Innovations Syst Softw Eng (2005) 1: 189–204
DOI 10.1007/s11334-005-0012-2

ORIGINAL PAPER

Omolade Saliu · Guenther Ruhe

Software release planning for evolving systems

Received: 20 April 2005 / Accepted: 1 June 2005 / Published online: 29 July 2005

Abstract Release planning is a crucial step in incremental software development. It addresses the issues involved with assigning features to sequence of releases of a system such that the most important technical, resource, risk and budget constraints are met. These problems are difficult to solve for even mid-sized systems. The issues become even more challenging in evolving systems where we need to consider the characteristics of the existing system, as the existing components of the system have their own history and status in terms of size, complexity, health, criticality, and understandability.

In this paper, we present the foundations for handling release planning for evolving systems in a rigorous manner. Based on a formalized problem description, we present a new solution approach for release planning of evolving systems called S-EVOLVE*. From analyzing and comparing different characteristics of the target components, where new features will be implemented, we obtain a more detailed perspective of the potential impact of implementing one feature or another. As part of this analysis, we have applied the analytic hierarchy process (AHP) to define weighting factors for component modifiability. The information gained is used for designing release plans based on thresholds for the relative extent of modifiability acceptable for a release. A set of structurally different release plans is generated based on solving a specialized integer linear-programming problem. The plans are proven to be semi-optimal for the stated objectives.

A case study is performed to demonstrate the added value of the approach. The evolving system under consideration is the intelligent decision-support tool ReleasePlanner. We compare and discuss results, for planning future releases, for the cases with and without consideration of system constraints.

O. Saliu (✉) · G. Ruhe
Laboratory for Software Engineering Decision Support,
University of Calgary,
2500 University Drive NW, Calgary, AB T2N 1N4,
Canada
E-mail: saliu@cpsc.ucalgary.ca; ruhe@cpsc.ucalgary.ca

Keywords Evolving systems · Release planning · Decision support · Analytic hierarchy process · Component modifiability · Case study

1 Introduction

As state-of-the-art software development moves further away from the rigid and monolithic waterfall model, the importance of release planning is brought to the forefront. Incremental software development offers sequential releases of software systems with additive functionalities in each increment. This approach allows customers to receive parts of a system early. Thus, each increment is a collection of features that form a complete system that would be of value to the customer. A major problem faced by companies developing or maintaining large and complex systems is deciding which features should be in which releases of the software [1].

Release planning for incremental software development assigns features to releases such that the most important technical, resource, risk, and budget constraints are met. Release planning, therefore, generalizes prioritization of features or requirements [2]. Release planning is a very complex problem involving different stakeholder perspectives, competing objectives and different types of constraints.

Since all features cannot possibly be delivered in a single release, some features will be delivered first while others will have to wait. In a perfect world, software companies would be able to deliver everything that customers want exactly when they want it. Unfortunately, this is not realistic. Budgets are capped, resources are limited, and schedules are constrained. With real-world constraints in mind, one needs to find a systematic way to make qualified decisions on which features need to, and realistically can, be delivered in which release. Finding the solution to this problem is the very essence of incremental software release planning.

Large software systems continually evolve to cope with changing customer requirements. Thus, software development typically proceeds as a series of changes to a base set of software. Most software products evolve as they are put into

use, because there is a need to extend functionality of the system by adding new features and also correcting errors that are discovered during operation of the software. Such evolution is achieved by modifying parts of the existing components of the software system to implement the new features and required changes.

Release planning in this context must take into consideration the operating environment and quality of existing components of the software system. This aspect of planning is extremely important since we cannot add new functionalities or change existing functionalities without proper knowledge of the impact of these changes. This aspect is completely ignored in current release planning solution approaches.

In this paper, we present the foundations to handle release planning for evolving systems more rigorously. Our initial work in this area [3] focused on using the defect history derived from operational usage of the system to characterize the health of software components. In this sequel, we present a more comprehensive approach to characterize the relative impact of integrating features into an existing system.

Based on a formalized problem description and following the paradigm of software engineering decision support, we apply the solution approach EVOLVE*. From analyzing and comparing different characteristics of the target components, where the features will be implemented, we receive a more detailed perspective of the potential impact of implementing one or the other feature. As part of this analysis, we have applied the analytic hierarchy process (AHP) [4] to define weighting factors for component modifiability. The information gained together with that of extent of modification to be done on the component is used for designing release plans based on thresholds for the relative amount of impact acceptable for a release.

The rest of the paper is organized as follows. In Sect. 2 we present the underlying assumptions and the subsequent formalization of traditional release planning. We discuss the challenges raised by consideration of evolving systems. To address these challenges, we propose a component modifiability evaluation approach based on AHP in Sect. 3. We extend the former solution approach EVOLVE* by these results to understand the potential impact of feature implementation in different parts of the system better. The resulting approach, called S-EVOLVE* (system EVOLVE star), is described in Sect. 4.

A case study was performed to demonstrate the concepts and the added value of the new approach. The system under consideration is the intelligent decision support tool ReleasePlanner. The results can be found in Sect. 5. Finally, Sect. 6 gives an outlook to future research.

2 Release planning in a nutshell

2.1 Analysis of existing techniques

Release planning can be approached from different perspectives. We have identified two fundamental approaches which we will call: (1) the art and (2) the science-based planning of releases. The art of release planning involves mainly relying on human intuition, communication, and capabilities to negotiate between conflicting objectives and constraints. The science of release planning involves emphasizing formalization of the problem and applying computational algorithms to generate best solutions. Both art and science are important to achieve meaningful release planning results. Our focus here is on the science-based approach.

A number of release planning methods have been developed. In [3], we have performed a comparative analysis of seven methods.

- Estimation-based management framework for enhancive maintenance [5].
- Incremental funding method [6].
- Cost–value approach for prioritizing requirements [7].
- Optimizing value and cost in requirements analysis [8].
- The next release problem [1].
- Planning software evolution with risk management [9].
- Hybrid intelligence (EVOLVE*) [10].

From performing a comparative analysis, the following deficits in the existing release planning methods were observed:

1. There is no major focus on addressing system constraints. The attempt in [9] assumes operational risk of system failure can be given probabilistic values, without deriving such information from the architecture, code base, and other historical data of the system.
2. Except for the technique in [10], there is a generally lack of stakeholder involvement in release planning. However, this is of paramount importance to ensure that the right product is developed.
3. Failure to include resource consumption as an integral consideration when deriving plans. Many methods just consider effort while many don't consider resource consumption at all. Consequently, proposed plans are likely to fail during implementation.
4. The scope of planning is often limited to just the next release. However, it is also useful to give an answer to the question: when is a certain feature supposed to be released (if not in the next release)?
5. Lack of decision support tools that are fully developed, based on sound methodology and able to generate and evaluate meaningful decision alternatives.
6. Release planning has been largely focused on fixed-release intervals and no current work exists on release planning with flexible time intervals.

