Software Release Planning: An Evolutionary and Iterative Approach

D. Greer*) and G. Ruhe&)
University of Calgary
2500 University Drive NW
Calgary, AB T2N 1N4, Canada
*)des.greer@qub.ac.uk
&)ruhe@ucalgary.ca

Abstract
To achieve higher flexibility and to better satisfy actual customer requirements, there is an increasing tendency to develop and deliver software in an incremental fashion. In adopting this process, requirements are delivered in releases and so a decision has to be made on which requirements should be delivered in which release. Three main considerations that need to be taken account of are the technical precedences inherent in the requirements, the typically conflicting priorities as determined by the representative stakeholders, as well as the balance between required and available effort. The technical precedence constraints relate to situations where one requirement cannot be implemented until another is completed or where one requirement is implemented in the same increment as another one. Stakeholder preferences may be based on the perceived value or urgency of delivered requirements to the different stakeholders involved. The technical priorities and individual stakeholder priorities may be in conflict and difficult to reconcile. This paper provides i) a method for optimally allocating requirements to increments; ii) a means of assessing and optimizing the degree to which the ordering conflicts with stakeholder priorities within technical precedence constraints; iii) a means of balancing required and available resources for all increments; and iv) an overall method called EVOLVE aimed at the continuous planning of incremental software development. The optimization method used is iterative and essentially based on a genetic algorithm. A set of the most promising candidate solutions is generated to support the final decision. The paper evaluates the proposed approach using a sample project.

Keywords
Incremental Software Development, Release Planning, Requirements Prioritization, Software Engineering Decision Support, Genetic Algorithm.

1. Introduction

Incremental software development addresses the time-to-delivery of software products. Instead of delivering a monolithic system after a long development time, smaller releases are implemented sequentially. If applicable, this approach has many advantages over the traditional waterfall approach. Firstly, requirements can be prioritized so that the most important ones are delivered first and benefits of the new system gained earlier. Consequently, less important requirements are left until later and

so if the schedule or budget is not sufficient the least important requirements are the ones more likely to be omitted. Secondly, it means that customers receive part of the system early on and so are more likely to support the system and to provide feedback on it. Thirdly, being smaller, the schedule/cost for each delivery stage is easier to estimate. Fourthly, user feedback can be obtained at each stage and plans adjusted accordingly. Fifthly, perhaps most importantly, an incremental approach allows for a much better reaction to changes or additions to requirements.

These advantages have particularly been capitalized on in agile methods. Agile methods have in common the idea of release planning. For example, in Extreme Programming [1], a software product is first described in terms of 'user stories'. These are an informal description of user requirements. In the planning process, these stories are prioritized using the perceived value to the user and broken into a series of releases. Based on estimates of how long each story in a increment will take to implement, an iteration plan is developed for delivering that release. Each increment (or release) is a completed product of use to the customer. At any time new stories may be added and incorporated into a future release.

Delivering software incrementally necessitates a process of analysing requirements and assigning them to increments. The typical advice from proponents of incremental delivery is to decide on the increments and then deliver these according to user value [9]. User value may be assessed in terms of cost-benefit calculations or a combination of these and risk assessments [11]. This assumes that requirements are already assigned to increments and only then are the increments ordered. However, often the case is that any given requirement could be delivered in one, several or even all releases. Deciding which increment to deliver requirements in and deciding the order of requirements is a decision that depends on variables that have a complex relationship. Different stakeholder and technical precedence constraints have to be taken into account. Simultaneously, available and required effort for each increment has to be balanced.

There are a number of existing approaches to requirements prioritization. Some have been studied and compared in [17]. Among them is the analytic hierarchy process (AHP), binary search tree creation, greedy-type algorithms and other common sorting methods. In AHP, candidate requirements are compared pair-wise to estimate their relative importance [16]. Most of the algorithms described in [17] need $O(n^2)$ comparisons between the n alternatives. The effort required for this soon becomes prohibitive for a larger number of requirements. In addition to that, none of the mentioned algorithms takes into account different stakeholder perspectives. Release planning including effort constraints is not considered by any of the mentioned algorithms. Finally, the underlying model is fixed and is not allowing any changes in requirements, priorities or constraints (if considered at all).

In this paper we describe an evolutionary and iterative approach called EVOLVE which offers decision support for software release planning. There are five sections. The next section will discuss and formally describe the problem of release planning. Section 3 will present our new solution approach EVOLVE that combines the fundamental ideas of evolution and iteration. A sample case study in Section 4 is used to illustrate and initially validate the practicability of the proposed approach. Section 5 will provide a summary of the findings and identify potential extensions.

2. Problem Statement

2.1. Incremental Software Delivery

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. In the simplest case requirements are fully known and complete at the beginning of the project of incremental software. It is more likely however that, even during the initial increments, the product will evolve so that some requirements will change and new ones will be introduced [24]. In addition to that, priorities and constraints might have changed as well. Figure 1 illustrates the situation.

