DOI: 10.15625/vap.2017.00038

ITERATION SCHEDULING USING BAYESIAN NETWORKS IN AGILE SOFTWARE DEVELOPMENT

Nguyen Ngoc Tuan, Huynh Quyet Thang

School of Information and Communication Technology, Hanoi University of Science and Technology

tuanmasu@gmail.com, thanghq@soict.hust.edu.vn

ABSTRACT: *In software industry nowadays, Agile Software Development methods have been largely adopted. Agile Software Development methods themselves can be considered a certain level of reducing projects risks. However, optimization of software project scheduling has always been big challenges in both practice and academia, since industrial software development is a highly complex and dynamic process. There is also a need for a probabilistic method that better model and predict uncertainty in software projects. This paper proposes an enhanced method and algorithm by combining optimized agile iteration scheduling and the ability to predict and handle risks in resource-constrained contexts of Bayesian Networks. Based on the method, a software was developed as a support tool for managers to control their project schedules. The tool also provides a reliable set of strategies of sequencing tasks in agile iteration scheduling.*

Keywords*: Agile Software Development, Project Risks, Iteration Planning, Iteration Scheduling, Bayesian Networks.*

I. INTRODUCTION

A. Overview

Traditional software development methods often characterized as predictive which focus on visioning and planning the future in full details. A predictive development team announces exactly what features are planned for the entire duration of the development process. Agile methods, in contrast, are adaptive. An adaptive team would have difficulty describing what features are planned for the entire duration of the development process, but they focus on adapting to changing realities quickly. When project changes occur then the team adapt themselves to the changes as well [1, 2, 3].

Agile methods break deliverables into small *iterations* (this would reduce overall risk of realization of software features [4, 5]). Iterations are short time frames (time boxes) that typically last from one to four weeks. Each iteration is a full software development cycle which includes planning, requirements analysis, design, coding, unit testing, and acceptance testing when a working software is demonstrated to users and/or customers. This minimizes overall risks and allows quick adaption to changes [6, 7, 8, 9].

Adopting Agile practices and processes brings certain benefits to organizations such as quicker return on investment, higher product quality, and better customer satisfaction [10]. However, they lack a sound methodological support of planning (contrary to the traditional plan-based approaches). VersionOne's survey [11] identified 26 principal factors and the second one was iteration planning. The survey also showed that three out of the five most important concerns (in the total of 13 most commonly cited greatest ones) about adopting agile within companies are 1) *the loss of management control* (36%), 2) *the lack of upfront planning* (33%) and 3) *the lack of predictability* (27%).

In addition, there have been some tools and techniques for project scheduling that project managers used (Fox & Spence [12], Pollack-Johnson [13]). Akos Szoke [14] presented a new approach for iteration scheduling in agile software projects. Vahid Khorakadami et al. [15] presented an improved approach to incorporate uncertainty using Bayesian Networks (BNs) in general project scheduling.

B. Optimized agile iteration scheduling

Scheduling problems constitute an important part of the combinatorial optimization problems. Scheduling concerns about the allocation of limited resources to tasks over time. The goal is the optimization of one or more objectives in a decision-making process [14]. Software project scheduling has to deal with the fact that resources such as human, time, technology and money are not always predetermined. Moreover, there are always risks (uncertain events which cause badly impacts) in software projects.

Technical tasks are the main concepts of iteration scheduling. These tasks are fundamental working units accomplished by developers. The aim of iteration scheduling is to break down selected requirements into technical tasks and to assign them to developers (and usually require some working hour realization effort that is estimated by the team). In other words, iteration scheduling aims at determining a feasible fine-grained plan for the development that schedules the implementation of selected features within an iteration [5].

Optimized (agile) iteration scheduling problem can be derived by selecting the extreme-valued schedule from the potentially feasible alternatives. This can be considered as an optimization problem in which the resource allocation

consists in assigning time intervals to the execution of the activities (realization tasks) while taking into consideration both temporal constraints (precedence between tasks) and resource constraints (resource availability) and the minimum execution time objective.

Although agile software development represents a major approach to software engineering, there is no well-established conceptual definition and sound methodological support of agile iteration scheduling.

