

Chapter VII

Requirements Prioritisation for Incremental and Iterative Development

D. Greer

Queens University Belfast, UK

Abstract

The problems associated with requirements prioritisation for an incremental and iterative software process are described. Existing approaches to prioritisation are then reviewed, including the Analytic Hierarchy Process, which involves making comparisons between requirements and SERUM, a method that uses absolute estimations of costs, benefits, and risks to inform the prioritisation process. In addition to these, the use of heuristic approaches is identified as a useful way to find an optimal solution to the problem given the complex range of inputs involved. In particular genetic algorithms are considered promising. An implementation of this, the EVOLVE method, is described using a case study. EVOLVE aims to optimally assign requirements to releases, taking into account: (i) effort measures for each requirement and effort constraints for each increment; (ii) risk measures for each requirement and risk limits for each increment; (iii) precedence constraints between requirements (where one requirement must always be in an earlier or the same increment as another); (iv) coupling constraints between requirements (where two or more must be in the same increment); and (v) resource constraints (where two or more requirements may not be in the same increment due to using some limited resource). The method also handles uncertainty in the effort inputs, which are supplied as distributions and simulated

using Monte Carlo simulation before carrying out the genetic algorithm operations. In addition to handling uncertainty, EVOLVE offers several advantages over existing methods since it handles a large range of factors. The overall implementation of the method allows the inputs to be changed at each iteration, and so better fits reality where requirements are changing all the time.

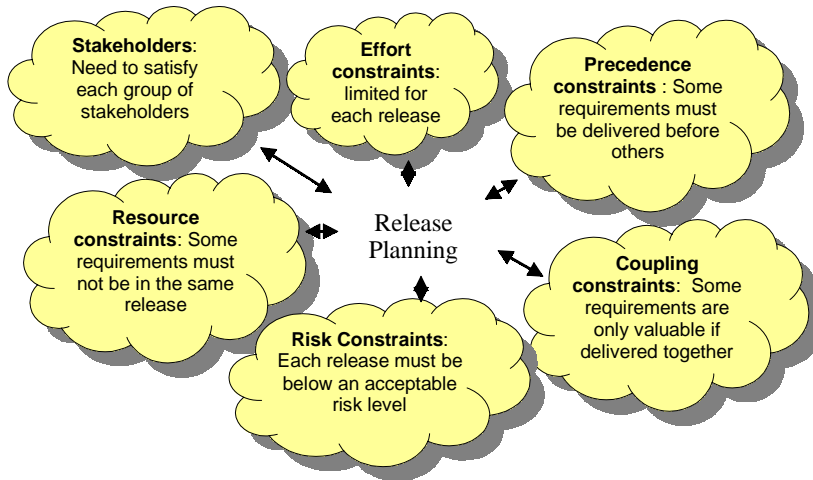
Introduction

In any given project, requirements arise from various stakeholders. These stakeholders may be users, developers, project managers, business managers, or other categories of people affected by the system. In the case of *new* software applications, there are typically a large number of requirements, some of which are essential, others desirable, and some relatively unimportant. For *existing* applications, there will be a backlog of new requirements, potential fixes, and enhancements, again with differing priorities. In both cases it is usually impractical to implement all requirements simultaneously because of the cost involved, staff limitations, and market or user pressures to have the software implemented. Thus some form of prioritisation is necessary. The output of this process will depend on the effort required for each requirement against the overall effort available, the value arising out of delivering certain requirements at a given time, the risks incurred by delivering or not delivering given requirements, and on dependencies between requirements. In addition to this we have the preferences of business and developer stakeholders, who may be of different levels of importance and may be geographically dispersed with different viewpoints about what should be delivered and when. For some of these stakeholders a given requirement may be essential to the success of the product; others may believe that it might even damage the success of the product. In between these extremes some stakeholders may hold the view that a requirement is unimportant but hold no objection to it being included (Davis, 2003). Overall this is a complex problem, which is very difficult to solve to the satisfaction of all concerned. Management of requirements including priority assignment has been identified as a key success factor for commercial enterprises (De Gregorio, 1999). In the case of bespoke software development, there may be a small number of stakeholders, but in the case of commercial off-the-shelf software there may be hundreds or thousands of stakeholders (Regnell, Host, Natt och Dag, Beremark, & Hjelm, 2001).

Problem Discussion

In recent years there has been an increasing recognition that an incremental approach to software development is often more suitable and less risky than the traditional waterfall approach (Greer, Bustard, & Sunazuka, 1999a). This preference is demonstrated by the current popularity of agile methods, all of which adopt an incremental approach to delivering software rapidly (Cockburn, 2002). This shift in paradigm has been brought

Figure 1. Complexity of software release planning



about by many factors, not least the rapid growth of the World Wide Web, the consequent economic need to deliver systems faster (Cusamano & Yoffie, 1998), and the need to ensure that software better meets customer needs. Hence, we will concentrate on this process model in our problem discussion.

Planning and developing software is a very complex venture. As illustrated in Figure 1 stakeholder preferences, effort constraints, dependency constraints, resource constraints, and risk constraints all contribute to the complexity.