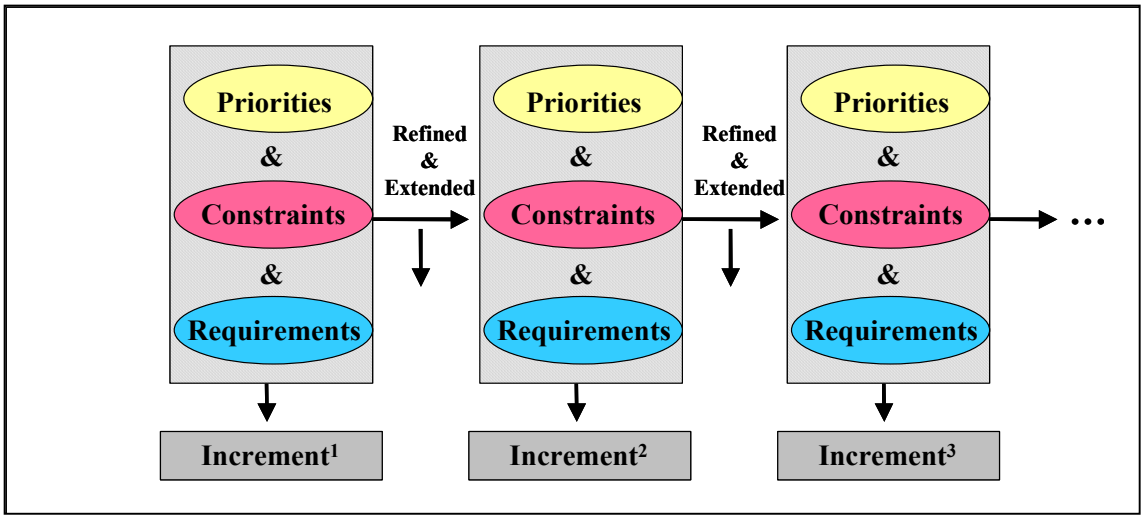

Figure 1: Release planning in a changing environment with open number of releases.

Typically each increment is a complete system that is of value to the client. This means, each new increment can be evaluated by client. The results feed back to the developers, who then take that information into account when implementing subsequent phases. This feedback may introduce changes to requirements or new requirements, priorities, and constraints.

In implementing this model, it is necessary to determine the content and priority of the incremental steps. Gilb suggests that this should be done by 'user value to cost ratio'. User value is, however, a subjective measure and may differ from customer to customer. In the simplest case 'user value' may be monetary benefit, but could also be an increase in product quality, or a reduction in product risk [12] for example. More likely, user value is a combination of many factors and may be different for different stakeholders. This may be manifested in a difference in viewpoint between those who will use the software and the business managers who are commissioning it. The problem is further compounded by the fact that there are also technical implications in the ordering of requirements. Some requirements may have mandatory precedences over other requirements or a condition that they are delivered in the same release.

Release planning for incremental software development includes the assignment of requirements to releases such that all technical and budget constraints are fulfilled. The

overall goal is to incorporate the different stakeholder priorities and the supposed impact on business value in an optimal way. To formulate this problem more formally, we introduce all the necessary concepts and notation.

2.2. Stakeholder Evaluation and Effort Estimates

One of the challenges to the software engineering research community is to involve stakeholders in the requirements engineering process [3]. In any project several stakeholders may be identified. These are likely to have differing priorities. We consider two kinds of evaluation. Both evaluations are on ordinal scale. One input to the process should be a ranking or scoring of requirements according to the perceived value (expected relative impact on business value of the final product). The other one is prioritization according to the degree of urgency (time-criticality) of each requirement to each stakeholder.

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 $prio(r_i, S_p, R^k) \in \{1,2,..,\sigma\}$ to requirement r_i as part of set of requirements R^k at phase k of the planning approach. $prio(r_i, S_p, R^k) = 1$ means highest priority in terms of urgency of requirement $r_i \in R^k$ from the perspective of stakeholder S_p. Typically, different stakeholders have different priorities for the same requirement. The requirements by themselves are assumed to be understandable by all stakeholders and sufficiently detailed to estimate the effort for their implementation.

Similarly, each stakeholder S_p assigns a value denoted by $value(r_i, S_p, R^k) \in \{1,2,..,\partial\}$ to requirement r_i as part of set of requirements R^k at phase k of the planning approach. $value(r_i, S_p, R^k) = 1$ means highest priority in terms of the supposed impact of requirement r_i to the final business value of the final product, taken from the perspective of stakeholder S_p.

2.3 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., $Inc^s \cap Inc^t = \varnothing$ for all s,t $\in$ {k, k+1, k+2, …}. 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, …}.

2.4 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: $(r_i, R^k) \rightarrow \Re^+$ where $\Re^+$ 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 available 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 constraints $\Sigma_{r(i) \in Inc(k)}$ effort(r_i, R^k) $\leq Size^k$ for all increments Inc^k.

2.5 Precedence and Other Dependency Constraints

In a typical real world project, it is likely that some requirements must be implemented before others. There might be logical, technical or resource related 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, technical or resource related reasons that the realization of one requirement must be in place in the same increment as another one. Again, since we are looking at 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 $\omega^k(r_i) = \omega^k(r_j)$.

2.6 Problem Statement for Software Release Planning

At any phase k, 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^* with $\omega^*(r_i) = s \in$ {1,2,….} to increments Inc^s such that

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

(2) $\omega^*(r_i) \leq \omega^*(r_j)$ for all pairs $(r_i, r_j) \in \Psi^k$
(Precedence constraints)

(3) $\omega^*(r_i) = \omega^*(r_j)$ for all pairs $(r_i, r_j) \in \xi^k$
(Coupling constraints)

(4) $A = \Sigma_{p=1\ldots,q}\ \lambda_p[\Sigma_{r(i),r(j) \in R(k)}\ \text{penalty}\ (r_i, r_j, S_p, R^k, \omega^*)] \Rightarrow \min!$
with $\text{penalty}(r_i, r_j, S_p, R^k, \omega^*) :=$

(4.1) 0 if $[\text{prio}(r_i, S_p, R^k) - \text{prio}(r_j, S_p, R^k)][\ \omega^*(r_i) - \omega^*(r_j)] > 0$

(4.2) $\left|\ \text{prio}(r_i, S_p, R^k) - \text{prio}(r_j, S_p, R^k)\right|$ if $\omega^*(r_i) = \omega^*(r_j)$

(4.3) $\left|\ \omega^*(r_i) - \omega^*(r_j)\right|$ if $\text{prio}(r_i, S_p, R^k) = \text{prio}(r_j, S_p, R^k)$

(4.4) $[\text{prio}(r_i, S_p, R^k) - \text{prio}(r_j, S_p, R^k)][\ \omega^*(r_j) - \omega^*(r_i)]$ otherwise

