Quantitative Studies in Software Release Planning under Risk and Resource Constraints

Günther Ruhe
University of Calgary
2500 University Drive NW
Calgary, AB T2N 1N4, Canada
ruhe@ucalgary.ca

Des Greer
Queens University Belfast
Belfast BT7 1NN
United Kingdom
des.greer@qub.ac.uk

Abstract

Delivering software in an incremental fashion implicitly reduces many of the risks associated with delivering large software projects. However, adopting a process, where requirements are delivered in releases means decisions have to be made on which requirements should be delivered in which release. This paper describes a method called EVOLVE+, based on a genetic algorithm and aimed at the evolutionary planning of incremental software development. The method is initially evaluated using a sample project. The evaluation involves an investigation of the trade-off relationship between risk and the overall benefit. The link to empirical research is two-fold: Firstly, our model is based on interaction with industry and randomly generated data for effort and risk of requirements. The results achieved this way are the first step for a more comprehensive evaluation using real-world data. Secondly, we try to approach uncertainty of data by additional computational effort providing more insight into the problem solutions: (i) Effort estimates are considered to be stochastic variables following a given probability function; (ii) Instead of offering just one solution, the L-best (L>1) solutions are determined. This provides support in finding the most appropriate solution, reflecting implicit preferences and constraints of the actual decision-maker. Stability intervals are given to indicate the validity of solutions and to allow the problem parameters to be changed without adversely affecting the optimality of the solution.

Keywords: *Incremental Software Development, Release Planning, Uncertainty, Quantitative Analysis, Decision Support, Genetic Algorithm, Risk Management, Resource constraints.*

1. Background and Motivation.

There is a growing recognition that an incremental approach to software development is often more suitable and less risky than the traditional waterfall approach [10]. This preference is demonstrated by the current popularity of agile methods, all of which adopt an incremental approach to delivering software rapidly [20]. This shift in paradigm has been brought about by many factors, not least the rapid growth of the World Wide Web, the consequent economic need to deliver systems faster [4] and the need to ensure that software better meets customer needs. Further, it has been recognized that even where an incremental model is not the initial process of choice, it is often adopted at the maintenance stage due to unanticipated changes [19].

Software development companies have real constraints for competitive market edge and delivery of a quality product. To achieve quality processes and practices there are permanent trades-offs to the different aspects related to the final quality of the product. Decision processes are the driving forces to organize a corporation's success [6]. Software Engineering Decision Support (SEDS) is a new paradigm for learning software organizations [21]. Its main goal is provide support for decision-making based on best knowledge and experience, computational and human intelligence, as well as a suite of sound and appropriate methods and techniques. Computerized decision support should be considered in unstructured decision situations characterized by one or more of the following factors: complexity, uncertainty, multiple groups with a stake in the decision outcome (multiple stakeholders), a large amount of information (especially company data), and/or rapid change in

Proceedings of the 2003 International Symposium on Empirical Software Engineering (ISESE'03)

information. Support here means to provide access to information that would otherwise be unavailable or difficult to obtain; to facilitate generation and evaluation of solution alternatives, and to prioritize alternatives by using explicit models that provides structure for particular decisions.

In the incremental software process model, requirements are gathered in the initial stages and, taking technical dependencies and user priorities into account and the effort required for each requirement, the system is divided into increments. These increments are then successively delivered to customers. It is often true that any given requirement could be delivered in one, several or even all releases. Consequently, there is a need to decide which requirements should be delivered in which release. It may make sense to deliver those with highest user value in the earliest increments, and thus gain more benefits early on [8]. Since there are likely to be many different users all with different viewpoints on what user value of requirements is, this decision is potentially very complex. Exacerbating this is the fact that there are a range of constraints, one of which is the desired maximum effort for any given increment. In addition to this factor, risk is an important consideration. A given project may have a risk referent. This is a level of risk, which should not be exceeded [3]. In an incremental delivery model, this means that a given release has also a risk referent.

In response to these issues, we have developed an evolutionary and iterative approach called EVOLVE+ that offers quantitative analysis for decision support in software release planning. The model is extended from [11] and takes into account: (i) the priorities of the representative stakeholder groups with respect to requirements; (ii) the effort estimate for implementing each requirement and the effort limit for each release; (iii) precedence constraints, where one requirement must occur in a release prior to the release for another requirement; (iv) coupling constraints where a group of requirements must occur in the same release; (v) resource constraints where certain requirements may not be in the same release; and (vi) a risk factor estimate for each requirement and a maximum risk referent value, calculated from this for each release.