In developing computer-based systems, the inputs of all stakeholders should be taken into account during the planning and development processes (Hart, Hogg, & Banerjee, 2002). In prioritising requirements, similarly, the viewpoints of all affected stakeholders must be taken into account (Bubenkbo, 1995). This is already a complicated process if there are several stakeholders with diverging viewpoints (Jiang, Klein, & Discenza, 2002). However there are other important factors such as the available effort for a given system or release versus the effort required. There are also likely to be dependencies between requirements. This may be due to the fact that certain requirements must be in place before others. It could also be that two or more requirements must be delivered together or indeed that they should not be delivered together in a certain timeframe due to some resource constraint. Other factors are the level of risk the organisation is exposed to in a given system or a given release (Charette, 1989). The nature of the software process is also relevant to the problem. In what follows we will assume that whatever the process, it will result in individual releases. This is true even of the waterfall model, where the intention is to deliver a complete system, but there will be subsequent releases in the maintenance phase to cover deferred requirements, errors, and enhancements (Rajlich & Gosavi, 2002).

Stakeholders Viewpoints and Increment Planning

A software system at any stage of its life can be described by a set R^1 of requirements, that is, $R^1 = \{r_1, r_2, \dots, r_n\}$. Using an incremental approach, at the first stage ($k=1$) a set of requirements are planned for delivery as Inc^1 . At subsequent stages ($k>1$), new requirements are added and others are removed and a new subset of requirements is planned for delivery in the next increment Inc^2 . This continues for all increments, Inc^k .

Suppose there are q Stakeholders S_1, S_2, \dots, S_q who have been assigned a relative importance between 0 and 1 by a project manager. Each stakeholder S_p assigns a priority, $prio(r_i, S_p, R^k)$ to each requirement r_i in the set R^k at phase k . $Prio()$ is a function: $(r_i, S_p, R^k) \rightarrow \{1, 2, \dots, \sigma\}$, performed by each stakeholder, S_p , where σ is the maximum priority score that can be assigned. Using the scheme suggested in Davis (2003), one practical interpretation this might be where $\sigma = 5$. In this scenario, $prio(r_1, S_1, R^1) = 5$ means that for stakeholder S_1 the first release will be useless without r_1 ; $prio(r_1, S_1, R^1) = 4$ means that the requirement should be included in R^1 ; $prio(r_1, S_1, R^1) = 3$ means that S_1 is neutral on the issue for r_1 in R^1 ; $prio(r_1, S_1, R^1) = 2$ means that the requirement should be excluded from R^1 ; $prio(r_1, S_1, R^1) = 1$ means that if r_1 is included in R_1 , then the release will not be useful to S_1 .

Thus the output from the release planning process is a definition of increments $Inc^k, Inc^{k+1}, Inc^{k+2}, \dots$ with $Inc^t \subset R^k$ for all $t = k, k+1, k+2, \dots$. The different increments are disjointed, that is, $Inc^s \cap Inc^t = \emptyset$ for all $s, t \in \{k, k+1, k+2, \dots\}$. The unique function ω^k assigns each requirement r_i of set R^k the number s of its increment Inc^s , i.e., $\omega^k: r_i \in R^k \rightarrow \omega^k(r_i) = s \in \{k, k+1, k+2, \dots\}$.

Effort Constraints

Effort estimation can be described as a function assigning each pair (r_i, R^k) an effort estimate, $effort()$ that is, $effort()$ is a function $(r_i, R^k) \rightarrow \mathfrak{R}^+$ where \mathfrak{R}^+ is the set of positive real numbers. These effort estimates are specific to a particular phase R^k , since efforts may be re-estimated following any increment. It is likely that the effort available for a given increment Inc^k is limited to a fixed value $Size^k$. Hence the sum of the effort estimated for all requirements in a planned increment must be within this limit. Thus $\sum_{r(i) \in Inc(k)} effort(r_i, R^k) \leq Size^k$ for all increments $Inc(k)$.

Dependency Constraints

Any set of requirements will contain dependencies wherein one requirement must always be before or after another. Thus for all iterations k there is a partial order Ψ^k on the product set $R^k \times R^k$ such that $(r_i, r_j) \in \Psi^k$ implies $\omega^k(r_i) \leq \omega^k(r_j)$. In an incremental approach, having a constraint that a given requirement is before some other is also met if they are in the same increment.

Similarly, there may be certain requirements that are only valuable if delivered together in the same increment. Thus 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 .

Additionally there are resource constraints to consider, where certain combinations of requirements may not be delivered in the same increment. This may be due to the fact that they share some limited resource. Hence resource(t) represents the resource capacity of t. In this case, there are index sets $I(t) \subset \{1, \dots, n\}$ such that $\text{card}(\{r_i \in R^k: i \in I(t)\}) \leq \text{resource}(t)$ for all releases k and for all resources t.

Risk Constraints

All development activities have associated risks, and a development team may wish to limit the extent of exposure to risk in a given increment. We define risk as the likelihood that some loss will be incurred in implementing a requirement due to some event and introduce a risk estimate for each requirement. Thus 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; that is, risk is a ratio scaled function $\text{risk}: (r_i, R^k) \rightarrow [0,1)$, where '0' means no risk at all and '1' stands for the highest risk.