(5) $B = \Sigma_{p=1\ldots,q}\ \lambda_p\ [\Sigma_{r(i) \in R(k)}\ \text{benefit}(r_i, S_p, \omega^*)] \Rightarrow \max!$
with
$\text{benefit}(r_i, S_p, R^k, \omega^*) = [\partial - \text{value}(r_i, S_p, R^k) + 1][\tau - \omega^*(r_j) + 1]$ and
$\tau = \max\ \{\omega^*(r_i): r_i \in R^k\}$

(6) $C(\alpha) = (\alpha - 1)A + \alpha B \Rightarrow \max!$
with $\alpha \in (0,1)$

(7) Determine K best solutions from $C(\alpha_1)$, $C(\alpha_2)$, $C((\alpha_3)$ with
$1 \leq K \leq 10$ and $0 < \alpha_1 < \alpha_2 < \alpha_3 < 1$.

The function (4) is to minimize the total penalties defined as the degree of deviation of the monotonicity property between requirements. Monotonicity property between two requirements is satisfied if one requirement is evaluated more promising than another, and this is true also for the sequence of the assigned increments.

The function (5) is to maximize the total benefit. For a fixed stakeholder, the benefit from the assignment of an individual requirement to an increment is the product of some value difference and some difference in increment numbers. The product is the higher, the earlier the requirement is released and the more impact on final business value is supposed. Finally the overall objective function (6) for one fixed value of α is to maximize a linear combination of (4) and (5). The case of α close to 0 means to give a (strong) priority to stakeholder priorities. In a similar way, α close to 1 means a (strong) priority is given to the achieved benefits of assignment ω^*.

All optimal solutions determined from this approach are known to be non-dominated (Pareto-optimal) [13]. The limitation of this approach is that in case of non-convex problems, only solutions located at the convex hull in the objective space are determined. However, our emphasis is to generate a (small) set of promising solutions from which the decision-maker finally can select. As optimality can't be guaranteed anyway, this limitation is not a real restriction in our case.

To offer a final set of K best solutions, three different values of α are considered. They reflect the different kinds of priorities including a balanced linear combination of the two criteria. The actual number K depends of the concrete problem. Typically, it will not more than ten to provide an overview of the existing (most promising) solutions. Both K and the individual values of α are supposed to be determined by the actual decision-maker.

3. Solution Approach EVOLVE

3.1. Genetic Algorithms

Genetic algorithms have arisen from an analogy with the natural process of biological evolution [14]. They are particularly well suited to NP-complete problems that cannot be solved by deterministically polynomial algorithms. One commonly discussed problem area to which genetic algorithms have been applied is the Travelling Salesman Problem (TSP) [4]. It has been empirically shown that genetic algorithms can generate high quality solutions being optimal or near optimal even for large-scale problems. In the area of software engineering, this approach was successfully applied by [2] to devise optimal integration test orders.

Genetic algorithms maintain a population of solutions or chromosomes. The 'optimal' population size is a matter for debate. Some have suggested higher populations [10] while others indicate that population sizes as low as thirty are adequate [21]. A large population size improves the probability of obtaining better solutions and so should speed up the optimization process, although this is at the expense of computation time. Each member of the population receives a fitness measure, i.e., the value of the objective function. This measure relates to how good the chromosome is at solving the stated problem.

Main operations applied to chromosomes of the population are selection, crossover, and mutation. Selection is effected by choosing two parents from the current population, the choice being determined by relating the fitness score to a probability curve [19]. In the case of the 'order method' of the Palisade RiskOptimizer tool [20], as used in this research, the crossover operator takes two parents, randomly selects items in one parent and fixes their place in the second parent (for example, items B and D in Figure 2). These are held in position but the remaining items from the first parent are then copied to the second parent in the same order as they were in originally. In this way some of the sub-orderings are maintained.

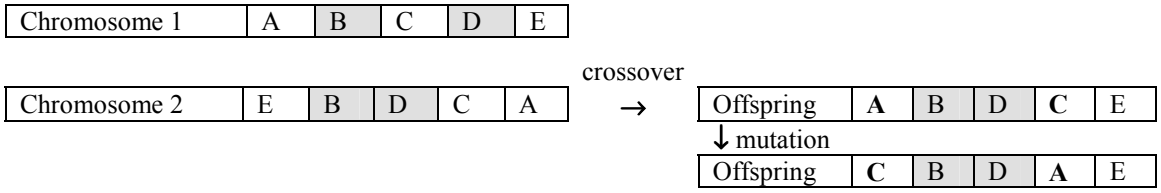

Figure 2: Illustration of crossover and mutation operators.

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. Since the values in the chromosome must remain constant, the normal approach to mutation where one or more variables are randomly changed will not work. Hence, in the order method as provided by the tool, mutation is effected via random swapping of items in the new offspring. The number of swaps is proportional to the mutation rate.

An example is shown in Figure 2 for the items A and C. The new offspring is ranked in the current population and the bottom ranked chromosome is discarded. Hence

the population size retains a steady state. The extent of mutation is controlled by the parameter mutation rate. The choice of 'best' mutation and crossover rates is sensitive to the type of problem and its characteristics [15].

At each generation, members of the population are assessed for fitness. Frequently in using genetic algorithms this fitness refers to a cost function that has to be minimised or a payoff function that should be maximized. The processes of evaluation, selection, crossover and mutation continue, and the net effect is a gradual movement towards higher fitness scores in the population. Since genetic algorithms operate on a population rather than a single entity, the possibility of becoming stuck at local optima is reduced. The choice of when to terminate the algorithm may be determined by a predefined number of iterations, a preset elapsed time or when the overall improvement becomes negligible.

3.2 Proposed Approach

The proposed approach called EVOLVE combines the computational strength of genetic algorithms with the flexibility of an iterative solution method. At each iteration, a genetic algorithm is applied to determine an optimal or near-optimal (related to the objective function (6)) assignment of requirements. Only assignments satisfying constraints (1) to (3) are considered.