In this paper, we are addressing deficits 1–5 with an emphasis on the first deficit. The fact that large software systems can involve the implementation of several hundred features constrains the capabilities of pure human judgment in making decisions about which features should come next and in which sequence. Human limitations on coping with large number of features, conflicting stakeholder interests, and the general complexity of release planning have led to efforts focused on the formal modeling of the problem [1, 8, 10]. In this sequel, we give an overview of the main formulation of the

optimization-based solution approach of EVOLVE*. This approach will later be extended to accommodate system specific characteristics resulting in a method called S-EVOLVE*.

2.2 Features and related decision variables

The concept of a feature is applicable and important for any software development paradigm. However, it is especially important for any type of incremental product development. Features are the selling units provided to the customer. Definitions of features are abundant in the literature. In the context of this research, we follow the definition given by [11] which defines a features to be "a logical unit of behavior that is specified by a set of functional and quality requirements".

We assume a set of features F = $\{f(1), f(2), \ldots, f(n)\}$. The goal is to assign the features to a finite number K of release options, with the option of postponing some features. A release plan is characterized by a vector of decision variables $x = (x(1), x(2), \ldots, x(n))$ with $x(i) = k$ if feature $f(i)$ is assigned to release option $k \in \{1,2, \ldots ,\mathrm{K}\}$, and $x(i) = K + 1$ if the feature $f(i)$ is postponed.

2.3 Stakeholders

Stakeholders are extremely important for performing realistic and customer-oriented release planning. We assume a set of stakeholders $S = \{S(1), \ldots, S(q)\}$. Each stakeholder $S(p)$ can be assigned a relative importance $\lambda(p) \in \{1, \ldots , 9\}$. The underlying meaning is

- $\lambda(p)$ = 1 means extremely low importance,
- $\lambda(p)$ = 3 means low importance,
- $\lambda(p)$ = 5 means average importance,
- $\lambda(p)$ = 7 means high importance,
- $\lambda(p)$ = 9 means extremely high importance.

The interpretation of the even values not shown is that they are refinements of the values above and below them. We have chosen a nine-point ordinal scale for expressing the extent of importance as this gives sufficient detail to differentiate between the importance of stakeholders. However, the whole approach is applicable in the same way for other scales.

2.4 Prioritization of features by stakeholders

To select and schedule features, there must be an agreed upon statement of priorities for features. In our proposed model, prioritization by each stakeholder $S(p)$ is done with respect to three different types of criteria, each defined on a nine-point ordinal scale:

- Value (denoted value$(i,p) \in \{1, \ldots, 9\}$) assigned by stakeholder p to feature $f(i)$. The value-based priority measure is used to express the expected value that the implementation of this feature will add to the stakeholder. An increasing order of value corresponds to an increasing value-based priority. The ordinal nine-point scale we have used for the value is by no means the only type of measurement that can be used, since the value propositions may also be expressed in financial terms, or other measures of value.
- Satisfaction (denoted sat$(i,p) \in \{1, \ldots, 9\}$) assigned by stakeholder p to feature $f(i)$. This priority measure is used to express the extent of satisfaction with the situation that f(i) is assigned to the next release. A measure of satisfaction is different from that of value because it expresses the urgency with which this stakeholder desires the feature. An increasing order of satisfaction corresponds to an increasing satisfaction-based priority.
- Dissatisfaction (denoted dissat$(i,p) \in \{1, \ldots, 9\}$) assigned by stakeholder p to feature $f(i)$. This priority measure is used to express the extent of dissatisfaction with the situation that $f(i)$ is not assigned to the next release. An increasing order of dissatisfaction corresponds to an increasing dissatisfaction-based priority.

2.5 Technological constraints

A study of requirements repositories in the telecommunications domain by Carlshmare [12] concluded that only about 20% of the requirements were singular or independent of each other. The authors have identified different types of dependencies between features. We consider two types of technological constraints where features can either be in a coupling relation C or in a precedence relation P. Both relations are subsets of the product set F × F.

Two features $f(i)$ and $f(j)$ are coupled (written as $(i,j) \in$ C) if they are required to be implemented in the same release. This dependency can be due to implementation or usage issues. Simultaneously, feature $f(i)$ is in a precedence relation to feature $f(j)$ (written as $(i, j) \in$ P) if feature $f(j)$ is not allowed to be implemented in a release earlier than $f(i)$. In terms of the introduced decision variables, this means that

$$x(i) = x(j) \text{ for all } (i, j) \in \mathrm{C} \subset \mathrm{F} \times \mathrm{F} \text{ (Coupling)} \quad (1)$$

$$x(i) \leq x(j) \text{ for all } (i, j) \in \mathrm{P} \subset \mathrm{F} \times \mathrm{F} \text{ (Precedence)} \quad (2)$$

2.6 Resource consumptions

Different resources are required for the implementation of features. In addition, there are capacity bounds on the amount of resources available in each release cycle. In the general model, we have considered R types of resources tentatively involved in the implementation of features. Correspondingly, we define resource capacities Cap(r,k) for each resource type $r = 1, \ldots,R$ and all releases $k = 1, \ldots,K$. To become a feasible plan, decision variables must satisfy

$$\sum_{x(i)=k} \text{resource } (i, r) \leq \text{ Cap } (r, k) \quad (3)$$

for all releases $k = 1, \ldots,K$ and all resource types $r = 1, \ldots,R$.

2.7 System constraints

To model the potential impact of implementing feature $f(i)$ into the existing system S, we assume that S can be decomposed into M disjoint subsystems (which will be called components) C(1), C(2), ... , C(M), e.g.,

$$S = \cup_{m=1,\ldots,M} \mathrm{C}(m) \tag{4}$$

To implement a new feature, one or more of these components must be modified to integrate the feature. Even when we identify only one component as the impacted component, it should be noted that such components might interact with other components that would need to be modified as well. Thus, we assume a mapping Ψ from the set of features to the power set of all components. By that, each feature $f(i)$ is assigned a set of impacted components $\Psi(i) \subset S$.

For each feature $f(i)$, we define two factors describing the potential impact of integrating this feature into the system S. The first one is called the difficulty of modification DoM(C(m)) and refers to the inherent difficulty in modifying an affected component C(m) of system S to extend functionality of the system by implementing new features. It is an attribute of the component independent of feature $f(i)$. However, only components C(m) $\in \Psi(i)$ are considered for that DoM(C(m)). The second factor is called the extent of modification XoM(C(m), $f(i)$) and refers to the extent of impact the implementation of $f(i)$ has on the component C(m). This is mostly related to the extent of modification that would be done on the component.