In situations where there are complex relationships between several factors and consequently a very large solution space, genetic algorithms have been shown to be appropriate [1]. Genetic algorithms have arisen from an analogy with the natural process of biological evolution [13] and usually start with a population of randomly generated solutions, each member of the population being evaluated by some objective function. The next step is selection, where solutions (or chromosomes) are chosen from the population to be used in creating the next generation. In this process, those with a superior fitness score are more likely to be selected. The next generation is created by a crossover operator, which mixes pairs of chromosomes to create new offspring containing elements of both parents. A further operation, mutation is applied to introduce diversity in the population by creating random changes in the new offspring. As new offspring are created, typically the worst scoring chromosomes are replaced. The selection mechanism, the size of the population, the amount and type of crossover, the extent and mechanism of mutation and the replacement strategy are all variable and can be adjusted, dependent on the problem being solved.

Hence, a computationally very efficient algorithm based on the principles of evolution is proposed as a potentially suitable means to find L-best release plans and to handle uncertainties in effort estimates. Support for decision-making typically involves the development of not only one solution, but a set of ‘best’ candidates where the actual decision-maker can select from, according to their implicit preferences and subjective constraints.

The problem and the model used to solve it are described in Section 2. Our model is based on interaction with industry. The proposed solution approach called EVOLVE+ is presented in Section 3. In keeping with current advice for evaluation of software engineering theory, methods and tools empirically [14][16], we have designed and carried out a series of experiments to evaluate EVOLVE+. Randomly generated data for effort and risk of requirements were initially used as an example project described in Section 4. This includes stochastic variables for effort constraints as well as sensitivity and stability analysis. Section 5 will provide a summary of the findings and identify potential extensions for future research

2. Model Building and Problem Statement.

Release planning for incremental software development includes the assignment of requirements to releases such that all technical, risk, resource and budget constraints are fulfilled. The overall goal is to find an assignment of requirements to increments that maximizes the

Proceedings of the 2003 International Symposium on Empirical Software Engineering (ISESE'03)

sum of all (weighted) priorities of all the different stakeholders. In what follows, we more formally introduce all the necessary concepts and notation.

2.1 Stakeholder Priorities

One of the challenges to the software engineering research community is to involve stakeholders in the requirements engineering process. In any project, several stakeholders may be identified. A stakeholder is defined as anyone who is materially affected by the outcome of the project. Effectively solving any complex problem involves satisfying the needs of a diverse group of stakeholders. Typically, stakeholders will have different perspectives on the problem and different needs that must be addressed by the solution. Example stakeholders are the different types of users of the system, investor, production manager, shareholder, or designer. An understanding of who the stakeholders are and their particular needs are key elements in developing an effective solution for the software release planning problem.

It is assumed that a software system is initially specified by a set R^1 of requirements, i.e., $R^1 = \{r_1, r_2 \ldots r_n\}$. At this stage (k=1), we wish to allocate these requirements to the next and future releases. In a later phase k (>1), an extended and/or modified set of requirements R^k will be considered as a starting point to plan for increment k (abbreviated by Inc^k). The requirements are competing with each other (to become implemented).

Their individual importance is considered from the perspective of q different stakeholders abbreviated by $S_1, S_2, \ldots, S_q$. Each stakeholder S_p is assigned a relative importance $\lambda_p \in (0,1)$. The relative importance of all involved stakeholders is normalized to one, i.e., $\Sigma_{p=1,\ldots,q}\ \lambda_p = 1$.

Each stakeholder S_p assigns a priority denoted by $prio(r_i, S_p, R^k)$ to requirement r_i as part of set of requirements R^k at phase k of the planning approach. prio() is a function: $(r_i, S_p, R^k) \rightarrow \{1,2,..,\sigma\}$ assigning a priority value to each requirement for each set R^k. Typically, different stakeholders have different priorities for the same requirement.

2.2 Evolution of Increments

As result of the planning process, different increments will be composed out of the given set of requirements. These increments are planned up-front but the possibility of re-planning after any increment is allowed. This re-planning may involve changing some requirements, priorities and constraints and/or introducing new ones. It necessitates a reassignment of requirements (not already implemented in former releases) to increments. Throughout the paper, we assume that the number of releases is not fixed upfront. The complete modeling and solution approach remains valid with only minor modifications for the case of fixed number of releases.

Phase k of the overall planning procedure EVOLVE+ is abbreviated by EVOLVE+(k). The input of EVOLVE+(k) is the set of requirements R^k. The output is a definition of increments Inc^k, Inc^{k+1}, Inc^{k+2}, … with $Inc^t \subset R^k$ for all t = k, k+1, k+2, … The different increments are disjoint, i.e., $Inc^s \cap Inc^t = \emptyset$ for all $s,t \in \{k, k+1, k+2, \ldots\}$. The unique function ω^k assigns each requirement r_i of set R^k the number s of its increment Inc^s, i.e., ω^k: $r_i \in R^k \rightarrow \omega^k(r_i) = s \in \{k, k+1, k+2, \ldots\}$.