II. OPTIMIZATION MODEL FOR AGILE SOFTWARE ITERATION

Let R be the set of resources and the following typical properties for scheduling be interpreted on technical tasks to schedule them:

Effort: w_j - time estimation (in hours) is associated with each task. It is calculated by simple expert estimation (e.g. 2, 4, or 8 working hour (Wh)).

Pre-assignment: a_j - in some cases resource pre-assignment is applied before scheduling. It is used by the scheduler algorithm during resource allocation.

Let the vector $S = \{S_0, S_1, \dots, S_{n+1}\}$ be start times for tasks' realizations - where $S_j \geq 0 : j \in A \; and \; S_0 = 0$. The vector S is called a *schedule* of development. In this definition, the 0 and n + 1 are auxiliary elements to represent iteration beginning and termination, respectively.

Dependencies between tasks *j* and *j'* can be defined as:

$$S_j - S_{j'} + d_{j'} \geq P_{j',j} \quad : j', j \in A \tag{1}$$

with S_j is the start time for the realization of task j, $S_{j'}$ is the start time for task j', $d_{j'}$ is duration time of task j', $P_{j',j}$ presented precedence tasks, and A is the set of tasks need to be done in the iteration.

Let the $R_i \in N$ is a set of capacities of resources that have been assigned to the project in an Agile iteration. The effort estimation yields resource requirements r_(j,i)∈Z for each task j and each resource i. Let S be some schedule and let t be some point in time. Then let $A^*(S,t) = \{j \in A | S_j \leq t \leq S_j + w_j\}$ be the active set of tasks being in progress at position t. The corresponding requirement for resource $i \in R$ at time t is given by $r_i(S,t) = \sum_{j \in A^*(S,t)} r_{j,i}$.. As a consequence, the resource constraints can be treated as follows:

$$r_i(S,t) \leq R_i \quad : \quad i \in R \tag{2}$$

Thus, optimization problem for iteration scheduling can be formulated as follows:

Minimize $z = S_{n+1}$
Subject to
$S_j - S_{j'} + d_{j'} \geq P_{j',j} \quad : \; j, j' \in A$
$r_i(S,t) \leq R_i \quad : \quad i \in R$
$S_{n+1} \leq l^I$
with l^I is the length of the iteration. (3)

III. SOLVING THE OPTIMIZATION PROBLEM FOR ITERATION SCHEDULING

The vector r indicates the available resources (developers) in the iteration. Each w_j is the planned effort (duration) for technical task j–both development and defect correction. Every element of vector a_j contains a reference to a resource index ($a_j \in \{1..|r|\}$) which indicates resources pre-assignment to task *j*. The $a_j = 0$ means that task j is not pre-assigned.

Thus the algorithm will find the best resource to its realization. Precedence between tasks can be represented by a precedence matrix where $P_{j,j'} = 1$ means that task *j* precedes task j', otherwise $P_{j,j'} = 0$. Both conditions $P_{j,j} = 0$ (no loop) and *P* is directed acyclic graph (DAG) ensures that temporal constraints are not trivially unsatistifiable. Iteration time-box is asserted by variable l^I. It is used as an upper bound in resource allocation to prevent resources overloading. The result of the algorithm is a schedule matrix S where rows represent resources, and columns give an order of task execution. Thus $S_{i,p} = j$ means that task *j* is assigned to resource *i* at the position *p*. The ensure section prescribes the postcondition on the return value (S): every task *j* has to be assigned to exactly one resource *i*.