XoM(C(m), $f(i)$), is a local measure that is specific to each feature and the corresponding components hosting it. For XoM to be useful for release planning the measure must be determined by evaluating target components C(m) $\in \Psi(i)$ and projecting estimates of the extent of modification that would be performed on them. DoM(C(m)), on the other hand, is a global measure that is a function of the components only.

To determine XoM(C(m),$f(i)$), we define it to be the extent to which an existing component is to be changed by a proposed feature and measure it as a percentage of code modification relative to the original size of the component.

The two concepts are illustrated in Fig. 1. The system is assumed to consist of six components, e.g., S = {C(1), ..., C(6)}. Four components are affected by the assumed implementation of feature $f(i)$, e.g., $\Psi(i)$ = {C(1), C(2), C(4), C(6)}. The different grey levels of the components C(m) refer to the different levels in the difficulty of modification. The hatched areas within affected components describe the extent of modification XoM(C(m),$f(i)$). The larger the hatched areas, the larger the extent of modification will be.

We will use the functions DoM(S,$f(i)$) and XoM(S,$f(i)$) to describe the difficulty of modification and the extent of modification of feature $f(i)$ with respect to the whole system S. In Sect. 3, we will discuss an approach to determine components modifiability. These results are used in Sect. 4 to determine these values quantitatively.

The total impact of implementing feature $f(i)$ into system S is denoted by Impact($f(i)$). It is determined as the product of the difficulty of modification and extent of modification of the components affected by the feature $f(i)$. This is a relative measure where the detailed formula will be explained and justified in Sect. 4. We assume that, for each release k, the total impact of implementing new features (which includes updating existing ones) is restricted by a user-defined threshold $\beta(k)$. The actual value of the threshold is problem-specific and can be varied to study the sensitivity of proposed solutions with respect to this value.

$$\sum_{x(i)=k} \text{Impact}\ (S,\ f(i)) \leq \beta(k) \tag{5}$$

for all releases $k = 1, \ldots, K$.

2.8 Objective function

A release planning technique must have an approximate definition of objectives. Typically, it is a mixture of different aspects such as value, urgency, risk, satisfaction/dissatisfaction, return on investment, etc. The actual form of the explicit function tries to bring the different aspects together in a balanced way. In our model, we assume a multiplicative relationship between stakeholder satisfaction and dissatisfaction with stakeholder value perception. According to this assumption, the objective is the maximization of a function F(x) among all release plans x satisfying the above technological and resource constraints. F(x) is defined as

$$\mathrm{F}(x) = \Sigma_{k=1\ldots K} \xi(k) [\Sigma_{x(j)=k}\ \mathrm{WAP}\ (j)] \tag{6}$$

where

$$\mathrm{WAP}(j) := \sum_{p=1,,,q} \lambda(p) \cdot [\mathrm{sat}(j,\ p) + \mathrm{dissat}(j,\ p)] \cdot \mathrm{value}(j,\ p) \tag{7}$$

for all features $f(j)$ and releases $k = 1 \ldots K$.

For each release option k, parameter $\xi(k)$ describes the relative importance of the release option and its relative impact on the objective function.

3 An approach to characterize component difficulty of modification

3.1 Related results

In our attempt to develop a modifiability measure for software components, we take a brief overview of related work in characterizing evolving systems. Several studies using relevant release histories to predict or classify software systems according to their fault-proneness include those discussed in [13–19].

Software modules with histories of large numbers of defects are said to be highly likely to always be faulty [19]. As the code base of a system evolves through releases it begins to decay, which implies that current and/or future changes to the code are difficult to make such that any change to a system causes more faults to be introduced into the system [14,20]. The difficulty in changing decayed code

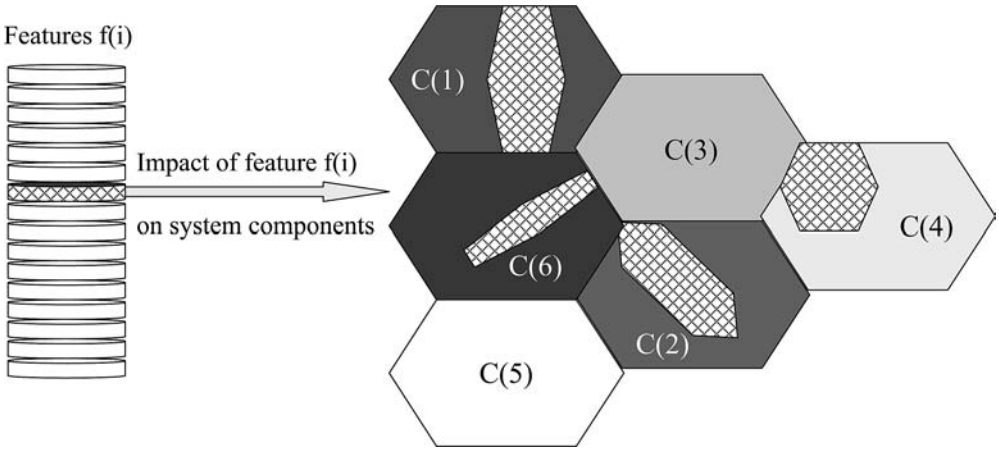


Fig. 1 Impact of the implementation of feature $f(i)$ on the system components C(m)

which constitutes part of our modifiability definition have been noted as affecting three aspects of product evolution [13, 21]: increased cost of implementing a change, increased time to complete a change, and reduced quality of the changed product. Ohlsson et al. [22] observed that identification of such parts of a system that need improvement (i.e., evolution-sensitive parts [23]) makes planning and managing for the next release easier and more predictable. Since we want to make feature selection and scheduling decisions with knowledge about the entire set of components of the system, a model of modifiability of all the components, and not only the evolution-sensitive parts, is desirable.

The software systems classification approaches mentioned above derived one form of metric or another from a collected set of data about the system under consideration to perform the classification. Regression analysis has specifically been formed to be extremel useful in such situations. Unfortunately, data collection and metrics definition are caught with difficulties: (1) Software engineering data are typically difficult to collect, (2) Even when the date are available, it is always difficult to guarantee their quality, (3) Metric defined on datasets from a specific from a specific software product cannot always be easily generalized to other products, and (4) Considering the many metrics available, and that many are widely criticized, how do we even select the appropriate ones?

Based on the difficulties we have outlined, we would not adopt any adhoc collection of metrics from repositories of software project data in this work. Our metrics to assess DoM and XoM will be based on the judgement of experienced experts that are familiar with system. However, previous empirical work in this area will guide our selection of attributes that describe the characteristics of system components.