2.3 Effort Constraints

Effort estimation is another function assigning each pair (r_i, R^k) of requirement r_i as part of the set R^k the estimated value for implementing this effort, i.e., effort() is a function: $(r_i, R^k) \rightarrow \mathfrak{R}^+$ where $\mathfrak{R}^+$ is the set of positive real numbers. Please note that the estimated efforts can be updated during the different phases of the overall procedure.

Typically project releases are planned for certain dates. This introduces a size constraint $Size^k$ in terms of effort of any released increment Inc(k). We have assumed that the effort for an increment is the sum of the efforts required for individual requirements assigned to this increment. This results in the set of constraints $\Sigma_{r(i) \in Inc(k)}$ $effort(r_i, R^k) \leq Size^k$ for all increments Inc(k).

2.4 Balancing Risk per Increment

Risk estimation is used to address all the inherent uncertainty associated with the implementation of a certain requirement. In what follows, we employ a risk score as an abstraction of all risks associated with a given requirement. These risks may refer to any event that potentially might negatively affect schedule, cost or quality in the final project results. For each pair (r_i, R^k) of requirement r_i as part of the set R^k the estimated value for implementing this effort, i.e. risk is a ratio scaled function risk: $(r_i, R^k) \rightarrow [0,1)$, where ‘0’ means no risk at all and ‘1’ stands for the highest risk. In what follows we assume that the risk assessment is done by expert judgment.

The idea of risk balancing is to avoid a concentration of highly risky requirements into the same increment. This leads to constraints $\Sigma_{r(i) \in Inc(k)}$ $risk(r_i, R^k) \leq Risk^k$ for each increment k. The risk per increment is supposed to be additive, and $Risk^k$ denotes the upper bound for the acceptable risk for the k-th increment, the risk referent.

2.5 Precedence, Dependency and Resource Constraints

In a typical real world project, it is likely that some requirements must be implemented before others. There might be logical or technical reasons that the realization of one requirement must be in place before the realization of another. Since we are planning incremental software delivery, we are only concerned that their respective increments are in the right order. More formally, for all iterations k we define a partial order Ψ^k on the product set R^k x R^k such that $(r_{i,} r_j) \in \Psi^k$ implies $\omega^k(r_i) \leq \omega^k(r_j)$.

With similar arguments as above, there might be logical or technical reasons implying that the realization of one requirement must be in place in the same increment as another one. Again, since we are concerned with incremental software delivery, we are only concerned that their respective increments are in the right order. More formally, for all iterations k we define a binary relation ξ^k on R^k such that $(r_{i,} r_j) \in \xi^k$ implies that $\omega^k(r_i) = \omega^k(r_j)$ for all phases k.

Finally, we consider resource constraints. We assume that certain requirements are known to allocate the same type of resource, and putting them into the same increment would exceed the given capacity of that resource. In general, there are index sets $I(t) \subset \{1,\ldots,n\}$ such that $card(\{r_i \in R^k: i \in I(t)\}) \leq resource(t)$ for all releases k and for all resources t. Therein, resource(t) denotes the capacity available of resource t.

2.6 Problem Statement for Software Release Planning

We assume an actual set of requirements R^k. Taking into account all the notation, concepts and constraints as formulated above, we can now formulate our problem as follows:

For all requirements $r_i \in R^k$ determine an assignment ω^*: $\omega^*(r_i)$= m of all requirements r_i to an increment $\omega^*(r_i)$= m such that

(1) $\Sigma_{r(i) \in Inc(m)}$ $effort(r_i, R^m) \leq Size^m$ for m = k,k+1,… (Effort constraints)

(2) $\Sigma_{r(i) \in Inc(m)}$ $risk(r_i, R^m) \leq Risk^m$ for m = k,k+1,… (Risk constraints)

(3) $\omega^*(r_i) \leq \omega^*(r_j)$ for all pairs $(r_{i,} r_j) \in \Psi^k$ (Precedence constraints)

(4) $\omega^*(r_i) = \omega^*(r_j)$ for all pairs $(r_{i,} r_j) \in \xi^k$ (Coupling constraints)

(5) $card(\{r_i \in R^k: i\in I(t)\}) \leq resource(t)$ for all releases k and all sets I(t) related to all resources t

(6) $A = \Sigma_{p=1\ldots,q}\ \lambda_p\ [\Sigma_{r(i)\in R(k)} benefit(r_{i,},S_p,\omega^*)]$ $\Rightarrow$ L-max! with $benefit(r_i,S_p,R^k,\omega^*)=[\pi\text{-}prio(r_{i,}S_{p,}\ R^k)+1][\tau - \omega^*(r_{j,})+1]$ and $\tau = \max\{\omega^*(r_i): r_i \in R^k\}$

The function (6) is to maximize the weighted benefit over all the different stakeholders. L-max means to compute not only one, but a set of L best solutions (L>1). That means for all feasible assignments ω^h, h=1,2,… there is a subset Ω (with $card(\Omega)$=L) of assignments such that any assignment outside is Ω is at least not better (related to (6)) than any assignment from Ω.. For a fixed stakeholder, the benefit from the assignment of an individual requirement to an increment is the higher, the earlier it is released and the more important it is.