The algorithm for iteration scheduling can be formulated as the following:

Algorithm 1: [14]

Input:

$r_i \in N, l^I \in N$ /* resources and length of each iteration*/

$a_j \in N : a_j \in \{1..|r|\}, w_j \in R$ /* Pre-assignment and duration of each task*/

$P_{j,j'} \in 0,1 \wedge P_{j,j} = 0 \wedge P\ is\ DAG$ /* precedence*/

Ensure:

$S_{i,j} \in 0,1 \wedge \forall j \exists! i S_{i,j} = 1$

$m \Leftarrow length(r)\ , n \Leftarrow length(d)$ /* number of resources and tasks */

$\boldsymbol{S} \Leftarrow [0]_{m,n}$ /* Initiate set of resources */

$rlist \Leftarrow \emptyset\ , slist \Leftarrow \emptyset$ /* Initiate 'ready list', 'scheduled list' */

for $j = 0\ to\ n$ **do**

  $pot \Leftarrow findNotPrecedentedTasks(P)$ /* Find potential task */

  $rlist \Leftarrow pot \setminus slist$ /* Construct ready list */

  if $rlist = \emptyset$ **then**

    return $\emptyset$ /* No schedulable task */

  endif

  $j \Leftarrow \max\{a_j\} : j \in rlist$ /* Select a task */

  if $a_j = 0$ **then**

    $i \Leftarrow selectMinLoadedResource(S)$ /* without assignment */

  else

    $i \Leftarrow a_j$ /* with assignment */

  endif

  $l \Leftarrow sum(S_{i,\{1..n\}})$ /*Examine load of resource i */

  if $(l + w_j) > l^I$ **then** /* overloaded iteration */

    return $\emptyset$

  endif

  $p \Leftarrow findNextPos(S, i)$ /* The next task */

  $S_{i,p} \Leftarrow j$ /* Assigned task j with resource i at position p */

  $slist \Leftarrow slist \cup \{j\}$ /* Add task j into slist */

  $P_{\{1..n\},j} = 0$ /* delete precedence related to scheduled task */

endfor

return S

IV. INCOOPERATING THE OPTIMIZATION PROBLEM WITH BAYESIAN NETWORKS

A. The need for a probabilistic model

The input of the above algorithm is defined which are resources, planned duration for each task, task precedence and the length of each iteration. It is assumed that given the resources, the task will be done in the planned duration. However, there are always risks in real projects such as those about personnel or technology. Those uncertainties can hardly be predicted, lead to the need of a probability model which can quantify the uncertain issues as well as addressing the most important concerns (cited in A. Overview). BNs is believed to be the good approach for modeling uncertainty in projects as well as helping project managers making decisions [15].

B. Bayesian Networks

Bayesian network (BN, or also be called as Bayesian Belief Network) models the causal relationships of a system or dataset and provides a graphical representation of this causal structure through the use of directed acyclic graphs (DAGs) with nodes and edges. The DAG representation provides a framework for inference and prediction. The nodes represent random variables with probability distributions, while edges represent weighted causal relationships between the nodes. Each node has a probability of having a certain value. A directed edge exists from a parent to a child. Each child node has a conditional probability table based on its parental values. Bayesian Network is based on Bayes' Theorem, and it is expressed in the basic form of Bayes rule as:

$$P(R|S) = \frac{P(S|R)P(R)}{P(S)} \quad (4)$$

The above Bayes rule is interpreted in terms of updating the belief (posterior probability of each possible state of a variable, that is, the state probabilities after considering all the available evidence) about a hypothesis R in the light of new evidence S. So the posterior belief P(R/S) is calculated by multiplying the prior belief P(R) by the likelihood P(S/R) that S will occur if R is true.

A BN consists of two parts: 1) *qualitative part* represents the relationships among variables by a directed acyclic graph, and 2) *quantitative part* specifies the probability distributions associated with every node of the model. The Figure 1 show a BN represents a simple case about the relationship between resources and duration in an iteration.

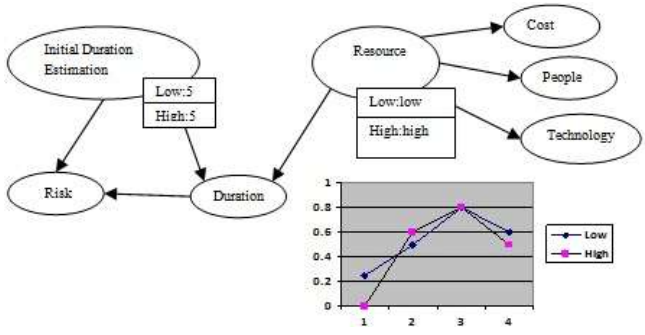

Figure 1. An example of Bayesian Network which represents a simple case.