Maximization of objective function (6) is the main purpose of conducting crossover and mutation operations. This function composed of (4) and (5) is computed at 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 best fitness score achieved within 0.5% deviation over six hundred simulations.

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 'firm' and all subsequent ones Inc^{k+2}, $Inc^{k+3,}$ …as tentatively again. This is illustrated in Figure 3. For all iterations, there is the next 'firm' increment, the one that will certainly be implemented in that iteration (solid border in Figure 3). There may also be other 'tentative' increments (dashed border), representing plans for future iterations. For iterations following the first one, implemented increments are shown greyed.

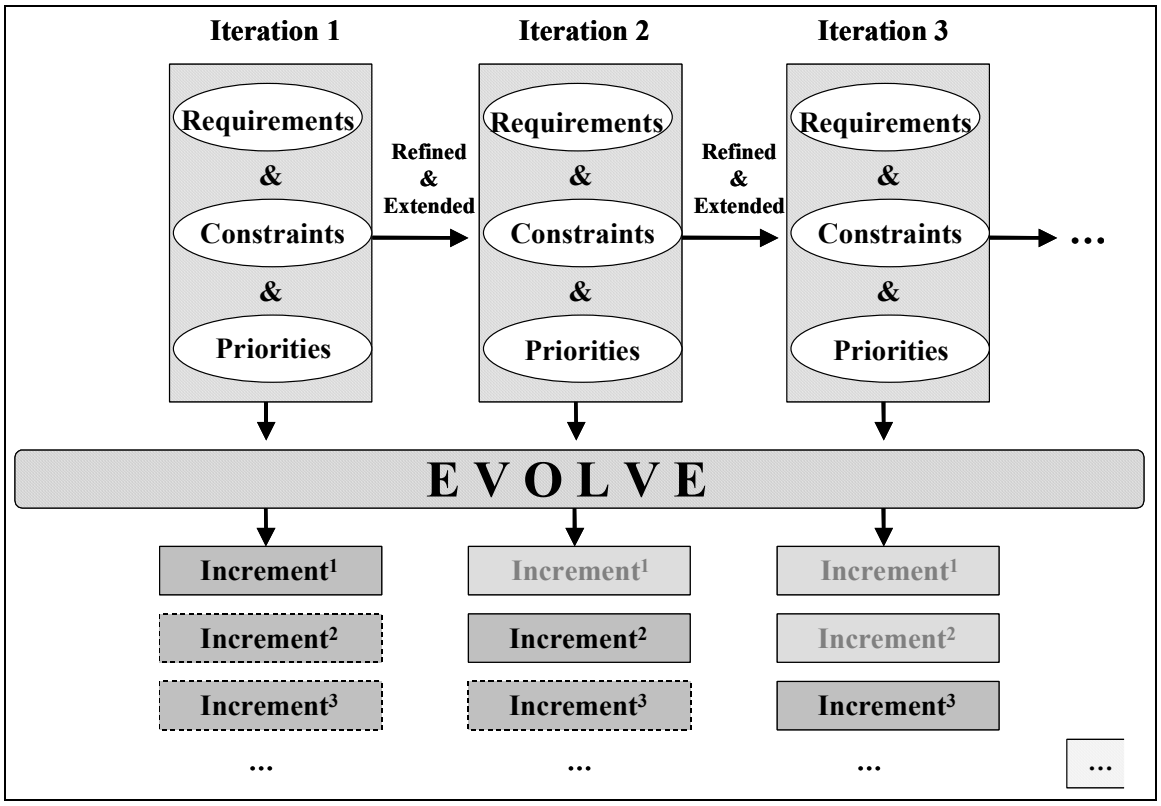

Figure 3. EVOLVE approach to assignment of requirements to increments.

The above process can be considered as infinite, i.e., the number of iterations (and the number of increments) has not to be defined upfront. This is considered to be the default situation for the formulation of the model. However, it is also possible that a fixed number of iterations is taken into account from the very beginning. The EVOLVE approach remains completely the same, with the only difference that there are some minor modifications in the computation of (6). If the number of releases is limited, then it might be possible that some requirements are excluded by a given solution. In other words, the increment number assigned to a requirement will be greater than the maximum number of increments. In such cases the stakeholder priority penalty is multiplied by a user-specified factor larger than one. This is to reflect the negative impact on the stakeholder concerned if that requirement is omitted from the project.

Solutions generated by EVOLVE are optimal or near optimal. Genetic algorithms, in general, cannot guarantee optimality. There is a great variance in the solutions generated by different runs of the solution algorithm. This variation concerns both changes in mutation and crossover rate. But even for fixed mutation and crossover rates, different solutions are obtained if the algorithms are applied several times. However, as known from empirical evaluations, there is a great likelihood to have the optimal solution among the ones generated, especially if a larger number of computations with varying parameters are conducted

3.3 Algorithms and Tool Support

In this research, we have made use of Palisade's RiskOptimizer tool [20]. 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 [6].

To ensure feasibility (1) in terms of available effort, a greedy-like procedure was applied. Original precedence (2) and coupling constraints (3) are implemented by specific rules used to check each generated solution. This is achieved via a table of pairs of requirements. In both cases, if any given solution is generated that violates either category of constraint, the solution is rejected and a backtracking operation is used to generate a new solution.

Weightings are used to discriminate between stakeholders. To allow greater flexibility and objectivity, we assume that these weightings are calculated using the pair-wise comparison method from AHP [23].

A further feature made us of in this research is the enforcement of constraints on the organisms generated. Organisms generated outside the solution space undergo a backtracking process, where the tool reverts to one of the parents and retries the crossover and mutation operations until a valid child is obtained. A summary description of the evolutionary algorithm is provided in the Appendix.