This idea of limiting increment risk is based on the idea of a risk referent, in the past applied at project level (Charette, 1989). Risk^k , refers to this maximum and denotes the upper bound for the acceptable risk in Inc^k . This involves summing the risk scores for all requirements in a given increment and checking the constraint $\sum_{r(i) \in \text{Inc}(k)} \text{risk}(r_i, R^k) \leq \text{Risk}^k$ for each increment k.

Problem Statement for Software Release Planning

We can now formulate our problem as follows (Greer & Ruhe, 2004):

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

- (1) $\sum_{r(i) \in \text{Inc}(m)} \text{effort}(r_i, R^m) \leq \text{Size}^m$
for $m = k, k+1, \dots$ (Effort constraints)
- (2) $\sum_{r(i) \in \text{Inc}(m)} \text{risk}(r_i, R^m) \leq \text{Risk}^m$
for $m = k, k+1, \dots$ (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) $\text{card}(\{r_i \in R^k: i \in I(t)\}) \leq \text{resource}(t)$
for all releases k and all sets I(t) related to all resources t
(Resource constraints)
- (6) $A = \sum_{p=1, \dots, q} \lambda_p [\sum_{r(i) \in R(k)} \text{benefit}(r_i, S_p, \omega^*)] \Rightarrow L\text{-max!}$ with
 $\text{benefit}(r_i, S_p, R^k, \omega^*) = [\sigma\text{-prio}(r_i, S_p, R^k) + 1][\tau - \omega^*(r_i) + 1]$ 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. For any 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 judged. L-max means to compute a set of L best solutions ($L > 1$) so that the uncertainty of the inputs is taken into account and the analyst retains the decision-making power

Existing Requirements Prioritisation Approaches

The simplest approach to requirements prioritisation is where experts abstract across criteria to compare requirements and assign a ranking. This can be carried out practically by pair-wise comparison or traditional sorting methods such as a bubble sort, minimal spanning tree, and binary search tree. There are also several well-established methods that have been applied such as card-sorting and laddering (for example, Maiden & Rugg, 1996; Rugg & McGeorge, 1997), which typically involve grouping cards according to given criteria and using the groups.

Prioritisation using AHP

One now well-documented technique for prioritisation is the Analytic Hierarchy Process (AHP) (Saaty, 1980). This has been applied to the real-world problem of requirements prioritisation (Karlsson & Ryan, 1997; Karlsson, Wohlin & Regnell, 1988). To date AHP has not been specifically applied to incremental software delivery. However, if dependencies between requirements were taken into account, it could easily be applicable. With AHP, the relative importance of the assessment criteria is first determined through their pair-wise comparison. For example, if the criteria for assessing requirements are taken to be *cost-benefit ratio*, *impact on system quality*, and *risk-reduction*, it might be decided that *impact on system quality* is four times as important as *cost-benefit ratio* and that *risk-reduction* is one-third as important as *cost-benefit ratio* (Table 1). The rest of the rows could be inferred from the first row, but more often the user enters all the rows, allowing a consistency calculation to be performed.

Using these scores, a weighting for each criterion is derived. This can be achieved by calculating the eigenvalues for each row or by using the technique of averaging over normalized columns (Saaty, 1980). In this technique each cell is divided by the sum of its

Table 1: Recording Priorities in AHP

	Cost-Benefit	Impact on Quality	Risk Reduction
Cost-Benefit	1	4	1/3
Impact on Quality	0.25	1	1/9
Risk Reduction	3	1/4	1
Sum	4.25	5.25	1.44

Table 2: Determination of Priorities of Criteria using AHP

	Cost-Benefit	Impact on Quality	Risk Reduction	Norm. Sum
Cost-Benefit	0.24	0.76	0.23	0.41
Impact on Quality	0.06	0.19	0.08	0.11
Risk Reduction	0.71	0.05	0.69	0.48

Table 3: Prioritising candidate requirements using AHP

Cost-Benefit	r ₁	r ₂	r ₃	Norm. Sum
r ₁	1	3	2	0.63
r ₂	$\frac{1}{3}$	1	$\frac{1}{9}$	0.15
r ₃	$\frac{1}{2}$	$\frac{1}{4}$	1	0.21

column. The result of this for the example is shown in Table 2. The results for each row are summed and normalised by dividing by the number of criteria. In this case the relative values for the three criteria are 0.41, 0.11, and 0.48, respectively.

A similar approach is then used to assess each candidate requirement in relation to the chosen criteria.

The same scoring mechanism is used for all requirement pairs, so that each requirement obtains a preference score with respect to each decision criterion. The overall rating for a requirement is obtained by summing the preference scores and multiplying by the weighting for that criterion. This rating is then used for the prioritisation.