C. Using BNs to enhance Agile iteration scheduling

The authors propose the following factors added to the above-mentioned algorithm:

a) A matrix which represents the relationship between each task duration and assigned resources:

$$[B]_{n+2,|r|} = \{b_{ij} \in [0,1] \,|\, i = 0..n+1, j = 1..|r|\} \quad (5)$$

i.e. the probability for task i to be done in the time w_i and allocated resource j is b_{ij}

b) When a schedule is created, its probability of completion is examined. An array should be imposed to represent the weights of tasks in an Agile iteration.

$$[M]_{n+2} = \{m_i | i = 0..n+1\} \quad (6)$$

Each task has an impact on the schedule for the iteration. Therefore we can have an array T of those impacts:

$$[T]_{n+2} = \{t_i | i = 0..n+1\} \quad (7)$$

In this research, we propose the following formula for t_i:

Let the set Di of resources allocated for task i in the iteration

$$t_i = \min\{b_{ij} | j \in D_i\} \quad (8)$$

The probability for the schedule is done successfully:

$$p = \frac{\sum_{i=0}^{n+1} t_i * m_i}{\sum_{i=0}^{n+1} m_i} \quad (9)$$

c) The algorithm with Bayesian Networks

The algorithm for calculating T:

Input: S, B
for $i = 0$ *to* $n+1$ **do**
  $t_i = 1$ /*Initiate the default t */
  for $j = 0$ *to* $|r|$ **do**
    if $\exists p \in P' S_{i,p}! = 0 \wedge b_{i,j} < t_i$ **then** /* Check if allocated */
      $t_i = b_{i,j}$ /* Update */
    endif
  endfor
endfor
return *T*

Thus, an enhanced algorithm using BNs – ***Algorithm 2*** is formulated as follows:

Algorithm 2:
Input:
$r_i \in N\ , l^I \in N$ /* Resources and length of each iteration*/
$a_j \in N:\ a_j \in \{1..|r|\}, w_j \in R$ /* Pre-assignments and durations of tasks*/
$P_{j,j'} \in 0,1\ \wedge P_{j,j} = 0\ \wedge P\ is\ DAG$ /* precedences */
$[B]_{n+2,|r|} = \{\ b_{i,j} \in [0,1]\ , i = 0..n+1, j = 1..|r|\}$ /* Matrix of completion probability */
$[M]_{n+2} = \{m_i | i = 0..n+1\}$ /* Weights of tasks in iteration */
$[T]_{n+2} = \{t_i | i = 0..n+1\}$ /*Impacts of tasks in iteration */
Ensure:
$S_{i,j} \in 0,1 \wedge \forall j \exists!\, i S_{i,j} = 1$
$m \Leftarrow length(r)\ , n \Leftarrow length(d)$ /* Number of resources and tasks */
$\boldsymbol{S} \Leftarrow [0]_{m,n}$ /* Initial set of resources */
$rlist \Leftarrow \emptyset\ , slist \Leftarrow \emptyset\ , P' \Leftarrow \emptyset$ /* Initiate 'ready list', 'scheduled list' and P' list */
for $j = 0\ to\ n$ **do**
$pot \Leftarrow findNotPrecedentedTasks(P)$ /* Find potential task */
$rlist \Leftarrow pot \setminus slist$ /* create ready list */
if $rlist = \emptyset$ **then**
return $\emptyset$ /* No schedulable task */
endif
$j \Leftarrow \max\{a_j\} : j \in rlist$ /* select a task */
if $a_j = 0$ **then**
$i \Leftarrow selectMinLoadedResourceandMaxPro(S)$ /* without assignment */
else
$i \Leftarrow a_j$ /* with assignment */
endif
$l \Leftarrow sum(S_{i,\{1..n\}})$ /* calculate load of resource i */
if $(l + w_j) > l^I$ **then** /* overloaded iteration */
return $\emptyset$
endif
$p \Leftarrow findNextPos(S, i)$ /* the next task */
$P' \Leftarrow P' \cup \{p\}$ /* add the index of the next task into P' */
$S_{i,p} \Leftarrow j$ /* assigned task j with resource i at position p */
$slist \Leftarrow slist \cup \{j\}$ /* add task j into slist */
$P_{\{1..n\},j} = 0$ /* delete precedence related to scheduled task */
endfor
$T \Leftarrow computingfrom(S, B)$ /*calculating matrix T from B */
$x = computingProFrom(S, T, M)$ /* Calculating x from S, T, M */
return S, x, P'