3. Solution Approach

3.1 EVOLVE+ Approach

The proposed approach called EVOLVE+ combines the computational strength of genetic algorithms with the flexibility of an iterative solution method. At all iterations, a genetic algorithm is applied to determine L-best solutions of (1) - (6).

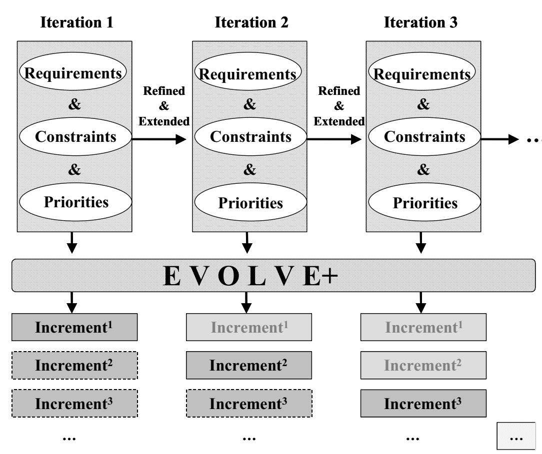

Figure 1. EVOLVE+ approach to assign requirements to increments.

Maximization of objective function (6) is the

Proceedings of the 2003 International Symposium on Empirical Software Engineering (ISESE'03)
0-7695-2002-2/03 $ 17.00 © 2003 IEEE

main purpose of conducting crossover and mutation operations. This function is computed as a fitness value for each optimization step of the genetic algorithm. The algorithm terminates when there is no further improvement in the solution. This is calculated as no improvement in the objective function value achieved within 0.5% deviation over six hundred simulations. The algorithm works by producing orderings of requirements and, if valid, calculating this fitness score.

For each solution generated by the genetic algorithm, each of the constraints is checked. The effort constraint (1) is handled by a greedy-like increment allocation algorithm, which is run on every ordering produced by the genetic algorithm. This can be a discrete effort estimation or, alternatively, the model allows for the possibility that effort estimates are represented by probability distributions. If this is the case, the RiskOptimizer tool, provided with the necessary parameters, simulates the distribution using Latin Hypercube sampling over thousands of iterations. The increment allocation is implemented as before, but the effort constraint is calculated as the value at a chosen percentile. This allows the decision maker to select the required confidence levels for keeping within the effort constraints for individual releases. The risk constraint is checked by summing the total risk for each release and comparing this with the risk referent (2). Precedence constraints (3), coupling constraints (4) and resource constraints (5) are implemented by specific rules used to check each generated solution. In the case of precedence and coupling constraints a table of pairs of requirements is held. For precedence constraints, the first member of the pair is a predecessor of the second. For coupling constraints, the pairs must be in the same release. For the resource constraints, a list of resources is maintained and for each resource an available capacity limit is estimated. This refers to number of occurrences allowed for that resource in a given release. The required usage of these resources for each requirement is held in a table and the total usage for any release must not exceed the available capacity of any resource. In all three cases, if any given solution is generated that violates the constraint, the solution is rejected and a backtracking operation is used to generate a new solution.

In calculating the benefit, weightings are used to discriminate between stakeholders, these weightings being calculated using the pair-wise comparison method from AHP [22].

EVOLVE+ is an evolutionary approach. At iteration k, a final decision is made about the next immediate increment Inc^k and a solution is proposed for all subsequent increments Inc^{k+1}, $Inc^{k+2,}$ …The reason for the iterative part in EVOLVE+ is to allow all kinds of late changes in requirements, prioritization of requirements by stakeholders, effort estimation for all requirements, effort constraints, precedence and coupling constraints as well as changes in the weights assigned to stakeholders. This most recent information is used as an input to iteration k+1 to determine the next increment Inc^{k+1} as final and all subsequent ones Inc^{k+2}, $Inc^{k+3,}$ …as tentatively again. This is illustrated in Figure 1.

3.2 Algorithms and Tool Support

In this research, we have made use of Palisade's RiskOptimizer tool [17]. The RiskOptimizer tool provides different algorithms for adjusting the variables. Since we are concerned with ranking requirements, the most suitable one provided is the 'order' method. This method generates different permutations of a starting solution and is designed for optimizing rankings of objects. The order genetic algorithm is described in [5].