4. Case Study

4.1. Description of Sample Project

To demonstrate practicability of the approach, we study a sample software project with twenty requirements, e.g. $R^1 = \{r_1,\ldots,r_{20}\}$. The technical precedence constraints in our typical project are represented by partial order Ψ (compare Section 2.5) as shown below. This states that r_4 must come before r_8 and r_{17}, r_8 before r_{17}, and so on.

$$\Psi^1 = \{(r_4,r_8), (r_4,r_{17}),(r_8,r_{17}),(r_9,r_3),(r_9,r_6),(r_9,r_{12}),(r_9,r_{19}),(r_{11},r_{19})\}$$

Further, some requirements were specified to be implemented in the same increment as represented by binary relation ξ, as defined in Section 2.5. This states that r_3 and r_{12} must be in the same release, as must r_{11} and r_{13}.

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

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. In general, effort sizes may be different for different increments.

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_3 and r_7). However, the judgement is more conflicting in other cases (as for r_{13} and r_{12}).

Table 1: Sample Stakeholder-assigned Priorities

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

In a similar way, each stakeholder assigns a value to each requirement based on the contribution to the business value of the product by that requirement. Table 2 demonstrates this. Again, in some cases there is general agreement (as for r_8 and r_{19}), and others more conflict (as for r_{17} and r_{18})

Table 2: Sample stakeholder assigned values.

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

The stakeholders S_1 to S_5 were weighted using AHP by pair-wise comparison from a global project management perspective with results as shown in Table 3. The stakeholder weightings λ = (0.211, 0.211, 0.421, 0.050, 0.105) are computed from the eigenvalues of the matrix shown in Table 3: The technique of averaging over normalized columns [23] can be used to approximate the eigenvalues.

Table 3: Matrix of pair-wise comparison of stakeholders on a nine-point scale of AHP.

		Stakeholder				
		S_1	S_2	S_3	S_4	S_5
Stakeholder	S_1	1	1	½	4	2
	S_2	1	1	½	4	2
	S_3	2	2	1	8	4
	S_4	¼	¼	1/8	1	½
	S_5	½	½	¼	2	1

4.2. Implementation

We used the default population size of fifty for the genetic algorithm. Having assigned the increments, the precedence and coupling constraints are checked, and the current solution is only accepted if these are met. Precedence and coupling constraints being met, the penalty score is calculated using the stakeholder assigned priorities as shown in Table 1. This is executed by (automatically) pair-wise comparing the priorities of each requirement r_i for stakeholder S_i, with reference to their increment assignment $\omega(r_i)$. The benefit calculation for a given stakeholder and a given requirement is calculated using (5).

As the crossover and mutation operations are performed the best solutions are kept and eventually, when no further improvement is detected, a solution such as that shown in the lower half of Figure 4 is produced. The method can be conducted using a chosen crossover rate and mutation rate, or with a range of crossover rates and mutation rate combinations. We also suggest that solutions are produced for various values of α. This parameter has relevance in determining the bias given towards the benefit function (high value of α) or toward the penalty function (low value of α). Moreover, a set of solutions is composed out of that such that the decision-maker can finally decide according to some additional (subjective) criteria.

Because there is little guidance in the literature regarding the most appropriate crossover rate or mutation rate for this type of problem, preliminary experiments were carried out using a range of crossover rates between 0.1 and 1 in steps of 0.1. Similarly, a range of mutation rates were tried: 0.05, 0.1, 0.15, 0.2, 0.25 and 0.3 as well as the built in 'auto' mutation rate. Auto-mutation automatically increases the mutation rate gradually in an attempt to improve the solution and in practice this was shown to be adequate and so the tool was programmed to carry out the optimization process using all of the above crossover rates with auto-mutation. In keeping with the recommendations in the literature and in the tool documentation, we maintained a population of fifty organisms.

4.3. Sample Project Results

The initial solution for assigning the 20 requirements of the set R^1 and the result of a sequence of optimization steps is shown in Figure 4. Choosing the value α in (6) as $\alpha = 0.5$ results in an improvement of the objective function (7) from –41.8 to +23.0. $\alpha = 0.5$ corresponds to an equal balance between reduction of penalties and achieving maximum benefit.

Starting (arbitrary) solution

GA Rank	1	2	3	4	5	6	7	8	9	10	11	12	13	*15*	*14*	16	17	18	19	20
Requirement	r_5	r_4	r_{18}	r_2	r_7	r_8	r_{14}	r_{12}	r_3	r_{15}	r_1	r_9	r_{20}	r_6	r_{19}	r_{11}	r_{10}	r_{16}	r_{13}	r_{17}
Effort Estimate	4	3	4	4	10	2	2	5	2	1	1	1	4	7	8	2	3	4	8	10
Release	**1**								**2**						**3**					**4**
																			Benefit	163.0
																			Penalty	246.6
																			Total Objective Function Value at α =0.5	-41.8

↓ n optimization steps via selection, crossover and mutation.

Optimized Solution

GA Rank	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Requirement	r_{20}	r_4	r_{18}	r_{16}	r_7	r_{11}	r_{12}	r_{10}	r_9	r_{13}	r_2	r_3	r_{19}	r_{17}	r_1	r_8	r_5	r_6	r_{14}	r_{15}
Effort Estimate	4	3	4	4	10	2	5	3	1	8	4	2	8	10	1	2	4	7	2	1
Release	**1**								**2**				**3**					**4**		
																			Benefit	168.0
																			Penalty	121.7

Total Objective Function Value at α =0.5	23.1

Figure 4: Solution generated from EVOLVE.

The optimization results for a single iteration are shown in Table 4. This shows the best three values of the objective function for C(0.2), C(0.5), and C(0.8), respectively.

Table 4: Top three solutions obtained for C(0.2), C(0.5), and C(0.8).