Where:

- $findNotPrecedentedTasks$ - Find tasks without priority constraints based on matrix of task priority P
- $selectMinLoadedResource(S)$- Select the resource with the minimum load and the highest probability of success (with the criteria of maximum load and probability).

Output: The set S of schedules set of time for tasks, and the probability of successful S.

V. TOOL AND EXPERIMENTAL RESULTS

A. Building tool

To elaborate the proposed model and algorithm, we built the tool BAIS (Bayesian Agile Iteration Scheduling) using Java programming language. The tool allows users to enter the number of resources (developers), the number of tasks, the length of iterations, tasks' precedences, pre-assignments and durations for tasks. The tool also requires the input for a table of probability for each resource finishes the assigned task in time (Figure 2).

BAIS implements four strategies for selecting tasks in scheduling (or scheduling rules):

- SPT (Shortest processing time first): sequences the tasks in increasing order of their processing time.
- LPT (Longest processing time first): sequences the tasks in decreasing order of their processing time.
- AF (Assigned First): sequences the tasks based on team pre-assignments.
- AF+LPT: the combination of AF and LPT.

Figure 2. Home GUI of tool BAIS

B. Experimental results and analysis

The authors use two data samples in two experiments:

a) The first sample is a randomly generated one. Given an iteration with two resources and 20 working-hour length (unbound). Its number of tasks is 8 and there is only one precedence that is the 5th task need to be done before the 3rd one. Table 1 shows T_1, T_2 are probability tables of resource 1 and resource 2 successfully done their tasks (i.e., finish in time as scheduled) if the previous tasks were done in time, and D_1, D_2 are their probability tables of finishing their tasks in time if the previous tasks were over schedule.

Table 1. The first data sample

Task	Time	Pre-assignment	T_1	T_2	D_1	D_2
1	4	-	0.92	0.83	0.38	0.34
2	3	-	0.74	0.72	0.27	0.22
3	5	-	0.93	0.85	0.22	0.34
4	4	2	0.94	0.84	0.43	0.36
5	2	-	0.83	0.73	0.26	0.42
6	5	-	0.82	0.96	0.17	0.23
7	4	-	0.73	0.94	0.32	0.52
8	3	-	0.68	0.73	0.27	0.13

The results of four strategies of selecting tasks SPT, LPT, AF, and AF+LPT:

- SPT: the shortest time that all resources finish in an iteration – makespan - is16 (hours). Resource 1 is assigned the tasks 5, 2, 7, 3 and Resource 2 is assigned the tasks 8, 1, 4, 6 respectively. The probability for success is 68,83%.
- LPT: makespan is 17. Resource 1 is assigned the tasks 6, 7, 8, 3 and Resource 2 is assigned the tasks 1, 4, 2, 5 respectively. The probability for success is 60,42%.

- AF: makespan is 15. Resource 1 is assigned the tasks 1, 2, 6, 8 and Resource 2 is assigned the tasks 4, 5, 3, 7 respectively. The probability for success is 66,77%.
- AF+LPT: makespan is 17. Resource 1 is assigned the tasks 6, 7, 8, 3 and Resource 2 is assigned the tasks 4, 1, 2, 5 respectively. The probability for success is 60,37%.

According to the above result, if the team considers minimizing makespan is the first optimized criteria, then the AF strategy should be chosen. If we want the highest probability of success, then we take the SPT strategy. Figure 3 shows the Gantt chart yielded by BAIS for the SPT strategy.