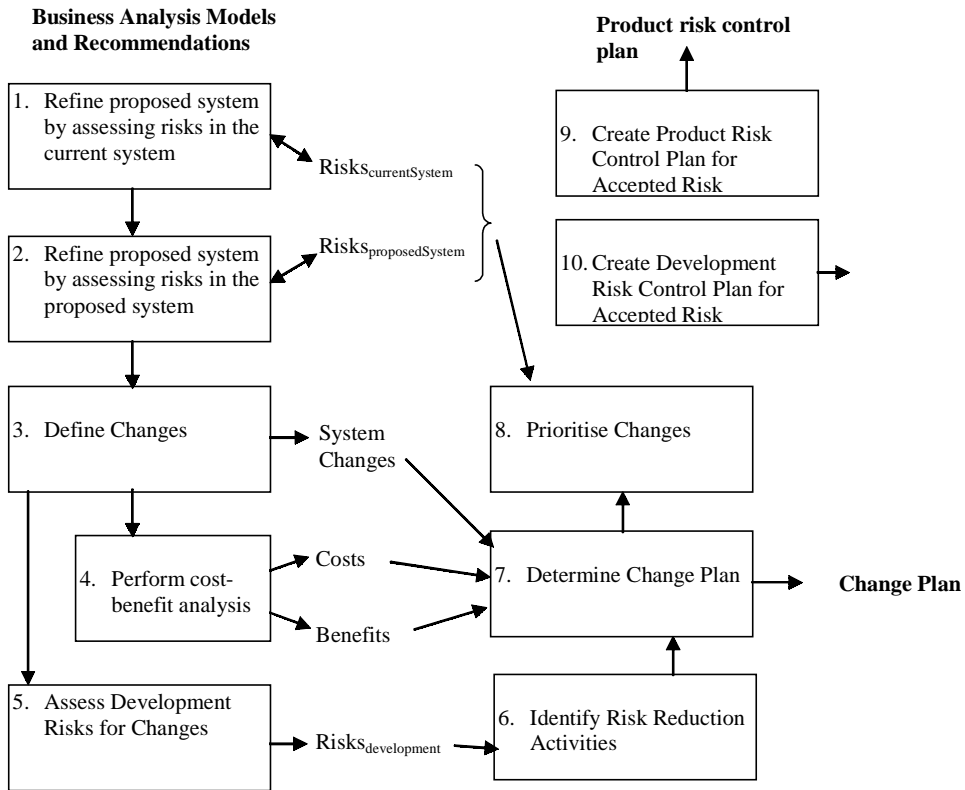
In an incremental model context, it is then a matter of assigning these requirements to releases using a greedy-type algorithm.

Prioritisation using Absolute Estimations

Changes can also be ordered by assessing them directly against criteria rather than relative to each other. This method can be classified as an *absolute assessment* approach (Greer, Bustard, & Sunazuka, 1999b). The use of such measurement has a number of distinct advantages over relative assessment. One is that fewer assessments are needed – just one for each requirement against each criterion. It also avoids the need to compare requirements with each other, which can be difficult, especially if they are unrelated.

One such approach is SERUM (Greer et al., 1999b), which uses estimations for cost, benefit, development risk, and operational risk reduction to inform the prioritisation process. The process as illustrated in Figure 2 starts with a business analysis from which the requirements have arisen. These requirements are refined by assessing risks in the current system (Stage 1) and ensuring that new requirements are created or amended to reduce those risks, where appropriate. Similarly since the requirements define the proposed system, the risks associated with the proposed system are assessed and where possible the requirements amended or new ones created (Stage 2). The requirements for the new system are defined in more detail in Stage 3, and cost-benefit analysis is carried

Figure 2: SERUM approach to prioritising requirements



out in Stage 4. This is a high-level assessment using a simple scoring system. In Stage 5 a risk assessment is made concerning the development of each requirement. Thus we have the following variables for each requirement: (i) the development risk; (ii) the risk reduction in moving from the current system to the proposed system, as calculated from Stages 1 and 2; (iii) the cost of the requirement and (iv) the benefit from the requirement. (Note that (iii) and (iv) could be combined as cost-benefit ratio.) Using these estimations, a prioritisation is determined in Stage 8. In practice Stage 8 is carried out by giving preference to the most important criteria, but alternative viewpoints can be made, ultimately the choice of approach being up to the user. From industrial studies (Greer et al., 1999a), the favoured criteria is often the benefit accrued from the requirement, although in mission, critical systems risk reduction may be more important. The risk plans in Stages 9 and 10 are useful by-products of the process.

SERUM does not at present formally handled dependencies between requirements, although this and a means to better support the final decision making process are subjects of current research.

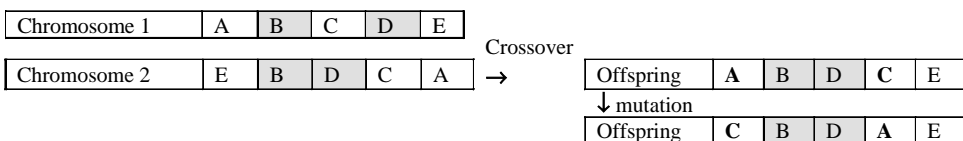
Using Heuristics

In the combinatorial optimisation field, there are a group of techniques that have been collectively termed “heuristics”. This grouping generally refers to techniques that try to find “near-optimal” solutions at reasonable computational cost (Reeves, 1995). Examples of techniques available for solving NP-hard problems (where deterministic means may not be feasible) such as the one outlined above for Release Planning include Simulated Annealing, Tabu Search, and Genetic Algorithms. We have used Genetic Algorithms for reasons outlined in the next section.

