1 and DEV1) like this: 10 (normal risk time estimate)*0.38 (PF)*1.10 (FF) = 4.13.

Table 5: Fitness factor for all developers and tasks

Task / Developer	1	2	3	4	5	6	7	8
1	1.10	1.10	1.10	1.10	1.20	1.20	1.00	1.00
2	1.05	1.05	1.05	1.05	1.20	1.15	1.05	1.05
3	1.00	1.00	1.00	1.00	1.20	1.15	1.10	1.10
4	1.00	1.05	1.20	1.20	1.00	1.00	1.20	1.20
5	1.10	1.10	1.00	1.00	1.20	1.20	1.00	1.00

Table 6: Time per each developer and task

Task / Developer	1	2	3	4	5	6	7	8
1	4.13	4.13	2.06	2.06	2.25	2.25	1.88	1.88
2	9.84	9.84	4.92	4.92	5.63	5.39	4.92	4.92
3	8.75	8.75	4.38	4.38	5.25	5.03	4.81	4.81
4	9.38	9.84	5.63	5.63	4.69	4.69	5.63	5.63
5	8.25	8.25	3.75	3.75	4.50	4.50	3.75	3.75

In order to find the most efficient scheduling, we model the task allocation with an optimization model. Z Denotes the total time required to complete the task. It is calculated based on the assignment of task j to developer i, X_{ij} multiplied by the effort of each developer for that task a_{ij} from Table 6, and summed over all allocations. In order to account for different risk levels, risk factor, X_0, is introduced. We can vary this parameter systematically and derive the highest value that meets certain constraints. This value will not be a probabilistic risk but more in the spirit of info-gap models of uncertainty [20]. The optimization model is:

$$\min Z = \sum_i \sum_j a_{ij} \cdot (1 + X_0) \cdot X_{ij} \quad (2)$$

s.t.

$\forall j, \sum_i X_{ij} = 1$ - *Each task can be preformed by one person only. This constraint also mandates that the task gets done in the present sprint.*

$\forall i, \sum_i X_{ij} \cdot a_{ij} \cdot (1 + X_{0j}) <= 10$ - *A developer has only 10 days to spend on all her tasks in the present sprint*

$\forall i = 1..5, j = 1..8, X_{ij} \in \{0,1\}$ - *The number of developers and tasks could vary.*

$a_{ij} \geq 0$ *(from table 6)*

The result of allocating developers to tasks for $X_0 = 0$ is presented in Table 7. This result is optimistic because it relies on the low risk estimations from Table 1: The total time spent on all the tasks is 31.62 days (out of possible 50). In this scenario, the absence of developer 4 could be resolved easily with the original plan by assigning her tasks to developer 2 who is not assigned a task in this sprint by the original plan.

If we use the optimization model we can prove that the plan fails at +14% off the estimates given in Table 1. This means that if the estimations would be equally wrong with more then 14% up across all tasks, the tasks will not fit into the sprint even if nothing out of the ordinary happens. Table 8 presents the last feasible allocation with 14% increase of the estimated time to complete tasks. With such increase, all developers are assigned tasks and one developer almost reaches maximum capacity.

Table 7: Task allocation and total time spent on all tasks – optimistic plan

Task / Developer	1	2	3	4	5	6	7	8	Total effort of developer(s)
1	0.00	1.00	1.00	0.00	0.00	0.00	1.00	1.00	9.94
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	8.75
4	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	4.69
5	0.00	0.00	0.00	1.00	0.00	1.00	0.00	0.00	8.25
	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	31.62

Table 8: Task allocation and total time spent for all tasks – pessimistic possible plan

Task / Developer	1	2	3	4	5	6	7	8	Total effort of developer(s)
1	0.00	1.00	0.00	0.00	0.00	0.00	1.00	1.00	8.98
2	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	6.15
3	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	9.98
4	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	5.34
5	0.00	0.00	1.00	1.00	0.00	0.00	0.00	0.00	8.55
	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	38.99

The optimization model could be easily executed for any number of risk factors. When we compare the model suggested in this article to the RQFD model by Reich and Levy [21], we can see that both models emphasize the importance of prompt reaction to changes in the internal and external environment of the project. This dynamic nature becomes the "way of life" in the software industry. Agile philosophies are based on this exact principle. In making the resource allocation decision, RQFD takes into account two factors and their combination: investment and quality, while the HRA model takes into account fitness and productivity factors. Fitness factor reflects quality; when we assign the fittest developer for a task, she will produce the highest quality possible. Productivity factor is the same as the investment in the RQFD model. When productivity levels are high, the investment is lower, since for the same fixed salary (in IT, the salaries are usually fixed), a person can produce a lot more. The main difference between the models is that RQFD stays general, testing different scenarios, while HRA claims that the quality and investment are inseparable of a specific person and therefore the optimized task assignment is crucial and it will lead to the result of a certain quality and investment. Future work could explore the integration of both models.