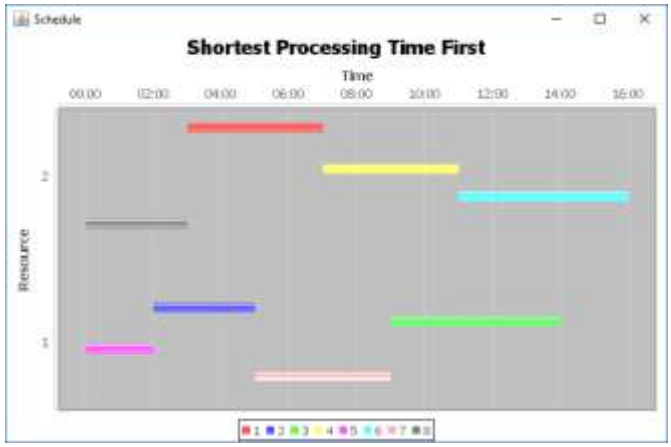

Figure 3. Gantt chart for SPT strategy

b) The second experiment was carried out with a real-life project data from the Company A. There are 3 developers who need to finish 15 tasks in the project. Each iteration is carried out in 40 hours. The authors asked the project manager and experts in the company to provide the pre-estimated probability of finishing the tasks based on each developer's experience. The calculation from the tool BAIS gives us the probability table shown in Table 2.

Table 2. The probability table for tasks and resources

Taks	Time	Probability					
		Previous tasks in time			Previous tasks over schedule		
		Re. 1	Re. 2	Re. 3	Re. 1	Re. 2	Re. 3
1	6	1.00	0.96	0.65	0.88	0.8	0.49
2	9	1.00	1.00	0.9	0.96	0.85	0.68
3	5	1.00	1.00	0.87	0.7	0.81	0.65
4	6	1.00	1.00	0.83	0.88	0.74	0.75
5	9	1.00	1.00	0.89	0.9	0.77	0.66
6	6	1.00	1.00	0.96	0.98	0.9	0.82
7	2	1.00	1.00	0.8	0.85	0.75	0.75
8	6	1.00	1.00	0.77	0.95	0.75	0.68
9	6	1.00	1.00	0.75	0.96	0.9	0.75
10	4	1.00	1.00	0.89	0.85	0.72	0.67
11	8	1.00	1.00	0.89	0.8	0.9	0.67
12	7	1.00	0.94	0.75	0.9	0.74	0.68
13	4	1.00	1.00	0.82	0.84	0.75	0.62
14	3	1.00	0.86	0.75	1.00	0.84	0.67
15	4	1.00	1.00	0.76	0.83	0.9	0.67

The results for makespans and overall probability: SPT: makespan is 30, the probability for success is 97,98%; LPT and AF+LPT: makespan is 28, the probability for success is 46.98 %; AF: makespan is also 28, and the probability for success is 88,89%. According to these results, the project manager should really pay more attention on the tasks assignments and adjust the iterations' times during the project.

The results of both experiments show that the tool can support decisions in agile software development planning to tailor the best plan for the specific project context and users' and/or customers' feedbacks by altering constraints,

capacities and priorities. In a single project, the manager can also use the tool to predict next phases or next iterations schedule and better understand how the failure of the phase can impact the whole project.

VI. CONCLUSION

The research has developed an algorithm for agile iteration scheduling with the cooperation of Bayes Networks to support software teams to analyze the schedules as well as predict the chance of their success. The method improves the quality of agile software development planning to provide lower level risks by considering all major planning factors (e.g. dependencies, capacities) in a mathematical optimization model.

The results of experiments on the available data sets and indicate that the approach can provide practical value as a decision support tool for agile iteration planning. To further affirm this, more representative real-life data sets needed and some case studies can be carried out.

This research can be considered as a step towards a conceptual model of agile iteration planning and scheduling. Since the research gives better insight into resource-constrained project scheduling problems, this may suggest a new optimization problem on agile iteration scheduling.

The developed tool provides options for scheduling rules which enable us to compute an optimal active schedule for the singular resource or overall project. An upgrade can be further developed which incorporates BNs for representing and analyzing causal models involving uncertainty. The version can even provide a set of tools for constructing probabilistic inference and decision support systems on BNs and thus can assist software project managers in making decisions in scheduling and planning all kinds of software projects.