Selection is effected by choosing two parents from the current population, the choice being determined by relating the fitness score to a probability curve [18]. The crossover operator mixes two solutions maintaining some sub-orderings of both. Mutation is carried out after crossover and is intended to introduce variance and so avoid terminating at a local solution. Thus, mutation introduces new orderings in the population that might not be reached if only crossover operations were used. In the order method, this means that a random change in order occurs by swapping, the number of swaps being proportional to the mutation rate. In cases where organisms are generated outside the solution space a backtracking process is employed, where the tool reverts to one of the parents and retries the crossover and mutation operations until a valid child is obtained. This case arises if one or more of the constraints are violated. The extent of crossover and mutation is controlled by the genetic operators *crossover rate* and *mutation rate*.

The process of selection, crossover and mutation continues, with the worst performing organisms being replaced with the newly created organisms, which have better fitness scores. The genetic algorithm is terminated after a certain number of optimizations or after there ceases to be any significant improvement in the best fitness score achieved. This is achieved by stating that the process should terminate if there is no improvement (or less than a certain percentage increase) in the best fitness score over n optimizations.

4. Evaluation

4.1 Description of Sample Project

In evaluation of the method, a sample software project with twenty requirements was used. We have represented this initial set, R^1 of requirements by identifiers r_1 to r_{20}. The technical precedence constraints in our typical project are represented by the set, Ψ (compare Section 2.6) as shown below. This states that r_1 must come before r_6, r_{19}, r_3 and r_{12} and r_{11} before r_{19}.

$\Psi^1 = \{(r_1,r_6), (r_1,r_{19}),(r_1,r_3),(r_1,r_{12}),(r_{11},r_{19})\}$

Further, some requirements were specified to be implemented in the same increment as represented by the set ξ, as defined in Section 2.6. This states that r_3 and r_{12} must be in the same release, as must r_{11} and r_{13}.

$\xi^1 = \{(r_3, r_{12}),(r_{11}, r_{13})\}$

Resource constraints are represented by index sets for the set of those requirements asking for the same resource. In our sample project, we have a resource T_1 which has a capacity of 1 (resource(T_1)=1) that is used by requirements r_3 and r_8. The sample project had one single resource constraint represented by the following index set.

$I(T_1) = \{3,8\}$

Each requirement has an associated effort estimate in terms of a score between one and ten. The effort constraint was added that for each increment the effort should be less than twenty-five, i.e., $Size^k = 25$ for all releases k. Five stakeholders were used to score the twenty requirements with priority scores from one to five. These scores are shown in Table 1. As we can see, different stakeholders in some cases assign more or less the same priority to requirements (as for r_1 and r_{17}). However, the judgment is more conflicting in other cases (as for r_{11}).

The stakeholders, S_1 to S_5, were weighted using AHP pair-wise comparison from a global project management perspective. The technique of averaging over normalized columns [22] can be used to approximate the eigenvalues. As a result, we achieved the vector (0.211, 0.211, 0.421, 0.050, 0.105) assigning priorities to the five stakeholders.

Table 1: Sample stakeholder-assigned priorities.

		Stakeholder				
		S_1	S_2	S_3	S_4	S_5
Requirement	r_1	1	1	1	1	1
	r_2	3	3	2	4	3
	r_3	2	2	2	2	1
	r_4	3	2	2	4	2
	r_5	4	5	3	4	4
	r_6	4	4	4	5	5
	r_7	2	1	1	1	1
	r_8	3	5	3	5	5
	r_9	5	5	5	3	4
	r_{10}	3	4	5	4	3
	r_{11}	5	2	3	1	4
	r_{12}	2	4	3	2	4
	r_{13}	1	5	2	2	2
	r_{14}	3	3	4	3	5
	r_{15}	1	3	2	2	3
	r_{16}	3	1	2	2	2
	r_{17}	5	5	5	5	5
	r_{18}	1	2	2	3	1
	r_{19}	2	3	4	3	4
	r_{20}	5	5	4	3	4

With regard to genetic algorithm parameters, we used the default population size of fifty and the built in auto-mutation function which detects when a solution has stabilized and then adjusts the mutation rate to try to improve the solution. In preliminary experiments, we established that it was not possible to consistently predict the best crossover rate. Hence we used a range of crossover rates for each experiment.

4.2 Empirical Studies

In evaluating the method, we at first randomly generated solutions for the sample data. Using deterministic estimations for effort it was found out of 1000 random rankings, only 1.7% were valid. This does not even consider the objective function value. This demonstrates how difficult it would be to successfully employ a manual or even a greedy-type increment assignment. However, this is not surprising for a NP-complete problem like the one under consideration.