Genetic Algorithms

Genetic algorithms have been derived from an analogy with the natural process of biological evolution (Carnahan & Simha, 2001; Holland, 1975). With genetic algorithms an initial population is created and pairs of solutions, or *chromosomes*, are selected according to some fitness score. In this process of *selection* those members of the population with higher scores are given a higher probability of being chosen. The selected parents are then mixed using an operation known as *crossover*. Specifically, the method EVOLVE (Greer & Ruhe, 2004), which we will describe in the next section, makes use of the “order method” as described in Davis (1991): the crossover operator selects two parents, randomly selects items in one of them, and fixes their place in the second parent (for example, items B and D in Figure 3). 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. The resulting offspring is a mixture of its parents. A further operation, *mutation*, is applied. This is a random change to the chromosome and is intended to introduce new properties to the population and avoid merely reaching a local optima rather than the global optimum. 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, mutation is effected via random swapping of items in the new offspring. An example is shown in Figure 3 for the items A and C. The number of swaps is proportional to the mutation rate. The offspring is then added to the population so long as it represents a feasible solution. The population is normally maintained at a predetermined level, and so lower-ranked chromosomes are discarded. The net effect of this is a gradual movement toward an optimum solution. Termination of the algorithm can occur after a certain number of iterations, after a set time, or when improvement has ceased to be significant.

Figure 3: Illustration of crossover and mutation operators.



The extent of crossover and mutation is controlled by the variables *crossover rate* and *mutation rate*. The choice of “best” mutation and crossover rates is sensitive to the type of problem and its characteristics (Haupt & Haupt, 2000).

In applying this to requirements prioritisation, the chromosomes are made up of ordered requirements. Selection involves picking two of these from a randomly created population. Crossover is then the mixing of requirements in the chosen chromosomes and mutation is a swapping of two requirements in the offspring. A check is then made to see if the child meets all specified constraints and, if so, it is added to the population. If not a backtracking operation is carried out and the process retried. The population is maintained at a predetermined number by culling the weakest valued chromosome. A detailed algorithm is provided in the Appendix.

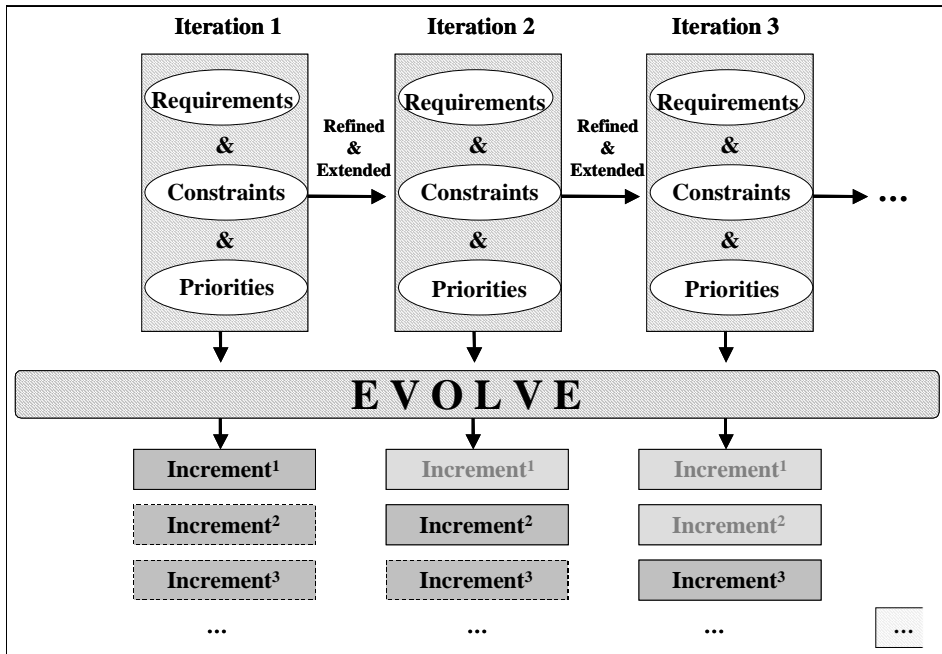
Solution Approach

In choosing an approach to solving the problems described in the Problem Discussion, the techniques described in the previous section could probably all be adapted for use. However there are two main issues to be resolved, if we are to incorporate the viewpoints of many stakeholders, to apply effort constraints, to manage risk levels, and to take account of dependencies. Firstly, any method needs to be usable and scalable. In the case of techniques that involve pair-wise comparison there is typically $O(n^2)$ comparisons between n alternatives. For example, using AHP with 20 requirements, a total of 190 ($n*(n - 1)/2$) pair-wise comparisons must be performed for each criterion. Secondly, given such a complex problem domain and such a large solution space, it is impossible to incorporate all of the constraints and find an optimum solution with any deterministic prioritisation technique. Given these factors, combinatorial optimisation techniques seem very suitable. The choice of Genetic Algorithms is based on the work of others, where, for example, Genetic Algorithms have been found appropriate to general cases such as the Travelling Salesman problem (Carnahan & Simha, 2001) and in software engineering to system test planning (Briand, Feng, & Labiche, 2002) and network route planning (Lin, Kwok, & Lau 2003).

Evolve Method

In EVOLVE software releases are planned as increments, but the planning process is repeated at each iteration. At each iteration the inputs include the current set of requirements, the constraints as described in (1)-(3) in the section on Problem Statement for Software Release Planning, and the stakeholder priorities. The objective function (6) in that section is applied for each solution. The purpose of the genetic algorithm is to determine the best (most optimal) release plan.