The output of the HRA model can be used in a simulation tool, shown in the next section, to produce very fast conclusions and analyze project weaknesses. All the above is done with very simple and easy to understand tools. We claim that a tool that is easy to understand is likely to be used.

The RQFD and HRA models differ in the way calculation is done. The HRA Model uses linear programming for resource allocation, while the RQFD model uses non-linear programming. Actually, the non-linear approach comes to play with the simulation and risk analysis part of the HRA model. The use of linear programming for human resource allocation is a common and well documented

practice giving fast primary results that then can be inserted into a non-linear simulation model for risk and analysis assessments.

5. SIMULATION OF HRA MODEL

In section 4, we described one possible scenario that showed how we can use the HRA model to meet better project goals. Nevertheless, this is not enough. In this section, we demonstrate a series of scenarios based on the HRA model.

Simulation is the most effective tool to work with in order to find weaknesses and calculate risk factors [13, 15, 22]. Also, it is possible to test model adjustment prior to employing the model in "real life". It is a relatively cheap way to make statistical calculations of the risks and finding the path that possesses the least risk of the project.

In the former sections, we treated the numbers as if they are deterministic. For example, we calculated the average time estimation from the estimates that were given by the team or the experts during iteration plan. One can argue that unknown factors, like time estimates bare a more stochastic nature [16]. We performed the simulation using the Arena (Rockwell Software) version 9. Arena allows modeling a process with its resources and then it simulates it to find the process characteristics (e.g., length) by assuming certain distributions over the process parameters. Figure 1 depicts the model, after the end of an iteration, 8 tasks were created and processed), waiting queue and working queues are empty, and the total effort spent in the 10-days sprint was 40 days.

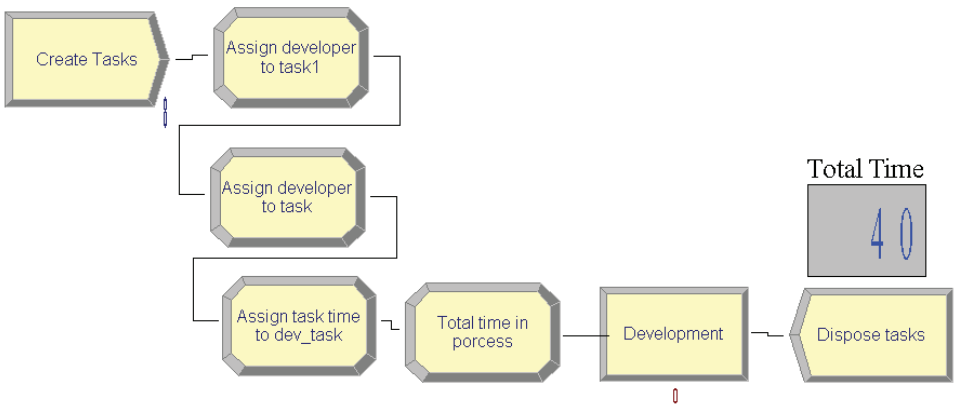

Figure 1: the HRA model set in simulation programs

The original estimation consisted of fixed determinate time estimations, but since the tasks are performed by people, even when the same person is doing the same task, unlike machines, she will not perform it in the same time. One of the common distributions to be used with human task time estimation is the Erlang distribution. We use it for simulation to get different actual times for the same estimated times.

The Erlang distribution is a probability distribution with wide applicability primarily due to its relation to the exponential and Gamma distributions. This distribution is often used to describe the distribution of waiting times in queuing systems. Also it has been used in describing the distribution of software development times [23], we use it for the same purpose, to manipulate normalized time estimations.