3.2 Overview of the characterization model

To perform the quality modeling to characterize the difficulty of component modification, we contend that there is no universal measurement that works across all organizations and projects types. Therefore, local aspects of the measurement program should be considered depending on what type of data the organization keeps and the maturity of the organization. Based on our survey of existing works that are targeted at deriving quality measures of a system based on product release histories [13–18], we will present a high-level modifiability framework that could guide data collection and metrics derivation for assessing modifiability.

In the assessment framework shown in Fig. 2, we have identified five major factors that contribute to the modifiability of a component. Most of these factors can be assessed based on further refinement into lower-level criteria that are directly measurable and for which data can be easily collected. The metrics for the lower-level factors would need to be aggregated to provide the assessment values for the upper-level factors and eventually the topmost modifiability goal. The definition of the metrics can be performed based on the level of historical data available in the organization and experiences of the developers and designers of the system under consideration.

The factors described in Fig. 2 are by no means exhaustive since several local factors could also contribute to modifiability. We further explain the meaning of each of these factors below.

3.3 The five key attributes

3.3.1 Size

Most components of a software product exhibit varying sizes depending on the functionalities implemented in them and other product factors. Some are very compact and isolated while others span tens of thousands of lines of code. When designing and implementing larger components, it is harder to keep all details and interactions perfectly straight in one's mind. Thus, with all other factors being equal, the likelihood of design/implementation deficiencies in larger components is greater than it would be for smaller components. Following a similar argument, when failures are detected, finding a defect in a smaller component is normally less time-consuming and less complicated than in a component of larger size.

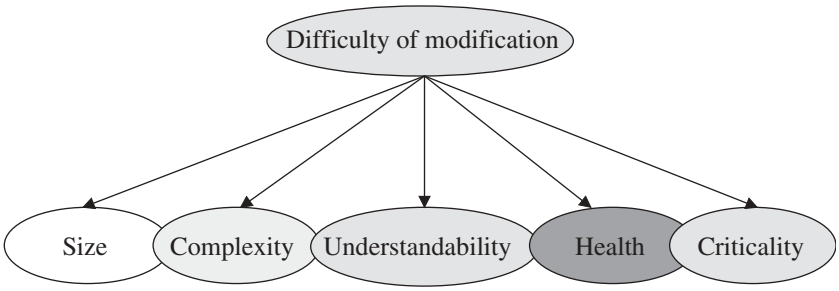


Fig. 2 The hierarchical goal-based modifiability assessment framework

3.3.2 Complexity

Lehman's laws of software evolution [24] states that software complexity tends to increase while the quality of the product will tend to decrease. Some of the popular code complexity measures existing include the McCabe cyclomatic complexity [25] and Halstead's program volume [26]. A comparison of several of these complexity measures have been discussed by Schneidewind and Hoffman in [27] where they also found that, irrespective of the complexity measures used, programs with high complexity also have high number of faults and vice versa. As a matter of fact, the complexity of a code base is determined by the structure of the code, which is also largely affected by the coupling (e.g., number of other interfacing components or communicating components) of different components of the system. As the coupling value of a component increases, so does the number of other components that need to be modified whenever a change affects one of them. Empirical studies by Graves et al. [13] and Eick et al. [14] have equally identified interconnectedness of components as part of the factors that could result in the likelihood of code decay, which in turn makes a component difficult to modify.

3.3.3 Understandability

This refers to the case with which the component can be understood by the developers who are modifying it. Empirical studies have revealed that this criteria is a function of the expertise of whoever is making the changes [14], the growth rate of the component from one release to the other [24], and how long the component has been part of the system [14]. A rapidly growing component at every release may be more difficult to understand than a component that has minimally changed across several releases. This is even truer when different developers have worked on the rapidly growing components at different release points.

Subject matter experts (SMEs) for some components are far more familiar with their domain than others, with some SMEs being the original designers/implementers of those components who are thus intimately familiar with the reasoning behind the choices and decisions made in designing the component. In other cases, original designers have moved on and component ownership has bounced around from developer to developer, meaning that understanding of some components and technical details of their current design and implementation are lacking.

3.3.4 Health

Health of a component refers to the extent it can be trusted to exhibit consistency in performance without failures. This is a function of the failures or defects that have been reported against that part of the system in the past during the course of usage, severity of the faults causing the failures, how recent a failure is, and the number of changes the component has gone through that are not related to failure. All of this information can be derived from the release history of the software product, which is kept in change-management systems and version-control systems. If components are unhealthy, they will be fault-prone, and changing any small part of the code can be very risky and time-consuming.

Empirical studies [13, 14] reveal that correlations exist between the health (i.e., ability to use or reuse existing code base [28]) of program code, the quality of the resulting product, and the functionality that can be added to the system. As a code base evolves, it becomes more difficult to add new features to it [13, 20]. In his dissertation, Dayani-Ford [28] observed that assessing the health of existing products has received little attention in the literature.

3.3.5 Criticality

In any system, some parts are mission critical while others are not, and normal operation could still be carried out without the noncritical parts. Some components add only minor functionality, or functionality that is rarely, if ever, used. Clearly, deficiencies in these kinds of components, although unpleasant, are not service impacting. Since customers can live without such components, deficiencies in these components can be addressed when time permits. The criticality of a component must be taken into consideration when evaluating the modifiability because extra effort might be needed to ensure that mission-critical components achieve their goals. This also means that, if a bug was inadvertently introduced into one of these low-priority components while implementing a new feature, the consequences of this bug are not likely to be catastrophic. On the other hand, if a bug manifests itself as

a failure in a more critical system component, consequences on the system will likely be far greater and more profound.

3.4 Quantitative evaluation of the difficulty of modification

In our previous work [3], we have limited the measure of a component's modifiability to the health of a software component, given that the data required on health are easier to collect. Developing generalized modifiability metrics to measure each of the factors considered in Fig. 2 depends on how much historical data have been collected over time. Although this data will be extremely useful, they are not always available. As an alternative to the situation when we do not have historical information, we present a method that we describe later in this section to elicit expert opinions on a set of components, based on the modifiability factors described. Expert opinion has been adjudged to be acceptable in sciences where no accepted measurements are available, as long as the expert's opinions are elicited in a formalized way. To this end, we will use AHP as an acceptable method to elicit expert opinion in a formalized way. AHP consists of a set of steps where elements are evaluated pairwise according to a certain scale to determine which of the two elements being compared is more important and how much more important it is.

Gathering expert opinion about how these factors determine the modifiability of chosen components must be done from the perspective of participating knowledgeable experts (e.g., developers, designers, etc.) that are familiar with the architecture of the system under consideration. This in itself calls for the ranking of the experts' importance based on system familiarity.

We will describe the performance of the process to determine the relative extent of the difficulty of a component's modification in six steps.