Figure 4. EVOLVE approach to assignment of requirements to increments



At each iteration k , the next planned increment, Inc^k , is finalised as indicated by the solid border in Figure 4, and all other undelivered increments, Inc^{k+1} , Inc^{k+2} , ... and so forth are tentatively planned, as indicated by the dashed border.

Algorithms and Tool Support

A greedy-like procedure was applied in order to assign requirements to the increments. Original precedence (2), coupling constraints (3), and resource constraints (4) 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.

Description with Sample Project

In evaluation of the method, a sample software project with 20 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 Y as shown below. This states that r_2 must come before r_3 , r_2 before r_7 , r_6 before r_{15} , and r_7 before r_{15} .

$$\Psi^1 = \{(r_{2,r_3}), (r_{2,r_7}), (r_{6,r_{15}}), (r_{7,r_{15}})\}$$

Further some requirements were specified to be implemented in the same increment as represented by the set x. This states that r_{10} and r_{11} must be in the same release, as must r_{17} and r_8 .

$$\xi^1 = \{(r_{10}, r_{11}), (r_{17}, r_8)\}$$

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 that has a capacity of 1 (resource(T_1)=1) that is used by requirements r_3 and r_8 . The sample project had resource constraints represented by the following index set.

$$I(T_1) = \{19, 20\}$$

Each requirement has an associated effort estimate in terms of a score between 1 and 10. The effort constraint was added that for each increment the effort should be less than

Table 4: Sample stakeholder-assigned priorities.

		Stakeholder				
		S ₁	S ₂	S ₃	S ₄	S ₅
Requirement	r ₁	1	2	3	4	5
	r ₂	5	4	3	2	1
	r ₃	1	1	1	1	1
	r ₄	3	3	2	2	3
	r ₅	4	4	4	4	5
	r ₆	1	2	3	2	1
	r ₇	2	1	1	2	1
	r ₈	4	5	5	4	1
	r ₉	3	3	3	2	4
	r ₁₀	2	2	2	2	2
	r ₁₁	5	5	5	4	3
	r ₁₂	3	4	3	4	3
	r ₁₃	2	1	1	2	3
	r ₁₄	1	1	2	2	1
	r ₁₅	3	5	3	5	4
	r ₁₆	4	4	4	5	5
	r ₁₇	5	3	3	3	3
	r ₁₈	1	1	2	1	2
	r ₁₉	5	5	5	5	5
	r ₂₀	3	3	3	3	2

25, that is, $\text{Size}^k = 25$ for all releases k . Five stakeholders were used to score the 20 requirements with priority scores from 1 to 5. These scores are shown in Table 4. As we can see, different stakeholders in some cases assign more or less the same priority to requirements (as for r_3). However the judgment is more conflicting in other cases (as for r_1).

The stakeholders, S_1 to S_5 , were weighted using AHP pair-wise comparison from a global project management perspective. Using the technique of averaging over normalized columns (Saaty, 1980), we obtained the vector (0.174, 0.174, 0.522, 0.043, 0.087) assigning priorities to the five stakeholders.

With regard to the genetic algorithm, we used the RiskOptimizer tool from Palisade (2001) with a default population size of 50 and an auto-mutation function that detects when a solution has stabilized and then adjusts the mutation rate. 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, from 0.4 to 0.8 in steps of 0.1.

Uncertainty in Effort Estimates

The model used in EVOLVE allows the use of probability distributions rather than discrete values for effort. This reflects the real-world difficulties in estimating effort. In the case study, a triangular probability distribution was created as shown in Table 5 and simulated using Latin Hypercube sampling.

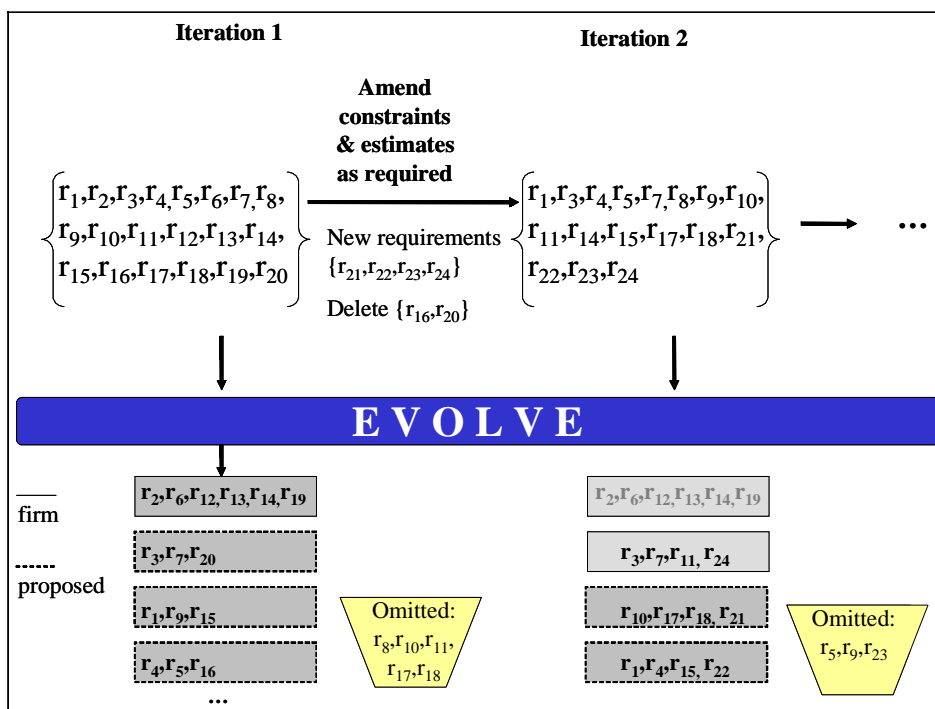
Table 5. Definition of triangular probability functions for effort to deliver requirements

