Volume 2, Issue 9, September 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Resource Oriented Intelligent Scheduling Scheme to Estimate Software Release

Prabhat Gaur*
Computer Department, MMEC Mullana
Haryana, India

Ashish Oberoi
Computer Department, MMEC Mullana
Haryana, India

Abstract— **Software Release Management is an important key technology for distributing the project/product to the customer. The key success factor of any Software Product lies in how delicately the product is released to the customer. "Release management is the process of planning, building, testing and deploying hardware and software, the version control and storage of software. There are number of approaches to estimate the software development estimation and the release planning. But there is no such approach that collects all the resources and capabilities in one shot and perform the intelligent process on it. We provide such an intelligent scheduling approach to estimate the software release by taking the inputs as initial requirement, availability and the capabilities of the resources. It will accept this data as the training set and perform decision making by analyzing the requirement and the availability of human and non human resources.**

Keywords— Software Release Plan, Agile Development, Genetic, offspring, Optimization

I. INTRODUCTION

Software engineering (SE) is the application of a systematic, disciplined, quantifiable approach to the design, development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software. Software development, a much used and more generic term, does not necessarily subsume the engineering paradigm. A software development process, also known as a software development life cycle (SDLC), is a structure imposed on the development of a software product. Similar terms include software life cycle and software process. It is often considered a subset of systems development life cycle.

Agile software development[1] is a group of software development methods based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change. It is a conceptual framework that promotes foreseen interactions throughout the development cycle. The Agile Manifesto introduced the term in 2001. There are many specific agile development methods. Most promote development, teamwork, collaboration, and process adaptability throughout the life-cycle of the project.

Agile methods[1] break tasks into small increments with minimal planning and do not directly involve long-term planning. Iterations are short time frames (time boxes) that typically last from one to four weeks. Each iteration involves a cross functional team working in all functions: planning, requirements analysis, design, coding, unit testing, and acceptance testing. At the end of the iteration a working product is demonstrated to stakeholders. This minimizes overall risk and allows the project to adapt to changes quickly. An iteration might not add enough functionality to warrant a market release, but the goal is to have an available release (with minimal bugs) at the end of each iteration. Multiple iterations might be required to release a product or new features

The release management process[3] is a relatively new but rapidly growing discipline within software engineering of managing software releases. As software systems, software development processes, and resources become more distributed, they invariably become more specialized and complex. Furthermore, software products (especially web applications) are typically in an ongoing cycle of development, testing, and release. Add to this an evolution and growing complexity of the platforms on which these systems run, and it becomes clear there are a lot of moving pieces that must fit together seamlessly to guarantee the success and long-term value of a product or project. The need therefore exists for dedicated resources to oversee the integration and flow of development, testing, deployment, and support of these systems. Although project managers have done this in the past, they generally are more concerned with high-level, "grand design" aspects of a project or application, and so often do not have time to oversee some of the more technical or day-to-day aspects. Release managers (aka "RMs") address this need. They must have a general knowledge of every aspect of the software development process, various applicable operating systems and software application or platforms, as well as various business functions and perspectives. SRM (Software Release Manager) is a prototype software release management tool[3] that we have developed over the past year. It was designed to explore the issues involved in satisfying the requirements presented in the previous

section, and employs a process similar to the traditional \label and archive" paradigm described in the same section. It assumes an archive containing a release is made available to users, and that users are responsible for locating and retrieving these archives. However, SRM enhances this process in two important ways. It structures the information used in the release management process, and then uses this structure in automating much of the process to support developers and users. It hides physical distribution from its users. In particular, dependencies among components can spread across multiple, physically distributed, organizational boundaries, and both developers and users are capable of accessing the information regarding components in a way transparent to the location of the information. SRM realizes this process through a four-part architecture: a logically centralized, but physically distributed, release database; an interface to place components into the release database; an interface to retrieve components from the release database; and a retrieve database at each user site to record information about the components already retrieved.

The rest of paper is organized as follows. Section II presents the related work, Section III presents the overview of genetic algorithm. Proposed approach is discussed in Section IV. Computational experimentation using the proposed Algorithm is presented in section V and then the conclusion is drawn.