3.4.1 Step 1: Definition of criteria and alternatives

The first step is structuring of the problem in a hierarchical form, as shown in Fig. 3. At the top level is the overall goal of difficulty of modifying components. In the second level are the five factors that contribute to the goal. The third level represents the alternatives (components) that are to be evaluated in terms of the factors in the second level.

3.4.2 Step 2: Assignment of priorities to experts

The second step is the assignment of importance values to the experts that would participate in the assessment of all the components for modifiability. During aggregation of the final result, the importance of the experts would be a multiplicative factor of the vectors of relative component modifiability developed by each participating expert.

3.4.3 Step 3: Prioritization of modifiability factors

The third step is to perform pairwise comparison between the factors impacting difficulty of modification. This pairwise comparison is carried out using AHP, which requires the construction of a 5×5 matrix made up of the five modifiability factors in the rows and columns. According to Saaty [4], the fundamental nine-point scale used for the purpose of this pairwise comparison is shown in Table 1. For each pair of factors (starting with size and complexity, for example) their relative importance value as described in the intensity of importance column of Table 1 is inserted in the position where the size row meets the complexity column. In the position where the complexity row of and the size column meet, the reciprocal value is inserted, while all the positions in the main diagonal of the matrix carries a value of 1 to show equal importance when comparing a factor to same factor. The question to be asked at this stage when comparing two factors to get the intensity of importance value is: of the two factors being compared, which of them is more important in determining the difficulty of modifying components?

After completing the pairwise comparison and appropriate values entered in the matrix, the eigenvalues computation of the matrix is done to establish a priority vector. The components of this vector represent the relative importance of the respective factor. This process is performed independently by each expert.

3.4.4 Step 4: Prioritization of components for a fixed factor

The fourth step is to carry out a relative pairwise comparison of the components with respect to each of the modifiability factor. The questions to ask when comparing two components based on the factors are of the following forms: How much more difficult is it to modify component A than component B with respect to size?" The same question applies to complexity, understandability, health, and criticality.

3.4.5 Step 5: Overall priorities of components

The fifth step is to establish the overall priorities of the components from each expert perspective. We lay out the priorities of the components with respect to each factor (i.e., δ_{mn} which are the priority vectors from step four) in a matrix and multiply each column of the matrix by the priority of the corresponding factor in the column (i.e., ξ_i). Then, we add the result of the multiplication across each row which results in the desired overall priority vector of the components from a single expert perspective. The structure of the table is illustrated in Table 2 (for Expert A).

3.4.6 Step 6: Aggregation of expert judgments

The sixth step is to establish the final aggregated overall relative priorities of the components based on their difficulty of modification from the context of all the experts. That is, the computed component prioritization vectors from each experts are linearly combined according to the importance of the experts to establish the final aggregated priority vector. The higher the priority value assigned to a component, the more difficult it is to modify the component.

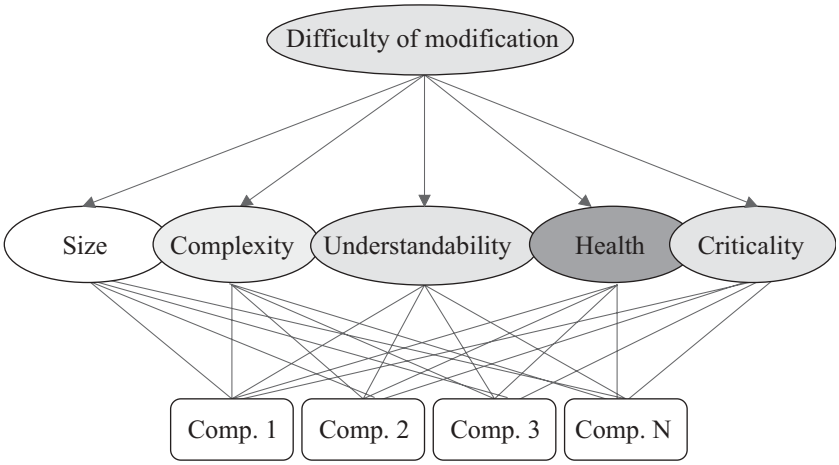


Fig. 3 Decomposition of the problem into a hierarchy

Table 1 Scale for pairwise comparisons using AHP

Intensity of importance	Definition	Explanation
1	Equal importance	The two activities or variables (i and j) are of equal importance
3	Moderate importance of one over another	Experience slightly favor one activity over another
5	Strong importance	Experience strongly favors one activity over another
7	Very strong importance	An activity is strongly favored and its dominance demonstrated in practice
9	Extreme importance	The evidence favoring one over another is of highest possible order of affirmation
2, 4, 6, 8	Intermediate values between the two adjacent judgments	When compromise is needed
Reciprocals	If activity i has one of the above numbers assigned to it when compared with activity j, then j has the reciprocal value when compared with i	

Table 2 Overall priorities of the components from the perspective of Expert A

	Complexity (ξ_1)	Understandability (ξ_2)	Health (ξ_3)	Criticality (ξ_4)	Functionality (ξ_5)	Priority vector (Expert-A)
Component 1	δ_{11}	δ_{21}	δ_{31}	δ_{41}	δ_{51}	$\sum \xi_{i*}\delta_{i1} = \lambda_{11}$
Component 2	δ_{12}	δ_{22}	δ_{32}	δ_{42}	δ_{52}	$\sum \xi_{i*}\delta_{i2} = \lambda_{12}$
Component 3	δ_{13}	δ_{23}	δ_{33}	δ_{43}	δ_{53}	$\sum \xi_{i*}\delta_{i3} = \lambda_{13}$
Component N	δ_{1N}	δ_{2N}	δ_{3N}	δ_{4N}	δ_{5N}	$\sum \xi_{i*}\delta_{iN} = \lambda_{1N}$

4 S-EVOLVE*: A method for planning releases of evolving systems

4.1 Overview

Our proposed method S-EVOLVE* is an extension of the EVOLVE* approach presented in [4]. It adds on EVOLVE* in that it takes into account the characteristics of the target system for feature implementation. In what follows, we present the process model and describe the individual steps of this model in more detail.

Figure 4 shows a generic process model describing main release planning activities and their related inputs and outputs. Therein, three roles that contribute to the process and products of release planning are identified—project managers, stakeholders, and the support environment. Activities occur directly under the roles that are actively involved. For example, project managers and stakeholders' roles are involved in feature elicitation, while a support environment maintains the group of features elicited. Major activities of the process model are described by rounded rectangles and the intermediate results of each activity are shown in ovals. The key generic functions of the model are described in Subsect. 4.2. These functions work seamlessly together to provide release plan alternatives for the decision maker. Their workflow is shown in Fig. 4.