	α = 0.2			**α = 0.5**			**α = 0.8**		
Rank:	1	2	3	1	2	3	1	2	3
Penalty (A)	121.7	129.3	132.3	121.7	132.3	142.9	132.3	150.5	141.9
Benefit (B)	167.9	166.8	173.0	168.0	171.6	165.5	171.7	176.1	172.7
Objective Function Value (C):	-63.8	-70.1	-71.3	23.1	19.7	11.3	110.9	110.8	109.8
C Adjusted to α = 0.5	**23.1**	**18.7**	**20.3**	23.1	**19.7**	**11.3**	19.7	**12.8**	**15.4**
Increment 1 (next)	r_4 r_7 r_{16} r_{18} r_{20}	r_4 r_7 r_{16} r_{18} r_{20}	r_4 r_7 r_{16} r_{18} r_{20}	r_4 r_7 r_{16} r_{18} r_{20}	r_4 r_7 r_{16} r_{18} r_{20}	r_4 r_{11} r_{13} r_{16} r_{18} r_{20}	r_4 r_7 r_{16} r_{18} r_{20}	r_4 r_{11} r_{13} r_{16} r_{18} r_{20}	r_4 r_7 r_{16} r_{18} r_{20}
Increment 2 (proposed)	r_2 r_3 r_9 r_{10} r_{11} r_{12} r_{13}	r_2 r_9 r_{11} r_{13} r_{14} r_{19}	r_1 r_2 r_3 r_9 r_{11} r_{12} r_{13} r_{14}	r_2 r_3 r_9 r_{10} r_{11} r_{12} r_{13}	r_1 r_2 r_3 r_9 r_{11} r_{12} r_{13} r_{14}	r_7 r_8 r_{10} r_{17}	r_1 r_2 r_3 r_9 r_{11} r_{12} r_{13} r_{14}	r_1 r_2 r_3 r_7 r_9 r_{12} r_{14}	r_3 r_8 r_9 r_{10} r_{11} r_{12} r_{13} r_{14}
Increment 3 (proposed)	r_1 r_5 r_8 r_{17} r_{19}	r_1 r_3 r_5 r_8 r_{12} r_{15} r_{17}	r_5 r_8 r_{15} r_{17} r_{19} r_1	r_1 r_5 r_8 r_{15} r_{17} r_{19}	r_5 r_8 r_{15} r_{17} r_{19}	r_1 r_2 r_3 r_5 r_9 r_{12} r_{19}	r_5 r_8 r_{15} r_{17} r_{19}	r_5 r_8 r_{15} r_{17} r_{19}	r_1 r_2 r_{15} r_{17} r_{19}
Increment 4 (proposed)	r_6 r_{14} r_{15}	r_6 r_{10}	r_6 r_{10}	r_6 r_{14} r_{15}	r_6 r_{10}	r_6 r_{14} r_{15}	r_6 r_{10}	r_6 r_{10}	r_5 r_6

Table 4 shows that changing values of α also changes the related best solutions. As α is increased the bias towards the benefit part of the objective function is increased. Hence with many of the solutions for α = 0.8, we find more of the higher valued requirements early in the increment plan, despite the fact that they are "out of sequence" in some stakeholder's views. In practice, the decision-maker can choose between these solutions based solely on the data presented, or with additional domain knowledge. The solutions shown in Table 4 contain seven unique permutations. These can be presented to

the project manager. The benefit function can be recalculated to the desired α for comparison purposes. In this case we have standardised all of the solutions using α=0.5.

At this stage, it is possible that the requirements for the overall project have changed. To simulate this, requirements r_{21}, r_{22} and r_{23} were added and r_{14}, r_{10} and r_{15} deleted. In a third iteration r_{24} and r_{25} were added. It is also possible to adjust the effort estimates between stages. The process of moving from the first iteration to the second and third iterations for the sample project is illustrated in Figure 5.