II. RELATED WORK

In [1], author analyzes the agile software development process, combining the evolution laws of agile software project, applying AHP method; an integration assessment model for agile software release is presented. In [2], Software release management is closely related to the management of software quality since only software with assured quality should be provided to users. For a software development project, management often faces the dilemma of when to stop testing the software and release it for operation, which requires careful decision making as it has great impact on both software reliability and project cost. In most existing research on the optimal software release problem, the cost considered was the Expected Cost [3] (EC) of the project. However, what concerns management is the Actual Cost (AC) of the project rather than the EC. Treatment (such as minimization) of the EC may not ensure the desired low level of the AC due to the uncertainty (variability) involved in the AC. In this paper, we study the uncertainty in software cost and its impact on optimal software release time in detail. The uncertainty is quantified by the variance of the AC and several risk functions. A risk-control approach to the optimal software release problem is proposed. New formulations of the problem which are extensions of current formulations are developed and solution procedures are established. Several examples are presented. Results reveal that it seems crucial to take into account the uncertainty in software cost in the optimal software release problem; otherwise, unsafe decisions may be reached which could be a false dawn to management.

III. GENETIC ALGORITHM

A genetic algorithm (GA) is a search heuristic that mimics the process of natural evolution. This heuristic is routinely used to generate useful solutions to optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.In a genetic algorithm, a population of strings (called chromosomes or the genotype of the genome), which encode candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem, evolves toward better solutions. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. The evolution usually starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached. Genetic algorithms find application in bioinformatics, phylogenetics, computational science, engineering, economics, chemistry, manufacturing, mathematics, physics and other fields.A typical genetic algorithm requires:

- a genetic representation of the solution domain,
- a fitness function to evaluate the solution domain.

A standard representation of the solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary programming; a mix of both linear chromosomes and trees is explored in gene expression programming. The fitness function is defined over the genetic representation and measures the quality of the represented solution. The fitness function is always problem dependent. For instance, in the knapsack problem one wants to maximize the total value of objects that can be put in a knapsack of some fixed capacity. A representation of a solution might be an array of bits, where each bit represents a different object, and the value of the bit (0

or 1) represents whether or not the object is in the knapsack. Not every such representation is valid, as the size of objects may exceed the capacity of the knapsack. The fitness of the solution is the sum of values of all objects in the knapsack if the representation is valid, or 0 otherwise. In some problems, it is hard or even impossible to define the fitness expression; in these cases, interactive genetic algorithms are used. Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions (usually randomly) and then to improve it through repetitive application of the mutation, crossover, inversion and selection operators. Genetic algorithms are simple to implement, but their behavior is difficult to understand. In particular it is difficult to understand why these algorithms frequently succeed at generating solutions of high fitness when applied to practical problems. The building block hypothesis (BBH) consists of:

- A description of a heuristic that performs adaptation by identifying and recombining "building blocks", i.e. low order, low defining-length schemata with above average fitness.
- A hypothesis that a genetic algorithm performs adaptation by implicitly and efficiently implementing this heuristic.

Goldberg describes the heuristic as follows:

"Short, low order, and highly fit schemata are sampled, recombined [crossed over], and re-sampled to form strings of potentially higher fitness. In a way, by working with these particular schemata [the building blocks], we have reduced the complexity of our problem; instead of building high-performance strings by trying every conceivable combination, we construct better and better strings from the best partial solutions of past samplings.

"Because highly fit schemata of low defining length and low order play such an important role in the action of genetic algorithms, we have already given them a special name: building blocks. Just as a child creates magnificent fortresses through the arrangement of simple blocks of wood, so does a genetic algorithm seek near optimal performance through the juxtaposition of short, low-order, high-performance schemata, or building blocks.

IV. PROPOSED APPROACH

Software Release is one of the most required managerial decisions performed by a project manager. The software release depends on many factors like the available human resources, non human resources, no of parallel processing, capabilities of the resources etc. There are number of approaches to estimate the software development estimation and the release planning. But there is no such approach that collects all the resources and capabilities in one shot and perform the intelligent process on it. We are provides such an intelligent scheduling approach to estimate the software release in two phases i.e. in two versions of the software system. Each version is described with the initial requirement, availability and the capabilities of the resources. It will accept this all data as the training set and perform decision making by analyzing the requirement and the availability of human and non human resources.