The major extension constituting S-EVOLVE* are the activities that appear in the shaded region of Fig. 4. The overall process model is an extension of [29].

4.2 Main steps

4.2.1 Feature elicitation

A variety of techniques are known to determine what the users and customers really want. Features should be described so they are understandable, consistent, complete, and correct. Collecting meaningful features is a complex problem in itself, and it is not studied in this paper. For an overview we refer the reader to [29].

4.2.2 Problem specification

Release planning should have an approximate definition of objectives. The focus could be related to maximizing the value of releases or maximizing the weighted sum of stakeholder satisfaction. This step formalizes the objective function that models the underlying problem and identifies all the constraints present in the features. These constraints may be related to interdependencies among the features, or on dependencies of features on the existing system environment when planning an evolving system. The result of this task is the objective definition and the identified constraints.

4.2.3 Resource estimation

There is always a limit to the resources available for software projects. Resource estimation activity in release planning is aimed at determining the likely amount of effort, cost, and schedule required to implement the features. This task is crucial to the success of the release planning process, since it matches the resource requirements of the problem with the available project resources. The products from this task are estimates of effort, cost, and schedules required to implement features. Some of the stakeholders may be interested in the result from the resource estimation activity in order to assign their votes reflecting their preferences. Resource estimation is also one of the bases upon which final prioritization and selection of features will be done during the release-plan generation activity.

4.2.4 Stakeholder voting

Stakeholders are the people who directly or indirectly influence, or are influenced by the software project plan. They may include developers, managers, customers, and users. Giving stakeholders the opportunity to vote on features according to their preferences is important because they decide the evaluation criteria for a product's or system's success. Prioritization of stakeholders is necessary to differentiate their relative importance levels. The result of this activity is a set of priorities assigned to features according stakeholders preferences.

4.2.5 Component modifiability assessment

As this activity goes somewhat beyond the traditional release planning performed independent of system characteristics, the main issue here is the characterization and assessment of the difficulty of modifying the components (i.e., DoM(C(m))) of the system. The method to achieve this using AHP is described in Sect. 3 above.

4.2.6 Feature-driven impact analysis

Feature-driven impact analysis (FDIA) refers to the process of identifying the component(s) of the existing system that would be impacted by the implementation of each feature. The judgments would be made by developers and designers who are familiar with the existing architecture of the system. The whole idea is to be able to quantitatively or qualitatively account for the impact of integrating each feature into existing components.

In most cases, implementation of a feature would require modification of several components. The values of DoM(C(m)) for each component must be aggregated for all such identified components that are impacted by the feature.

4.2.7 Impact quantification

It should be noted that the results for DoM derived using AHP are normalized relative measures (i.e., $0 \leq \text{DoM}(\text{C}(m)) \leq 1$), but XoM is not necessarily normalized, as it is a percentage. In the case of DoM we must look for a way to aggregate the fractions in a consistent way such that multiplication of several values less than one does not converge to zero too quickly.

We introduce the total difficulty of modification and the total extent of modification as the product of the respective component properties as follows:

$$\text{TXoM}(f(i)) = \prod\nolimits_{m=1..M} (1 + \text{XoM}(\text{C}(m),\, f(i))) \tag{8}$$

$$\text{TDoM}(f(i)) = 1 - \prod\nolimits_{m=1..M} (1 - \text{DoM}(\text{C}(m))) * \delta_{im} \tag{9}$$

where,

$$\delta_{\text{im}} = \begin{cases} 1 & \text{if C}(m) \in \psi(i) \\ 0 & \text{otherwise} \end{cases}$$

Thus, for any such feature considered in (8) and (9), the impact can be written as follows:

$$\text{Impact}(f(i)) = \text{TDoM}(f(i)) \times \text{TXoM}(f(i)) \tag{10}$$

4.2.8 Release plan generation

Information derived from the products of all the preceding tasks and other constraints are used in generating sets of release plan alternatives. After the assignment of features to releases, a procedure is necessary to determine whether any of the alternative solution plans generated meet the expectations of the decision-maker.

The S-EVOLVE* solution algorithms provide a set of nearly optimal solutions. Its computation is based on integer linear programming (IP) combined with heuristics that