In the previous section, we discovered that the risk margin of the deterministic time estimation is 14%. When these work time estimations simultaneously exceed +14%, it is no longer possible to perform the allocation in a manner that all tasks are completed in one sprint. This however, is a pessimistic result since the estimations are all assumed wrong. When we go forward with the simulation, we assume a different approach towards the risk by assuming a certain probability distribution of the time estimations. This requires that we have sufficient data to model the distribution of the different input parameters. Rarely, do we have such sufficient data to make such modeling.

In order to bypass this difficulty, and still be able to assess the quality of an agile plan, we run the plan simulation twice: once with developer 4 fully working on the project and another when developer 4 is

absent for 3 days. In both cases, we run the simulations with the same variety of Erlang values. By comparing the results of these runs, we could check how these situations differ and obtain this difference in a variety of distributions over the parameters. Every simulation cycle consisted of 50 execution of the same sprint model. For different mean values, we are checking the percentage of full completion of the 8 tasks in the 50 runs. We calculate the percentage of full completion by dividing the average tasks out from 50 runs with 8, which is the 100% of tasks completed in all executions.
We can see that there is hardly a difference between the plan with developer 4 working all the time and between her absent for 3 days. Consequently, even with distribution of high variability, we still succeed in the original plan if it is planned in an optimal manner by the deterministic model in step one. This increases our confidence in the quality of the original plan.

Table 9: Simulation analysis results

mean value	average completed with all developers	average with dev4 partly absent	% of full completion with all developers	% of full completion with dev4 partly absent
0.80	8.00	8	100%	100%
0.85	7.96	7.96	100%	100%
0.90	7.88	7.88	99%	99%
0.95	7.82	7.82	98%	98%
1.00	7.78	7.78	97%	97%
1.05	7.74	7.74	97%	97%
1.10	7.64	7.64	96%	96%
1.15	7.58	7.42	95%	93%
1.20	7.50	7.39	94%	92%
1.25	7.40	7.31	93%	91%
1.30	7.28	7.26	91%	91%

6. CONCLUSIONS

Present practice of agile development is prone to failures due to ad hoc project planning and task allocation. With an intuitive and easy to implement mathematical model called HRA, product managers could improve their decision-making in various situations. The HRA Model allows creating possible and optimized work allocations. In creating the most efficient allocation, it leaves *built-in spare time* (buffers) to react to a crisis, or, if no crisis has occurred, the spare time is used to advance work from the project backlog.
We claim that the list of cards for a specific sprint should be picked up according to the raw estimate done in the sprint plan. This raw estimate tells the team how much workload they can take in a specific sprint. Only then, actual work allocation should be done using the HRA Model, as demonstrated in this paper.
Risk factor should also be calculated, to show how risky the plan actually is. From the deterministic linear programming model, we got a risk margin of 14%. This is low, definitely indicating that the plan is exposed to risks. When we added a stochastic simulation of the plan obtained by the linear programming model, we could see that in most cases, the plan worked equally well with developer 4 fully or partially working. This indicates that the use of a better model of the variability of estimations, lead to much less pessimistic results.
Further, we presented an example of a change occurrence: a developer that was allocated to work on the sprint was unavailable due to sickness. In such cases, a fast reaction is very important, but is mostly intuitive or *ad hoc*. A tool that is easy to use and fits well with agile practices will probably deliver better choices to address process problems.
Using the HRA model will provide better and faster response to new situations. All data about the developers should be entered into the model once at the beginning of the project. Subsequently, it could be used as many times as needed in order to get an instant calculation of what the new most efficient allocation is. It is suggested that using the proposed HRA model alongside all agile methods will improve the efficiency of software development projects.
In the future, we intend to test different situations with the simulation software to support agile development design decisions. We intend to calibrate the model and the equation in empirical studies

of agile or scrum teams. Subsequently, we intend to extend the model to handle complete projects including the task selection for each sprint and the change handling based on project rather than sprint concerns.
This article established the theoretical basis for implementing the method. The example used for demonstrating it is quite simple. In future work we intend to demonstrate the model including all its parts (allocation & simulation) on complex and large-scale projects.