Three particular aspects of the method were investigated to validate their usability for larger and real-world problems.

4.2.1 Risk versus Benefit

Firstly, the relationship between the overall risk in a software release and the benefit objective function was considered. As expected, increasing the risk referent or level of risk acceptable in an increment increases the achievable benefit in the solutions provided by the method. As an example, we increased the initial risk referent normalized to 1.0. The resulting trade-off curve is shown in Figure 2.

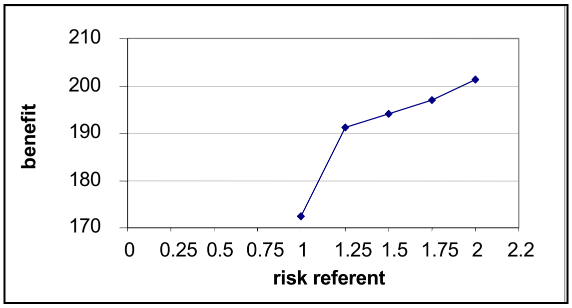

Figure 2. Trade-off relationship between risk referent and achieved benefit.

4.2.2 Uncertainty in Effort Estimates

Secondly, the level of uncertainty for the effort estimates was investigated. To achieve this, the effort estimates are treated as probability distributions. For our experiments, a triangular probability distribution for the estimated effort to implement requirements was used. The complete definition of all those functions is summarized in Table 2.

Using stochastic variables for the estimated effort of requirements introduces uncertainty into the model and the process of assigning requirements to releases. This means that the decision maker can plan the releases based on their confidence level in the estimates. In practice there is a trade-off between this level of confidence and the benefit achievable. It also means that the adopted release plan has associated with it, a percentage indicating the probability that it will that be adhere to its effort estimates.

For the computations using stochastic variable for the effort estimates, we have fixed the risk referent for each increment to 1.1. In the same experiment, we additionally introduced another resource constraint. Resource T_2 which has a capacity of 2, is used by requirements r_3, r_8, and r_{19}, e.g., $I(T_2) = \{3,8,19\}$. Precedence and coupling constraints were assumed to be the same as described in Section 4.1. Furthermore, for the sake of simplicity we assumed $Size^k = 25$ for all increments k.

Table 2. Definition of triangular probability functions for the effort of all requirements.

		Effort		
		Min	Mode	Max
Requirement	r_1	5	10	15
	r_2	2	4	6
	r_3	1	2	3
	r_4	2	3	4
	r_5	3	4	5
	r_6	4	7	10
	r_7	0	1	2
	r_8	0	2	4
	r_9	5	10	15
	r_{10}	1	3	5
	r_{11}	0	2	4
	r_{12}	4	5	6
	r_{13}	1	2	3
	r_{14}	7	8	9
	r_{15}	0	1	2
	r_{16}	3	4	5
	r_{17}	0	1	2
	r_{18}	3	4	5
	r_{19}	2	4	6
	r_{20}	5	8	11

Table 3 gives the results of the experiments for three different levels for the probability of not exceeding the effort bound. With increasing risk of exceeding the available effort capacity, the expected benefit is increasing.

4.2.3 L-best Solutions and their Stability

The final computation is related to determine L-best (L>1) solutions. This means that the decision-maker can finally choose from the L most promising solutions rather than accept one solution. This allows the decision maker to take into account the uncertainty of the data, the implicit constraints and their own preferences. Stability intervals are given to indicate the validity of solutions and to allow the problem parameters to be changed without adversely affecting the solution. Table 4 summarizes L-best solutions for a fixed level of risk (1.0) and L = 5.

Proceedings of the 2003 International Symposium on Empirical Software Engineering (ISESE'03)
0-7695-2002-2/03 $ 17.00 © 2003 IEEE

Table 3. Results for release planning under different levels of probability for not exceeding the effort capacity bound Effort = 25 (Risk Referent=1.1). Risk = Total Risk Score $\Sigma_{r(i) \in Inc(1)}$ risk(r_i, R^1). Effort = Total Effort at 95th percentile $\Sigma_{r(i) \in Inc(1)}$ effort(r_i, R^1).