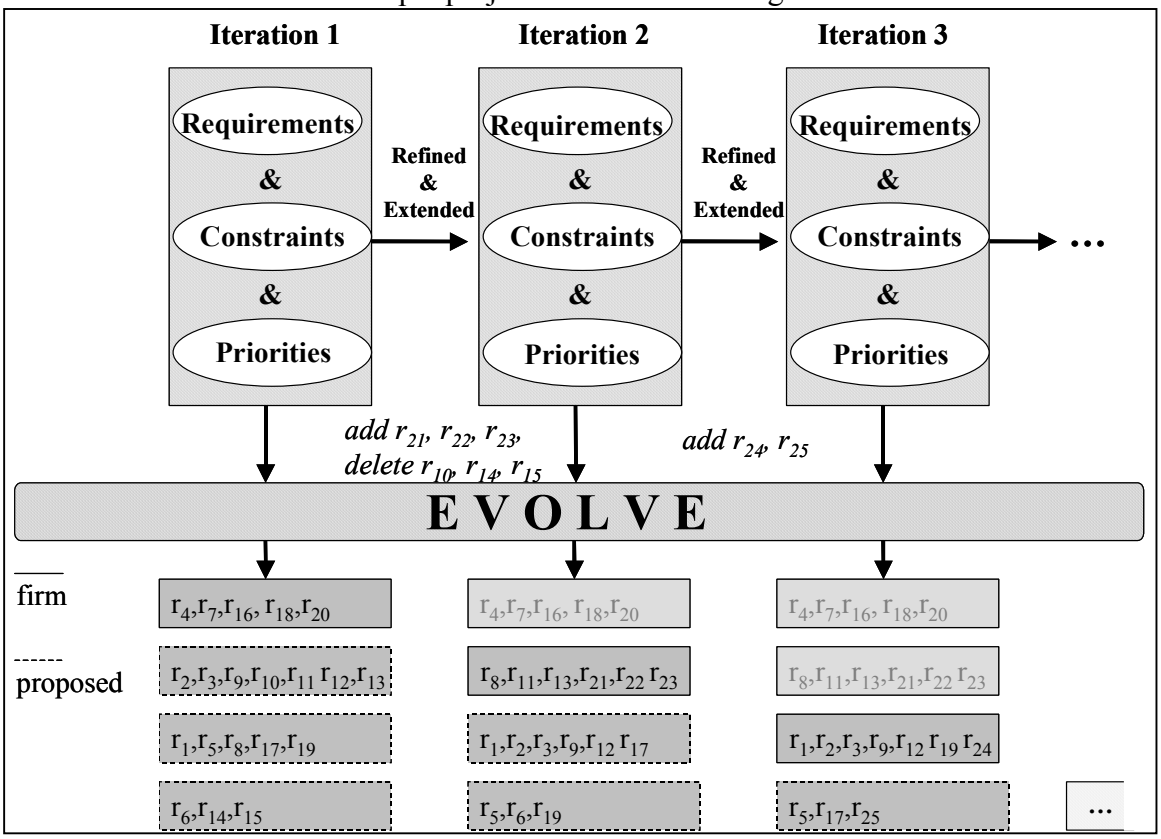

Figure 5: Results from sample project release planning – iterations 1, 2 and 3.

We studied algorithm performance and the choice of genetic algorithm parameters. We did not set out to investigate this aspect but our observations show that good results can be obtained from a wide range of crossover and mutation rates combinations. To test the consistency of the method some experimentation was carried out over ten optimizations using all combinations of crossover rates between 0.1 and 1 in steps of 0.1 and mutation rates from 0.05 to 0.3 in steps of 0.05. Overall, the results from this illustrated that, since there is no identifiable relationship between crossover rate and/or mutation rate and the result obtained, it should be possible to obtain a good result with a narrower range of parameters, say crossover rates from 0.6 to 0.9 and mutation rates from 0.15 to 0.25. This would dramatically cut the execution time for EVOLVE, if that was an issue.

Finally, we studied the frequency of the backtracking operation when an illegal chromosome (violating one of the constraints (1) to (3)) is detected. The dependency of the relative frequency of backtracking operations in dependence of the number of iterations is shown in Figure 6.

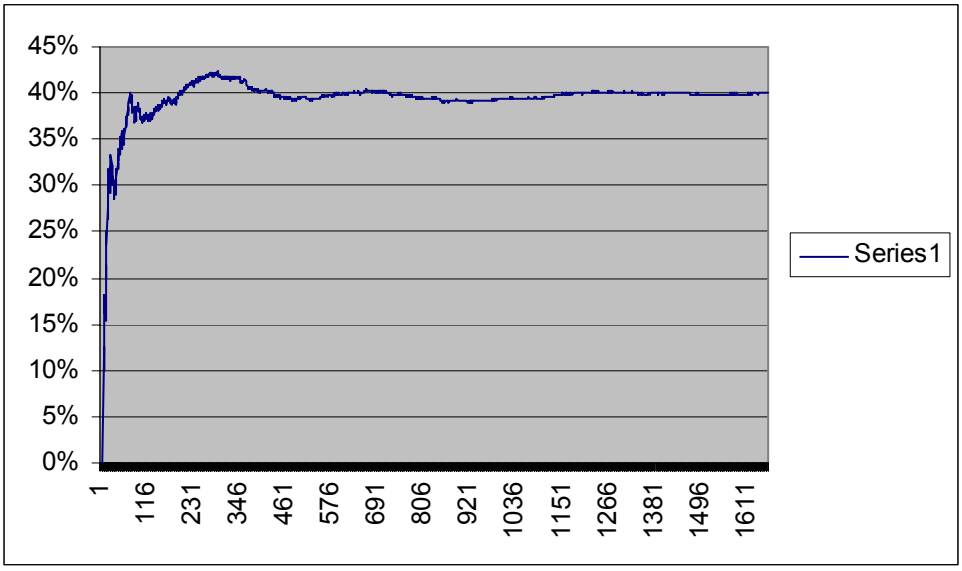

Figure 6: Relative frequency of backtracking operations in dependence of number of optimizations.

5. CONCLUSIONS

The idea of offering decision support always arises when decisions have to be made in complex, uncertain and/or dynamic environments. Most of the related problems are very complex including different stakeholder perspectives and constraints. Currently, there is an increasing effort to not only measure or model certain aspects of the development processes, but to go further and integrate all available data, information, knowledge and experience with a sound methodology to provide the backbone for making good decisions [22]. This mainly includes searching for all the objectives and constraints that influence a decision as well as elaborating the so defined solution space for possible courses of action.

Generation of feasible assignments of requirements to increments in a changing environment taking into account different stakeholder perspectives and even resource constraints is a very important but complex task. Our solution approach EVOLVE generates a typically small set of most promising candidate solutions from which the actual decision-maker can choose from. The emphasis of decision support is on support, not on actually making the decision. In the real world, additional and most recent influencing factors or constraints are taken into account in making the decision. This is achieved best through having a set of K best solutions [13].

The main contribution of the paper is a new and powerful solution method for software release planning called EVOLVE. Given a set of requirements with their effort estimations and a their categorisation into priorities by representative stakeholders, the method uses a genetic algorithm to derive potential release plans within pre-defined technical constraints. The new approach has several advantages over existing methods:

1. EVOLVE takes into account stakeholder priorities as well as effort constraints for all releases.

2. EVOLVE assumes that software requirements are delivered in increments. This is becoming more and more important as the realisation that software delivery in this fashion offers a substantial risk reduction. Making this assumption changes the prioritization problem to one of ranking and subset selection, rather than just ranking.
3. EVOLVE considers inherent precedence and coupling constraints. Existing approaches do not cater for dependencies between requirements. The facts that EVOLVE uses a genetic algorithm means that the final release plan arises from a population of solutions. This allows those solutions that break constraints to be disallowed without deterioration of the method.
4. EVOLVE offers greater flexibility by allowing changes in requirements, constraints and priorities. This better matches the reality of most software projects.
5. EVOLVE recognizes that stakeholders have priorities for requirements that may be conflicting. Other methods can also cope with this, but EVOLVE recognizes that there is a negative impact penalty of delivering requirements in a sequence contrary to a stakeholder's priority and a positive benefit of delivering high priority requirement earlier. EVOLVE optimizes the solution to balance the stakeholder desires to deliver high priority solutions earliest and to have certain requirements delivered before others. Further, as in real-world situation, not all stakeholders are treated equally, so that the effect of their input can be weighted.
6. EVOLVE approaches software release planning from a decision support perspective. This means, a set of most promising candidate solutions is generated. The decision maker has finally to choose one by considering most recent and also implicit constraints and context factors in addition to the original problem formulation.