REFERENCES

[1] Sutherland J., Agile development: lessons learned from the first scrum, Cutter Agile Project Management Advisory Service: Executive Update, 5(20):1-4, 2004.
[2] Deemer P., Benefield G., The scrum primer – an introduction to agile project management with scrum, http://www.scrumprimer.com, 2007.
[3] Tessem B., Maurer F., Job satisfaction and motivation in a large agile team, Agile Processes in Software Engineering and Extreme Programming, Lecture Notes in Computer Science, Volume 4536, Springer, Berlin, pp. 54-61, 2007.
[4] Abrahamsson P., Salo O., Ronkainen J., Warsta J., Agile Software Development Methods: Review and Analysis, VTT Publications 478 , 2002.
[5] Cohn M., Agile Estimating and Planning, Prentice Hall, 2005.
[6] ---, Agile Manifesto, http://www.agilemanifesto.org, 2001.
[7] Mahnic V., Drnovscek S., Agile Software Project Management with Scrum, University of Ljubljana, Faculty of Computer and Information Science, Ljubljana, Slovenia.
[8] Moore R., Reff K., Graham J., Hackerson B., Scrum at a Fortune 500 Manufacturing Company, Proceedings of AGILE 2007, 2007.
[9] Spence, J.W., There has to be a better way!, Proceedings of the Agile Development Conference, pp. 272- 278, 2005.
[10] Buglione L., Abran A., Improving estimations in agile projects: Issues and avenues, Proceedings of the 4th Software Measurement European Forum, pp. 265-274, 2007.
[11] Cao L., Ramesh B., Agile software development: Ad hoc practices or sound principles, IT Pro, 9(2):41-47, 2007.
[12] Duggan J., Byrne J., Lyons G.J., A task allocation optimizer for software construction, IEEE Software, 21(3):76-82, 2004.
[13] Huang S.-J., Han W.-M., Exploring the relationship between software project duration and risk exposure: A cluster analysis, Information Management, in press, 2008.
[14] Barretoa A., de O. Barros M., M.L. Werner C., Staffing a software project: A constraint satisfaction and optimization-based approach, Computers and Operations Research, 35(10):3073-3089, 2008.
[15] Aguilar-Ruiz J. S., Riquelme J. C., and Toro M., An evolutionary approach to estimating software development projects, Information and Software Technology, 43:875–882, 2001.
[16] Rao U.S., Kestur S., Pradham C., Stochastic optimization modeling and quantitative project management, IEEE Software, pp. 29-36, May-June 2008.
[17] Jørgensen M., Stein G., Avoiding Irrelevant and Misleading Information When Estimating Development Effort, IEEE Software, pp. 78-83, May-June 2008.
[18] Jurgensen M., Grimstad S., Over-optimism in software development projects: 'The winner's curse,' Proceedings Of the 15th IEEE International Conference on Electronics, Communications and Computers (CONIELECOMP), IEEE Computer Society Press, pp. 280–285, 2005.
[19] Hughes R.T., Expert judgment as an estimating method, Information and Software Technology, 38:67-75, 1996.
[20] Ben-Haim Y., Info-Gap Decision Theory: Decisions Under Severe Uncertainty, 2nd edition, Academic Press, London, 2006.
[21] Reich Y., Levy E, Managing product design quality under resource constraints, International Journal of Production Research, 42(13):2555–2572, 2004.
[22] Reich Y., Paz A., Managing product quality, risk, and resources through resource quality function deployment, Journal of Engineering Design, 19(3):249-267, 2008.
[23] Antoniol G., Cimitile A., Di Lucca G.A., Di Penta M., Assessing staffing needs for a software maintenance project through queuing simulation, IEEE Transactions on Software Engineering, 30(1):43-58, 2004.

Contact: Yoram Reich
Tel Aviv University
School of Mechanical Engineering
Tel Aviv 69978
Israel
+972-3-6407385
+972-3-6407617
Yoram@ENG.TAU.AC.IL
http://www.eng.tau.ac.il/~yoram

Shelly Menachem is a software engineer currently positioned as team leader of software projects in BMC Software, one of the world's largest software companies. She got a BA in industrial engineering and a MBA. She wants to combine he practical experience in management of software project with the theoretical knowledge from her studies and research ways to improve resource allocation in software projects.

Yoram Reich is a professor of mechanical engineering at Tel Aviv University. He received his PhD in applications of AI to design from Carnegie Mellon University. He develops design methods for early product development, knowledge management and information systems, as well as design theory and research methodology. He is Co-editor-in-chief of the journal Research in Engineering Design.