A. Proposed Algorithm

STEP 1: Input: Number of Features, Number of Human Resources, Non Human Resources, Release Specification, Workload Data, Productivity Factor
Output: Optimized Release Plan
STEP 2: Input the values
STEP 3: Score the features according to the non human resources required by certain features
STEP 4: Serialize the features
STEP 5: iteration = 0;
STEP 6: for each iteration do
Let X be the best Solution.
Let P1 , P2 be any random chromosomes
O1 be the offspring of P1 and P2
O2 be the offspring of O1 and X
Compare X and O2, if O2 is more optimal than X, replace X with O2
STEP 7: End

B. Architecture

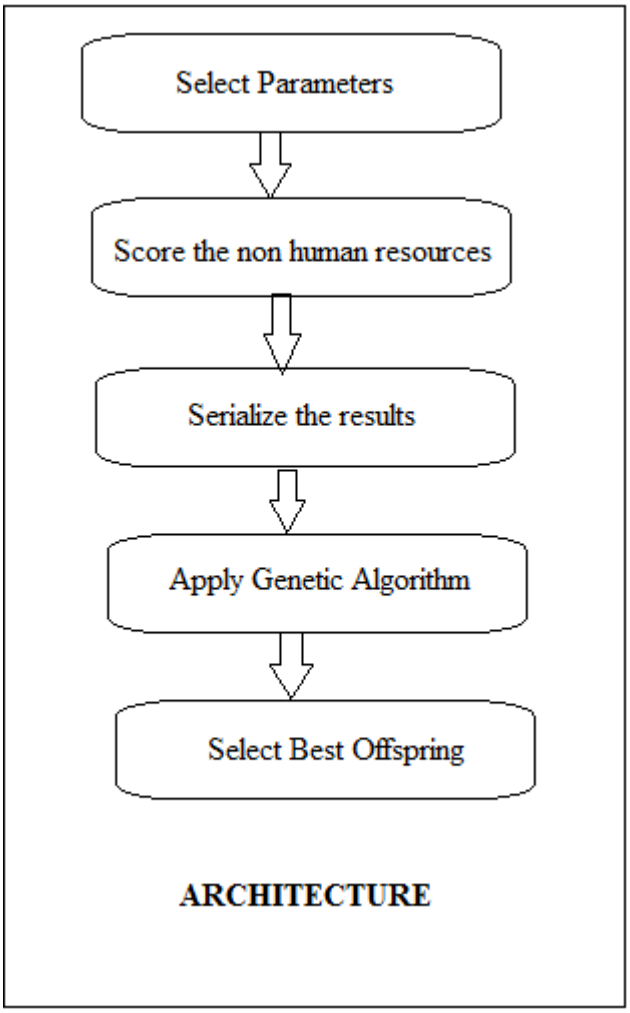

Fig. 1 Architecture

V. COMPUTATIONAL EXPERIMENTATION

Let us consider the example of Library Management System which will serve as the basis of input reading which will be given to the proposed technique. Firstly, number of features of Library Management System is agreed upon. In the proposed technique, following four features are considered:

- Acquisition
- Catalogue
- User Interface
- Review and Ratings

Secondly, number of instances of non human resources is decided. Non Human resources for each feature are decided separately. For example, non human resources required for Acquisition are Hard Disk, RAM, computation time and database. For each feature non human resources available are decided and also non human resource usage for first release is agreed upon. In proposed technique, it is considered that maximum available resources for each feature are 5. Accordingly non human resources for each feature is decided.

Now next step is distribution of workload. Every software project has three basic and most important tasks as:

- Design
- Implementation
- Testing

In proposed technique maximum amount of workload for a feature is 5 while least is 0. The workload for all the features is divided among these three tasks. Next, the most important aspect of software development is considered i.e. human resources. Number of developers is taken into account. Since each developer has a certain amount of time availability for software taking this in consideration, the time availability of each developer is divided among all the three tasks. This whole data is used to prepare a software release for the proposed software. In proposed technique, genetic algorithm is used to

optimize the release plan of software. The advantage of proposed technique over existing method can be explained with the following graph:

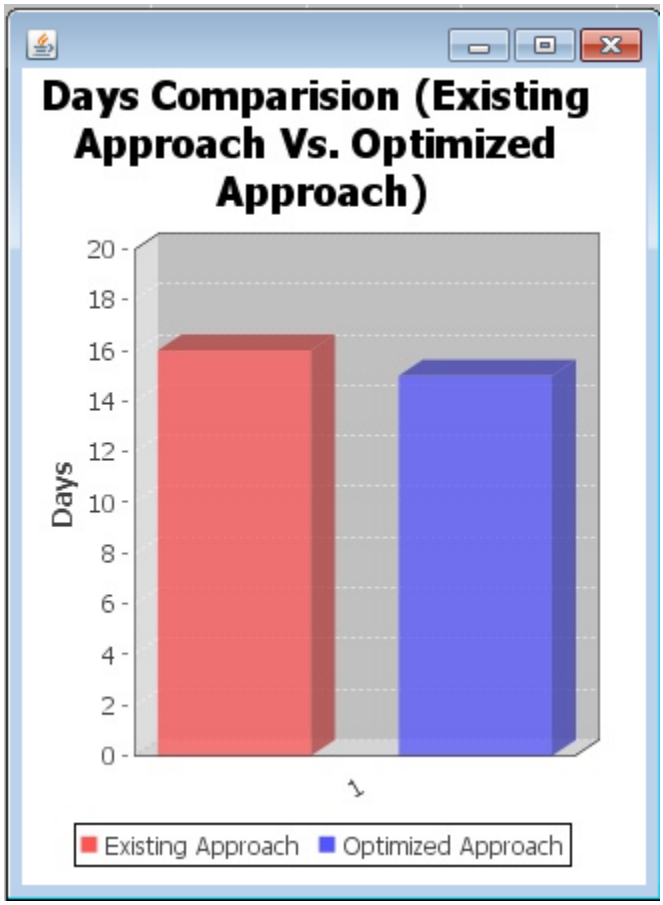

Fig. 2 Day Graph

From above graph, it is quite obvious that optimized approach plans the release plan which requires less number of days for completion of software as compared to existing approach.

VI. CONCLUSION

Software Release Management is an important key technology for distributing the project/product to the customer. It can be concluded from the proposed technique that genetic algorithms can prove to be of great usage for optimizing the software release plan. Further, more the attributes are considered like non human resources, human productivity factors, etc. more precise is the software release plan. Proposed technique can produce an optimal software release plan. Finally, with the proposed technique, we have achieved a technique to plan the software release which is more efficient than the existing schemes.

Few things can be still modified to further optimize software release management. Future work that can be carried out in this project is as follows:

- More attributes like cost, Line of Codes can be considered
- It can be modified to estimate release plan taking into consideration the implementation language that will be used

REFERENCES

[1] Bo Yang, Huajun Hu, and Lixin Ji, "A Study of Uncertainty in Software Cost and Its Impact on Optimal Software Release Time", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 34(6) , NOVEMBER/DECEMBER 2008 0098-5589/08 2008 IEEE

[2] O. Saliu, G. Ruhe. "Supporting Software Release Planning Decisions for Evolving Systems". 29th Annual IEEE/NASA Software Engineering Workshop, Maryland, pp14-26, April 2005.

[3] G. Mikulenas, K. Kapocius. "An Approach for Prioritizing Agile Practices for Adaptation. Information Systems Development", Part 7, pp485-498,2011.

[4] Ákos Sz_ke**.** Conceptual Scheduling Model and Optimized Release Scheduling for agile environments. Information and Software Technology, VOL 53(6), pp574-591, 2011

[5] Aaqib I., Farhan., Khan etc.. A critical analysis of techniques for requirement priorization and open research issues. International Journal of Reviews in Computing, pp8-18, 2009

[6] Jianbai Huang, Tao Yang, Yingjie Tang. Research on Software Process Metric Based on AHP and SPC. The Sixth Wuhan International Conference on E-Business (WHICEB2007), China, Wuhan, pp. 2443-2451,May 2007

[7] Ishizaka A.o, Labib A., Analytic Hierarchy Process and Expert Choice: Benefits and Limitations, ORInsight, VOL 22(4), pp. 201–220, 2009.

[8] Chen Li, Marjan van,Sjaak B. etc.. An integrated approach for requirement selection and scheduling in software release planning. Requirements Eng v15, pp375–396, 2010

[9] Kamal M. Application of the AHP in project management. International Journal of Project Management ,VOL 19, pp19-27, 2001

[10] Omolade S., Guenther R.. Software release planning for evolving systems. Innovations in Systems and Software Engineering, VOL 1(2), pp189-204, 2005

[11] Günther Ruhe, Des Greer. Quantitative Studies in Software Release Planning under Risk and Resource Constraints. Proceedings of the 2003 International Symposium on Empirical Software Engineering (ISESE '03), IEEE Computer Society Washington, DC, USA, pp1-10, 2003.

[12] D. Greer, G. Ruhe. Software Release Planning: An Evolutionary and Iterative Approach. Information and Software Technology, VOL 46(4), pp243-253, 2004