Overall, the novelty of the approach is found in these advantages and also the fact that a genetic algorithm has been used to solve the problem. Our findings indicate that genetic algorithms are easy to apply and effective for this type of problem. Further work will involve applying the method in a more complex industrial setting and obtaining feedback on its operational aspects and effectiveness. It is also planned to introduce the possibility of uncertainty and risk into the model, particularly in terms of the effort estimations and constraints. Further, a future web-based version of the method is envisioned, aimed at release and version planning in a maintenance situation.

Acknowledgements

The authors would like to thank the Alberta Informatics Circle of Research Excellence (iCORE) for their financial support of this research. Des Greer is a visiting researcher from Queens University, Belfast and their support is acknowledged. Many thanks are due also to Mr Wei Shen for conducting numerical analysis using RiskOptimizer and Dietmar Pfahl for supporting discussions.

References

[1] Beck, K., "Extreme Programming Explained", Addison Wesley, 2001.

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

[3] Bubenkbo, J. A. jr., "Challenges in Requirements Engineering", Proceedings of the. 2nd IEEE symposium on Requirements Engineering, pp 160-162, IEEE Computer Society, 1995.

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

[5] Cockburn, A., "Agile Software Development", Pearson Education, 2002.

[6] Davis, Lawrence, "Handbook of Genetic Algorithms", Van Nostrand Reinhold, New York, 1991.

[7] De Jong, K. A., "An analysis of the behaviour of a class of genetic adaptive systems". PhD thesis, University of Michigan, 1975.

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

[9] Gilb, T., "Principles of Software Engineering Managemen", Addison-Wesley, 1988.

[10] Goldberg, D.E., "Sizing populations for serial and parallel genetic algorithms", Proc. 3rd Intl. Conf. On Genetic Algorithms, ed. Shafer, J.D., Morgan Kaufman, Los Altos, CA, USA, 1989.

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

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

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

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

[15] Haupt R.L. Optimum population size and mutation rate for a simple real genetic algorithm that optimizes array factors. IEEE Antennas and Propagation Society International Symposium. Part vol.2, 2000, pp.1034-7 vol.2. Piscataway, NJ, USA

[16] Karlsson, J., "Software requirements prioritizing", Proceedings of the Second International Conference on Requirements Engineering, pp 110 –116, 1996

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

[18] McConnell, S., "Code Complete", Microsoft Press, Redmond, WA, 1993.

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

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

[21] Reeves, C.R., "Modern Heuristic Techniques for Combinatorial Problems", McGraw-Hill, Maidenhead, UK, 1995.

[22] Ruhe, G., "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.

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

[24] Wang, Q. and Lai, X., Proc. "Requirements Management for the Incremental Development Model", 2nd Asia-Pacific Conference on Quality Software, pp 295-301, 2001.

Appendix

This appendix presents a summary of the genetic algorithm used in EVOLVE.

Input:

S_{seed} = Initial seed solution
m = population size
cr = crossover rate
mr = mutation rate

Output:

The solution with the highest fitness score from the final population

Variables:

S_n =A Solution
P =current Population as a set of (Solution, fitness score) pairs = $\{(S_1,v_1), (S_2, v_2)\ldots.(S_m,v_m)\}$
$S_{parent1}$ = first parent selected for crossover
$S_{parent2}$ = second parent selected for crossover
$S_{Offspring}$ = result from crossover/ mutation operation

Functions:

NewPopulation(S_{seed},m): $S_{seed} \rightarrow P$, Returns a new population of size m.

Evaluate(S) provides a fitness score for a given solution, S.

Select(P) chooses from population P, based on fitness score, a parent for the crossover operation.

Crossover(S_i,S_j,cr) performs crossover of solutions S_i and S_j at crossover rate cr.

Mutation(S_i, mr) performs mutation on solution S_i at mutation rate mr.

IsValid(S_i) checks validity of solution S_i against the user-defined constrraints

BackTrack($S_{offspring}$) = proprietary backtracking operation on a given solution. This backtracks towards the first parent until a valid solution is created or a user-defined number of backtrack operations is reached.

Cull(P) removes the $(m+1)^{th}$ ranked solution from the population, P.

CheckTermination() is a Boolean function which checks if the user's terminating conditions have been met. This may be when a number of optimizations have been completed, when there has been no change in the best fitness score over a given number of optimizations, a given time has elapsed or the user has interrupted the optimization.

Max(P) returns the solution in population P that has the highest fitness score.

***Algorithm*:**

```
BEGIN
        P := NewPopulation(seed);
        TerminateFlag := FALSE;
        WHILE NOT (TerminateFlag)
            BEGIN
                S_parent1 := Select(P);
                S_parent2 := Select(P/ S_parent1);
                S_Offspring := Crossover(S_parent1, S_parent2, cr);
                S_Offspring := Mutation(S_Offspring, mr);
                If NOT IsValid(S_Offspring) THEN BackTrack(S_Offspring);
                IF IsValid(S_Offspring)
                        BEGIN
                                P := P ∪ {(S_Offspring, Evaluate(S_offspring)}};
                                Cull(P);
                        END;
                TerminateFlag = CheckTermination();
        END;
        RETURN(Max(P));
END.
```