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

Table 6. Sample results for release planning under different levels of probability for not exceeding the effort capacity bound $Effort = 25$ (Risk Referent=1.1)

Probability	Benefit (6)	Risk	Release	Assigned Requirements
99%	133	0.77	1	r ₂ r ₆ r ₈ r ₁₃
		0.64	2	r ₃ r ₇ r ₁₂ r ₁₅
		0.38	3	r ₁₀ r ₁₁ r ₁₄ r ₁₂
		0.82	4	r ₄ r ₁₇ r ₁₈ r ₁₉
95%	141	0.98	1	r ₂ r ₆ r ₁₂ r ₁₃ r ₁₄ r ₁₉
		0.44	2	r ₃ r ₇ r ₂₀
		0.41	3	r ₁ r ₉ r ₁₅
		0.72	4	r ₄ r ₅ r ₁₆
90%	144	0.97	1	r ₂ r ₇ r ₁₂ r ₁₃ r ₁₉
		0.36	2	r ₃ r ₁₄ r ₁₅ r ₁₇
		0.82	3	r ₄ r ₁₁ r ₁₈ r ₂₀
		0.68	4	r ₆ r ₉ r ₁₀ r ₁₆

Figure 5: Sample results from sample project release planning – iterations 1 and 2



This has the effect that the decision maker can plan the releases based on his or her confidence level in the estimates. In practice there is a trade-off between this level of confidence and the benefit achievable, as shown in Table 6. It also means that the adopted release plan has associated with it, a percentage indicating the probability that it will be adhere to its effort estimates.

At each iteration for a given effort confidence level, a solution is chosen from the L-best. The first increment is then firmed for implementation, and, at this point, any changes to the current set of requirements, their effort estimates, the stakeholder priorities, and the constraints can be made prior to planning the next and future releases. Figure 5 illustrates the process for our case study for the first two iterations. Suppose that the release plan for 95% confidence in table 5 is selected to be initiated. After the first iteration, the first increment ($r_2, r_6, r_{12}, r_{13}, r_{14}, r_{19}$) is firm and will be implemented. After this r_{21}, r_{22}, r_{23} , and r_{24} are added, with r_{16} and r_{20} being deleted. At this stage the stakeholders may change their priorities, and the effort estimates and the constraints may be revisited. This could be in response to new information gathered from the first iteration. A second increment is then firmed for delivery. In both iterations, future tentative increments are also given so as to provide details of the likely product evolution. In this example the number of planned increments has been limited to four. This means that there are some requirements that are omitted from the plan and do not contribute to the expected benefit. Figure 5: Sample results from sample project release planning – iterations 1 and 2


Conclusions

In this chapter we have reviewed some of the techniques available for prioritisation of requirements for incremental and iterative development of software systems. A category called *relative comparison* techniques can be formed. These are the methods that require the requirements to be compared to achieve a prioritisation. Once a prioritisation is achieved, a greedy algorithm taking into account the dependencies in the requirements could be applied to assign requirements to increments for release planning. This method should work well for small numbers of requirements but is impractical with larger sets of requirements. A further method as illustrated by the SERUM method could be classified as *absolute comparison* method. Here estimations are made for the factors considered important for prioritising requirements. Again, this could be applied to release planning for incremental and iterative development and has the advantage in producing useful metrics, such as risk assessments, that can be used elsewhere. One problem is in combining these estimations, and to date no attempt has been made to do this, the method rather depending on analyst judgement based on the information gathered.

Further, a recently developed technique has been described, EVOLVE, that uses Genetic Algorithms to optimise the prioritisation process for a number of stakeholders with differing priorities for requirements. The method takes into account differing stakeholder viewpoints, effort constraints, risk constraints, and dependencies between requirements. A case study has been used to illustrate the working of the method, and its efficacy is backed up by this and from experimentation and industrial feedback (Ruhe & Greer,

2003). Additional empirical work on this has confirmed the stability of the outputs and usability of the methods (Amandeep, Ruhe, & Stanford, 2004; Ruhe & Greer, 2003). EVOLVE also allows for uncertainty in the effort estimates, allowing decision makers to choose the level of confidence they require for effort estimates. One of the key strengths of EVOLVE is that the iterative planning process assumes changes will occur between phases. This better reflects the reality of software projects. Future work will involve developing the method further to take account of other possible constraints and situations.