Acknowledgment: The authors express gratitude thanks to Ms. Bui Thi Quynh Nga (5th year student at School of ICT, HUST) for her valuable help in the development of the tool BAIS.

REFERENCES

[1] Tore Dyba and Torgeir Dingsøyr, “Empirical studies of agile software development: A systematic review”, Information and Software Technology 50.9-10, pp. 833–859, ISSN: 0950-5849, 2008.

[2] P. Abrahamsson et al., Agile software development methods - Review and analysis, Technical report 478, *VTT PUBLICATIONS*, 2002.

[3] Sanjiv Augustine, “Managing Agile Projects”, Upper Saddle River, NJ, USA: Prentice Hall PTR, ISBN: 0131240714, 2005.

[4] Tsun Chow and Dac-Buu Cao, “A survey study of critical success factors in agile software projects”, Journal of System and Software 81.6, pp. 961–971. ISSN: 0164-1212, 2008.

[5] Mike Cohn, Agile Estimating and Planning, NJ, USA: Prentice Hall PTR, ISBN: 0131479415, 2005.

[6] Ken Schwaber and Mike Beedle, Agile Software Development with Scrum, 2001.

[7] Robert C. Martin, Agile Software Development, Principles, Patterns and Practices, 2002.

[8] Ade Miller, Distributed Agile Development at Microsoft patterns and practices, 2008.

[9] Agile Alliance, Manifesto for agile software development, [Online] Retrieved 14 May 2017. Available at: http://agilemanifesto.org

[10] Nguyen Ngoc Tuan and Huynh Quyet Thang, “Combining Maturity and Agility – Lessons Learnt From A Case Study”, Proceedings of the 4th International Symposium on ICT SoICT 2013: 267-274, 2013.

[11] VersionOne, 7th Annual Survey: 2012, The State of Agile Development, Full Data Report, 2013.

[12] Fox T. L. and Spence J. W., “Tools of the trade: a survey of project management tools”, Project Management Journal, 29, 20-28, 1998.

[13] Pollack-Johnson, “Project management software usage patterns and suggested research directions for future development”, Project Management Journal, 29, 19-29, 1998.

[14] Akos Szoke, Models and Algorithms for Integrated Agile Software Planning and Scheduling, PhD Dissertation, 2014.

[15] Vahid Khorakadami, Norman Fenton and Martin Neil, Improved Approach to Incorporate Uncertainty Using Bayesian Networks, Queen Mary University of London, United Kingdom, 2009.

LẬP LỊCH BƯỚC LẶP SỬ DỤNG MẠNG BAYES TRONG DỰ ÁN PHÁT TRIỂN PHẦN MỀM LINH HOẠT

Nguyễn Ngọc Tuấn, Huỳnh Quyết Thắng

TÓM TẮT: *Các phương pháp phát triển phần mềm linh hoạt (Agile Software Development) đã được sử dụng rộng rãi trong gần 20 năm qua. Các phương pháp này về mặt triết lý đã hướng vào việc giảm thiểu các rủi ro trong dự án phần mềm. Tuy nhiên trong thực tế công nghiệp phần mềm đầy phức tạp và biến động thì tối ưu hóa lập lịch dự án luôn là một thách thức lớn. Ngoài ra cũng có nhu cầu lớn về phương pháp định lượng để mô hình hóa, phân tích và dự đoán rủi ro trong các dự án phần mềm. Bài báo này cải tiến các phương pháp và giải thuật hiện có trong việc tích hợp lập lịch bước lặp tối ưu với khả năng dự đoán rủi ro của Mạng Bayes trong môi trường nhiều ràng buộc nguồn lực. Nhóm tác giả phát triển được thuật toán lập lịch bước lặp sử dụng mạng Bayes để hỗ trợ đưa ra lịch trình và đánh giá xác suất hoàn thành lịch trình, đồng thời đưa ra chiến lược lựa chọn nhiệm vụ trong lập lịch bước lặp mới để làm giảm tổng thời gian thực hiện bước lặp (makespan) trong ràng buộc về yêu cầu phân bổ nguồn lực trước đó.*