Probability	Benefit (A)	Risk	Effort	Release	Assigned Requirements
95%	201	0.56	23.4	1	r_1 r_2 r_8
		0.82	24.0	2	r_5 r_7 r_{10} r_{14} r_{16}
		1.08	24.4	3	r_3 r_4 r_6 r_{12} r_{15} r_{17}
		0.64	24.7	4	r_9 r_{11} r_{13} r_{19}
		0.24	13.4	5	r_8 r_{20}
90%	218	0.84	23.8	1	r_1 r_2 r_7 r_{18}
		0.89	24.3	2	r_4 r_5 r_8 r_{14} r_{16}
		1.05	23.9	3	r_3 r_6 r_{11} r_{12} r_{13} r_{15}
		0.45	23.4	4	r_9 r_{10} r_{17} r_{19}
		0.11	9.66	5	r_{20}
80%	224	0.9	24.6	1	r_1 r_3 r_{11} r_{12} r_{13}
		0.91	24.5	2	r_2 r_{14} r_{15} r_{16} r_{17} r_{18}
		1.00	23.6	3	r_4 r_5 r_7 r_8 r_9
		0.44	20.9	4	r_6 r_{10} r_{20}
		0.09	4.7	5	r_{19}

Table 4: L-best solutions (L=5) for software release planning and related stability intervals (Risk Referent=1.0).

Risk Level	Stability Interval	Benefit (A)	Release	Effort Σr(i)∈Inc(1) effort(ri, R1)	Assigned Requirements
0.85	[0, 0.15]	172.2	1	25	r_7 r_{10} r_{14} r_{15} r_{16} r_{20}
0.99	[0, 0.01]		2	25	r_1 r_3 r_{11} r_{12} r_{13} r_{19}
0.95	[0, 0.05]		3	25	r_2 r_4 r_5 r_9 r_{18}
0.55	[0, 0.45]		4	10	r_6 r_8 r_{17}
0.91	[0, 0.09]	171.8	1	25	r_1 r_7 r_8 r_{14} r_{18}
0.98	[0, 0.02]		2	25	r_2 r_3 r_{12} r_{15} r_{16} r_{17} r_{20}
0.90	[0, 0.10]		3	25	r_5 r_9 r_{10} r_{11} r_{13} r_{19}
0.55	[0, 0.45]		4	10	r_4 r_6
0.96	[0, 0.04]	170.2	1	25	r_1 r_2 r_7 r_{15} r_{17} r_{20}
0.90	[0, 0.10]		2	25	r_3 r_9 r_{11} r_{12} r_{13} r_{19}
0.93	[0, 0.07]		3	25	r_5 r_8 r_{10} r_{14} r_{16} r_{18}
0.55	[0, 0.45]		4	10	r_4 r_6
0.96	[0, 0.04]	170.1	1	25	r_1 r_2 r_5 r_8 r_{10} r_{15} r_{17}
0.90	[0, 0.10]		2	25	r_3 r_9 r_{11} r_{12} r_{13} r_{19}
0.93	[0, 0.07]		3	25	r_7 r_{14} r_{16} r_{18} r_{20}
0.55	[0, 0.45]		4	10	r_4 r_6
0.91	[0, 0.09]	168.9	1	25	r_1 r_7 r_8 r_{14} r_{18}
0.91	[0, 0.09]		2	25	r_2 r_3 r_9 r_{11} r_{12} r_{13}
0.97	[0, 0.03]		3	25	r_5 r_{10} r_{15} r_{16} r_{17} r_{19} r_{20}
0.55	[0, 0.45]		4	10	r_4 r_6

Proceedings of the 2003 International Symposium on Empirical Software Engineering (ISESE'03)
0-7695-2002-2/03 $ 17.00 © 2003 IEEE

5. Summary and Conclusions

This paper has described the investigation of a method for software release planning under effort, risk and resource constraints. The original model includes effort, precedence, coupling and resource constraints. Future research is directed to further relax the underlying assumptions concerning availability of precise and complete data.

The method developed and evaluated is EVOLVE+, which takes as its main input, a set of requirements with their effort and risk estimations. EVOLVE+ has been developed from earlier work and incorporates feedback from industry, so that the additional aspects of resource constraints, effort estimate uncertainty and risk analysis have been included. The new method uses a genetic algorithm to optimize the solution within pre-defined technical constraints. To assess potential release plans, an objective function has been defined that measures the benefits of delivering requirements in their assigned increment. The method is applied iteratively to provide candidate plans for the next and future releases.

EVOLVE+ differs from previous prioritization techniques[15] in that: it is specifically aimed at an incremental software process; it takes into account stakeholder priorities as well as effort constraints for all releases; it considers inherent precedence, coupling and resource constraints; it assumes that changes to requirements and the project attributes will take place over time, better matching the reality of most software projects; it caters for conflict in stakeholder's priorities while recognizing that stakeholders opinions are not always equal; and it generates the L-best solutions, allowing the decision maker to make the final choice.

In investigating the method we have used a sample project of twenty requirements. Overall, we found the method to provide feasible solutions, which provide a balance between the conflicting interests of stakeholders, take account of the available effort required for a given release, limits the level of risk in each software release and provides an estimate of the confidence level of the effort predictions for releases.