References

- Amandeep  he, G., & Stanford, M. (2004). Intelligent support for software release planning. To appear in *Proceedings 5th International Conference on Product Focused Software Process Improvement*.
- Briand, L.C., Feng J., & Labiche Y. (2002). *Experimenting with genetic algorithm to devise optimal integration test orders*. (Tech. Rep.). Department of Systems and Computer Engineering, Software Quality Engineering Laboratory Carleton University.
- Bubenkbo, J.A. Jr. (1995). Challenges in requirements engineering. *Proceedings of 2nd IEEE symposium on Requirements Engineering*, 160-162.
- Carnahan J., & Simha, R. (2001). Natures algorithms. *IEEE Potentials*, 21-24.
- Charette, R. (1989). *Software engineering risk analysis and management*. New York: McGraw-Hill.
- Cockburn, A. (2002). *Agile software development*. New Jersey: Pearson Education,.
- Cusamano, M.A., & Yoffie, D.B. (1998). *Competing on Internet time: Lessons from Netscape and its battle with Microsoft*. New York: The Free Press.
- Davis, A.M. (2003). The art of requirements triage. *IEEE Computer*, 42-49.
- Davis, L. (1991). *Handbook of genetic algorithms*. New York: Van Nostrand Reinhold.
- De Gregorio, G. (1999). Enterprise-wide requirements and decision management. *Proceedings of the Ninth International Symposium of the International Council on System Engineering*, 775-782.
- Greer, D., Bustard, D., & Sunazuka, T. (1999a). Effecting and measuring risk reduction in software development. *NEC Journal of Research and Development*, 40(3), 378-38.
- Greer, D., Bustard, D., & Sunazuka, T. (1999b). Prioritisation of system changes using cost-benefit and risk assessments, *Proceedings of the Fourth IEEE International Symposium on Requirements Engineering*, 180-187.

- Greer, D., & Ruhe, G., (2004). Software release planning: An evolutionary and iterative approach. *Journal of Information Systems*, 46(4), 243-253.
- Hart S., Hogg G., & Banerjee M. (2002). An examination of primary stakeholders' opinions in CRM: Convergence and divergence? *Journal of Customer Behaviour*, 1(2), 241-267.
- Haupt, R.L., & Haupt, S.E. (2000). Optimum population size and mutation rate for a simple real genetic algorithm that optimizes array factors. *Applied Computational Electromagnetics Society Journal*, 15(2), 94-102.
- Holland, J.H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press.
- Jiang J.J., Klein G., & Discenza R. (2002). Perception differences of software success: Provider and user views of system metrics. *Journal of Systems and Software*, 63(1) 17-27.
- Karlsson, K., & Ryan, K. (1997). Prioritizing requirements using a cost-value approach. *IEEE Software*, 14(5), 67-74.
- Karlsson, J., Wohlin, C., & Regnell, B. (1988). An evaluation of methods for prioritizing software requirements. *Journal of Information and Software Technology*, 39, 939-947.
- Lin X.H., Kwok Y.K., & Lau V.K.N. (2003). A genetic algorithm based approach to route selection and capacity flow assignment. *Computer Communications*, 26(9), 961-974.
- Maiden, N.A.M., & Rugg, G. (1996). ACRE: Selecting methods for requirements acquisition. *Software Engineering Journal*, 11(3), 183-192.
- Palisade Corporation. (2001). Guide to RISKOptimizer: Simulation optimization for Microsoft Excel Windows version release 1.0. New York: Palisade Systems Inc.
- Rajlich, V., & Gosavi, P. (2002). A case study of unanticipated incremental change. *Proceedings of International Conference on Software Maintenance*, 442-451.
- Reeves, C. (1995). *Modern Heuristic Techniques for Combinatorial Problems*, New York: McGraw-Hill.
- Regnell, B., Host, M., Natt och Dag, J., Beremark P., & Hjelm, T. (2001). An industrial case study on distributed prioritisation in market-driven requirements engineering for packaged software. *Requirements Engineering*, 6, 51-62.
- Rugg, G., & McGeorge, P. (1997). The sorting techniques: A tutorial paper on card sorts, picture sorts and item sorts. *Expert Systems*, 14(2), 80-93.
- Ruhe, G., & Greer, D. (2003). Quantitative studies in software release planning under risk and resource constraints, *Proceedings of the IEEE-ACM International Symposium on Empirical Software Engineering*, 262-271.
- Saaty, T.L. (1980). *The analytic hierarchy process*. New York: McGraw Hill.

Appendix

This appendix presents a summary of the EVOLVE genetic algorithm.

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 = $\{(S_1, v_1), (S_2, v_2), \dots, (S_m, v_m)\}$

$S_{parent1}$ = 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 constraints

BackTrack($S_{offspring}$) = proprietary backtracking operation on a given solution. This backtracks toward the first parent until a valid solution is created.

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

CheckTermination() a Boolean function that checks if the user's terminating conditions have been met. This may be when a number of optimizations have been completed, when there is 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
      Sparent1 := Select(P);
      Sparent2 := Select(P / Sparent1);
      SOffspring := Crossover(Sparent1, Sparent2, cr);
      SOffspring := Mutation(SOffspring, mr);
      If NOT IsValid(SOffspring) THEN BackTrack(SOffspring);
      IF IsValid(SOffspring)
        BEGIN
          P := P ∪ {(SOffspring, Evaluate(Soffspring))};
          Cull(P);
        END;
      TerminateFlag = CheckTermination();
    END;
  END;
  RETURN(Max(P));
END.

```