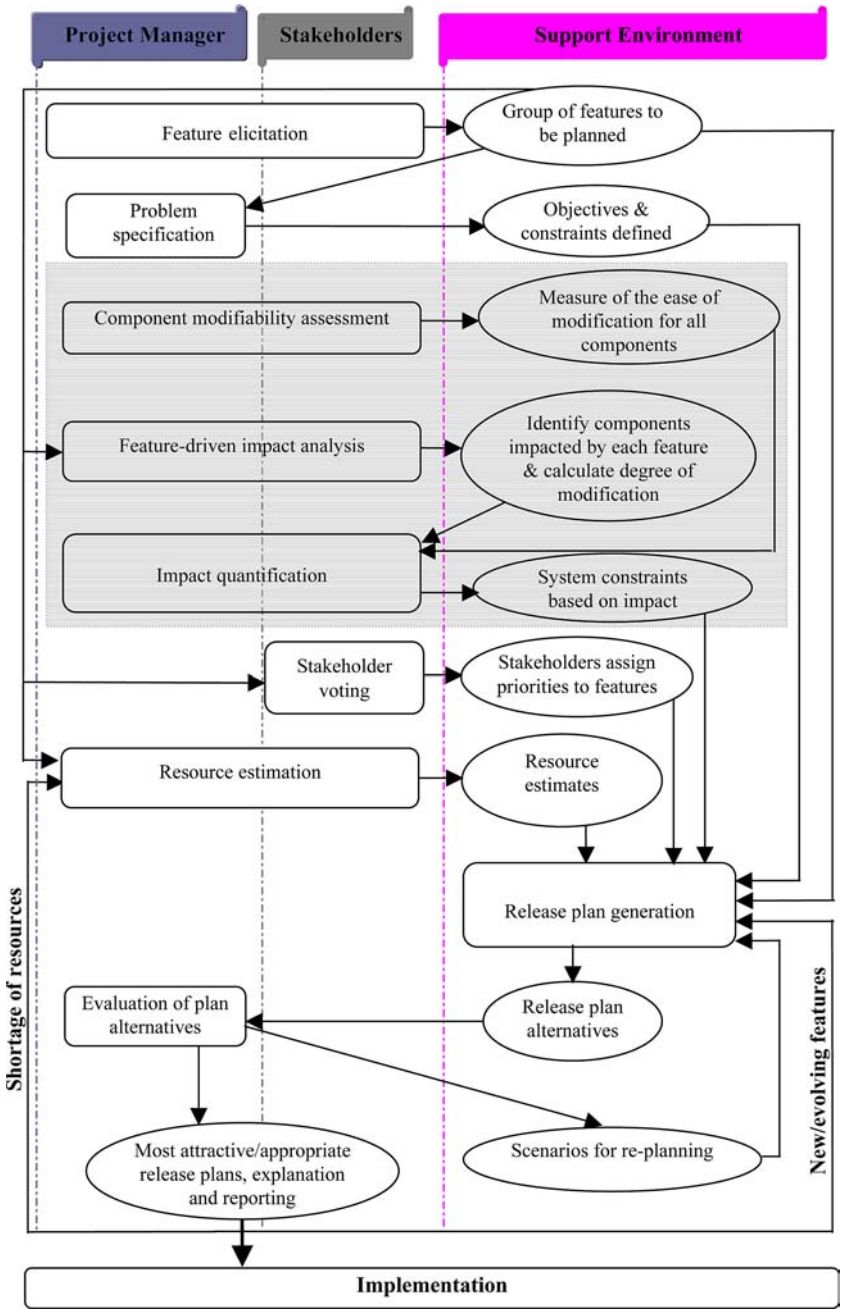


Fig. 4 The S-EVOLVE* process model

speed up the process of generating sufficiently good solutions. The solutions generated are guaranteed to be 95% of the best achievable for the started objective function. Among all these candidate solutions, the algorithm determines a subset that is maximally diversified in terms of its structure.

4.2.9 Evaluation of release plan alternatives

Analysis of sets of generated release plan alternatives is done during this task. If any of the plans meets the set objective and satisfies a sizable number of stakeholders, according to the decision-maker's benchmark, implementation of the selected plan follows. Otherwise, different replanning scenarios are formulated; which could involve redefinition of different parameters influencing release plan generation. Another set of release plan alternatives resulting from the replanning scenarios are generated, and the analysis is repeated. The result of this cycle of activities within the analysis task is a decision on the release plan to implement.

4.2.10 Implementation

This step refers to the actual realization of the generated plans. It is external to the planning process, but it takes the plans as input. Release plans usually do not remain stable over a project's lifetime—new features may surface while other features could evolve during implementation. Therefore, regeneration of the plan may be required to modify the release plan according to the changing environment.

5 Case study: planning of releases for the ReleasePlanner system

5.1 The evolving system under investigation

To illustrate the proposed method, we present a case study based on a real-world system. ReleasePlanner is a cutting-edge decision support system able to perform web-based release planning and prioritization based on comprehensive stakeholder involvement. It combines innovative ideas from computational intelligence, mathematical optimization, multi-criteria decision aid, and intelligent decision support systems. The system comprises eight main components and is more than 80k source lines of code in size. The system runs on two different Linux-based multi-threaded servers. The first server runs the database and the interface to it written in Java along with other business logic written in Java as well. The first server also runs the Apache web server which hosts the hypertext preprocessor (PHP) files for the web-based interface. The second server runs the optimization component and algorithms written in C and C++ which connect to the Java layer on the other server using the Java Native Interface (JNI) API. The overall effort in research and development incorporated into the system is about 22 person-years.

Among the companies who have performed trial or professional projects using the tool include: Corel iGrafx, Siemens Corporate Technology, Trema Laboratories, Nortel Networks, Solid Technology, City of Calgary, Autotech, and Ericsson Canada.

The presented case study involves performing release planning for the ReleasePlanner tool. To keep the size reasonable, we have reduced the number of features to 33. Four stakeholders have been involved in prioritization of the features. Because of space limitations, we do not provide the details of their voting.

There were five types of resources to be considered: analysts, developers, tester, user interface, and research types of resource. The detailed resource consumptions of the features are omitted. The capacities were defined according to the existing capacities for the next two releases.

5.2 Case study design

In planning the study, a small workshop was held to present the challenges of release planning for evolving systems. Our proposed method was also presented to the experts since they will be involved in the assessment of DoM and XoM for components of the system. After the presentation, the expert participants were asked to identify the components making up the system. The following eight components were identified: Reporting component, validator component, IP component, java broker component, import/export component, stakeholder voting analysis component, database (DB) connectivity component, and the alternative analysis wizard component.

To enable us accommodate different perspectives about the difficulty of modifying each of the components, the participants were given instructions to carry out their assessment of DoM independently and with no interactions, but following the AHP-based scheme we have discussed. The relative importance of the technical development team members participating in the experiment were provided by the project manager according to the developers' expertise in the project at hand: Expert-A (importance weight = 0.5), Expert-B (0.3), and Expert-C (0.2).

5.3 Assessment of difficulty of modification

The three experts first performed the pairwise comparison of the five modifiability factors in level 2 according to how important they are in determining DoM. Table 3 gives the results of this computation from the perspective of each of the experts. From the table, Expert-A has ranked the health of component to be more important in determining DoM than any other factor, and he has also ranked size as the least important in determining modifiability. In the case of Expert-B, understandability is considered the most important of all factors determining DoM. However, there is a general consensus among the three experts on the fact that size is the least important of all the modifiability factors.

Having established the priority vectors of the modifiability factors, each expert performed a pairwise comparison of the eight components with respect to each modifiability factor according to the guideline given in Sect. 3. For example, the relative comparison of the eight components with respect to complexity as a modifiability factor from the perspective of Expert-A is given in Table 4. The entries in the table are the AHP pairwise comparison between any two components in the row and column. The last column in Table 4 gives the priority vector indicating the relative difficulty of modifying

Table 3 Matrix of relative importance of modifiability factors according to three experts

Modifiability criteria	Expert		
	Expert-A	Expert-B	Expert-C
Size	0.053	0.049	0.022
Complexity	0.114	0.083	0.473
Understandability	0.251	0.517	0.271
Health	0.394	0.091	0.173
Criticality	0.188	0.259	0.061

Table 4 Pairwise comparison of modifiability of components with respect to complexity from the perspective of Expert-A

Complexity	Reporting	Validator	IP component	Java brokers	Import/export	Stakeholder voting	DB connectivity	Alternative analysis	Component priority vector
Reporting	1	8	8	5	1	1	5	2	0.236
Validator	1/8	1	3	5	1/7	1/5	1	1/3	0.060
IP component	1/8	1/3	1	1/3	1/7	1/7	1/3	1/3	0.025
Java brokers	1/5	1/5	3	1	1/5	1/5	1	1/5	0.038
Import/export component	1	7	7	5	1	1/3	4	1/3	0.169
Stakeholder voting analysis	1	5	7	5	3	1	5	2	0.251
DB connectivity class	1/5	1	3	1	1/4	1/5	1	1/5	0.044
Alternative analysis wizard	½	3	3	5	3	1/2	5	1	0.179

Table 5 Priority matrix of the eight components from the perspective of the three experts