The initial evaluation by a sample project gives sufficient confidence to apply EVOLVE+ for real-world data sets from industry. Future development work relating to EVOLVE+ will include a comprehensive empirical evaluation using real-world data. First steps in this direction are very promising. Two initial case studies indicate that EVOLVE+ is able to solve problems more effectively and more efficiently with even hundreds of requirements and a large number of involved stakeholders.

Acknowledgements

The authors would like to thank the Alberta Informatics Circle of Research Excellence (iCORE) for their financial support of this research. Many thanks are due to Mr Wei Shen for conducting numerical analysis using RiskOptimizer and Mark Stanford from Corel for stimulating discussions on model building.

References

[1] L.C. Briand, J. Feng and Y. Labiche, "Experimenting with Genetic Algorithm to Devise Optimal Integration Test Orders", TR Department of Systems and Computer Engineering, Software Quality Engineering Laboratory Carleton University, 2002.

[2] J. Carnahan and R. Simha, "Natures's Algorithms", *IEEE Potentials*, April/May, pp21-24, 2001.

[3] R. Charette, *"Software Engineering Risk Analysis and Management"*, McGraw-Hill, New York, 1989.

[4] M.A. Cusamano, and D.B. Yoffie, *"Competing on Internet Time: Lessons From Netscape and Its Battle with Microsoft",* The Free Press, New York, 1998.

[5] L. Davis, *"Handbook of Genetic Algorithms"*, Van Nostrand Reinhold, New York, 1991.

[6] G. De Gregorio, "Enterprise-wide Requirements and Decision Management", *Proc. 9th International Symposium of the International Council on System Engineering*, Brighton, 1999.

[7] J. Fitzgerald and A.F. Fitzgerald,., "A Methodology for Conducting A Risk Assessment", *Designing Controls into Computerized System*, Chapter 5, 2nd ed., Redwood, California, Jerry Fitzgerald & Associates, 1990.

[8] T. Gilb, "*Principles of Software Engineering Management"*, Addison-Wesley, 1988.

[9] D. Greer, D. Bustard, and T. Sunazuka, "Prioritisation of System Changes using Cost-Benefit and Risk Assessments", *Fourth IEEE International Symposium on Requirements Engineering*, pp 180-187, June, 1999.

[10] D. Greer, D. Bustard and T. Sunazuka, "Effecting and Measuring Risk Reduction in Software Development", *NEC Journal of Research and Development*, Vol.40, No.3, 1999 pp.378-38.

[11] D. Greer and G. Ruhe, "Software Release Planning: An Evolutionary and Iterative Approach", appears in: Journal for Information and Software Technology, 2003.

[12] H.W. Hamacher and G. Ruhe, "On Spanning Tree Problems with Multiple Objectives", *Annals of Operations Research* 52(1994), pp 209-230.

[13] J.H. Holland, "*Adaptation in Natural and Artificial Systems"*. University of Michigan Press, Ann Arbor, 1975.

[14] R. Jeffrey and L. Scott, "Has twenty five years of empirical software engineering made a difference", *Proceedings Asia-Pacific Software Engineering Conference,* (Eds. Strooper, P., Muenchaisri, P), 4-6 Dec. 2002, IEEE Computer Society, Los Alamitos, California, pp 539-546.

[15] J. Karlsson, C. Wohlin, and B. Regnell, "An evaluation of methods for prioritizing Software Requirements", *Information and Software Technology*, 39(1998), pp 939-947.

[16] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El-Emam, and J. Rosenberg, "Preliminary guidelines for empirical research in software engineering", *IEEE Transactions on Software Engineering*, Volume: 28(2002), pp . 721 –734.

[17] Palisade Corporation, Palisade Corporation, 31 Decker Road, Newfield, NY 14867, www.Palisade.com, September, 2002.

[18] Palisade Corporation, *Guide to RISKOptimizer: Simulation Optimization for Microsoft Excel Windows Version Release 1.0*, 2001.

[19] V. Rajlich and P. Gosavi, "A case study of unanticipated incremental change", *Proc. Software Maintenance,* 2002, pp 442 –451.

[20] D.J. Reifer, "How good are agile methods?", *IEEE Software*, vol.19(2002), no.4, pp 16-18.

[21] G. Ruhe, "Software Engineering Decision Support: Methodology and Applications", In: *Innovations in Decision Support Systems* (Ed. by Tonfoni and Jain), International Series on Advanced Intelligence Volume 3, 2003, pp 143-174.

[22] T.L. Saaty, "*The Analytic Hierarchy Process"*, McGraw Hill, New York, 1980.

Proceedings of the 2003 International Symposium on Empirical Software Engineering (ISESE'03)
0-7695-2002-2/03 $ 17.00 © 2003 IEEE