	Expert (weight)				
Component C(m)	Expert-A (0.5)	Expert-B (0.3)	Expert-C (0.2)	Global priority vector XoM(C(m))	DoM(c(m))
Reporting	0.126	0.111	0.075	0.111	
Validator	0.112	0.058	0.169	0.107	
IP component	0.087	0.209	0.435	0.193	
Java brokers	0.173	0.088	0.051	0.123	
Import/export component	0.158	0.230	0.063	0.160	
Stakeholder voting analysis	0.145	0.112	0.055	0.117	
DB connectivity class	0.086	0.086	0.091	0.087	
Alternative analysis wizard	0.113	0.106	0.061	0.100	

the components with respect to complexity. This last column was derived from the eigenvector of the 8×8 comparison matrix. This same procedure is repeated by Expert-A with respect to the four other criteria. Each of the priority vectors derived from each criteria form the column of Table 4 that shows the results for the five criteria from Expert-A's perspective. The priority vector in the last column of Table 4 shows the DoM from the perspective of Expert-A when all the criteria are taken into consideration. Each of the other participating experts performs the same exercise and finally generates similar tables as discussed for Expert-A.

After each of the experts has created similar priority vectors as given in Table 4, the weighted aggregation of those three priority vectors taking the relative importance of the three experts into consideration is given in Table 5. The priority vector in the last column of the table now establishes the final DoM of each component in the corresponding row.

5.4 Computation of the total impact

The DoMs have been established for each of the components independent of whatever feature that would impact them. Impact computation must be done for each feature to be implemented, which requires the evaluation of DoM and XoM in the context of the components that will be modified to implement the features.

To enable a more informed decision on which features will modify which components, the features were assigned to the three experts that would be responsible for the implementation of each feature, if the feature were to be implemented. We have carried out this activity differently than the way the assessment of DoM was performed, since it would be a less robust and objective method to elicit divergent views on the extent of modification from a expert that is not implementing a feature. But the argument holds for DoM because any of the experts would be familiar with the components and would have to work with them at one point or another.

Table 6 shows the set of features in the rows and the corresponding extent of modification required on the components appearing in the columns, which is measured as a percentage of the size of the component. An empty entry in a column of a component means that the component is not impacted by the feature appearing in the corresponding row. The DoM, XoM and Impact values of each feature appear in the last three columns of the table and were calculated using Eqs. 8–10.

5.5 Generation of plan alternatives

We have used ReleasePlanner to generate a set of alternative release plans for the two cases of (1) considering or (2) not considering system constraints. We will denote the solutions gained out of these computations as RPA1 and RPA2, respectively. As known from optimization theory [30], the objective function value of an optimal solution of RPA1 is less than or equal to the value of optimal solutions of RPA2. For the consideration of system constraints, we have used the results of the DoM and XoM computations as described in Subsects. 5.3 and 5.4, respectively. We introduced threshold boundaries to describe the acceptable level of impact caused by implementation of features. In Table 7, we present the

generated five alternative release plans for three levels of thresholds

(i) $\beta(1) = \beta(2) = 2.0$ (Impact = 2.0)
(ii) $\beta(1) = \beta(2) = 3.9$ (Impact = 3.9)
(iii) No consideration of impact (No impact)

The numbers in the table can be interpreted according to whether a feature is assigned to the next release ('1'), next-but-one release ('2') or postponed ('3').

We performed some further investigation of the sensitivity between the impact capacity threshold and the objective function value for the resulting problem. Figure 5 shows the results.

5.6 Interpretation of results

From the sequence of release plan alternatives generated for the different levels of impact we can make the following observations:

- There are substantial changes in the structure of the alternative solutions between all the three cases (i), (ii), and (iii). This clearly indicates the strong impact of system constraints at the release plan strategies.
- There are features assigned to the next release ('1') when system constraints are not considered but are suggested to be postponed ('3') for both variants of consideration of system constraints (features with IDs 1, 2, 7, and 11). The reason for that is that all these features have a relatively strong impact on the affected components (high value of impact).
- There are features mostly assigned to postponed ('3') without consideration of system constraints but are suggested for releases 1 or 2 for both variants of consideration of system constraints (features 10, 15, 19, and 22). In these two cases, features have lower relative impact on affected components than most of the other features.
- For some of the features the consideration of system constraints does not influence their proposed assignment to releases (features with IDs 4, 9, 14, 18, 20, 21, 22, 23, 24, and 28). The interpretation here is that the other factors influencing the assignment of features to a release (stakeholder priorities, resource consumption) are dominating the influence of the impact on the system.
- The more rigorously we take into account the system constraints (e.g., the more the threshold value becomes), the lower the value of the objective function and vice versa. This trend is depicted in Fig. 5. We can argue that the lower threshold level also relates to a lower level of inherent risk as the affected components are judged to be easier to modify.
- Within each set of five alternative solutions, there are structural differences as well. This is the justification to generate not just one, but a set of alternative solutions. This increases flexibility in decision making.
- Variance in the structure between the cases (i), (ii), and (iii) is bigger than among the five generated qualified solutions within each of the cases.

The above interpretations are closely related to the data and structure of the case study. However, we expect these to be general tendencies occurring more or less in the same way for systems having differences in the degree and difficulty of modification of components affected by the implementation of features.

6 Conclusions and future research

In this paper, we have presented a comprehensive release planning approach called S-EVOLVE*. It is characterized by providing nearly optimal solutions based on integer linear programming combined with heuristics as part of a branch-and-bound algorithm. Besides its ability to consider resource and technological constraints, the approach also considers characteristics of the target components for the implementation of features which results in plans more likely to meet the actual needs.

As part of this method, we have developed an approach for determining the difficulty of modification of the system components. For that purpose, we have developed an initial qualitative framework of driving factors for the difficulty of modification of components. Based on that, we used pairwise comparison to quantitatively determine the relative value of the difficulty of modification for the components of the system.

Our future research will be directed to refine and empirically evaluate the proposed framework. We will look at the validity of the actual predictions in terms of the effort actually needed to implement the features. This requires a mapping between the relative extent of difficulty determined so far and the additional effort needed for integrating features into more difficult parts of the system. We are currently investigating the possibilities of developing a generic and more sophisticated taxonomy scheme for classifying the modifiability of system components.

Further empirical investigations are needed to demonstrate the applicability of S-EVOLVE* and to determine its limitations. A current weakness in the methodology is that the impact quantification and the related constraints are relative measures between components. However, absolute quantifiers coming from careful analysis of historical data would help to characterize the difficulty of modification of a component in absolute terms.

In the proposed release planning methodology, the release dates are assumed to be known or predetermined externally. Another direction of research is devoted to addressing open-scope release planning. In that case, determining the release date will be part of the decision-making and optimization process.

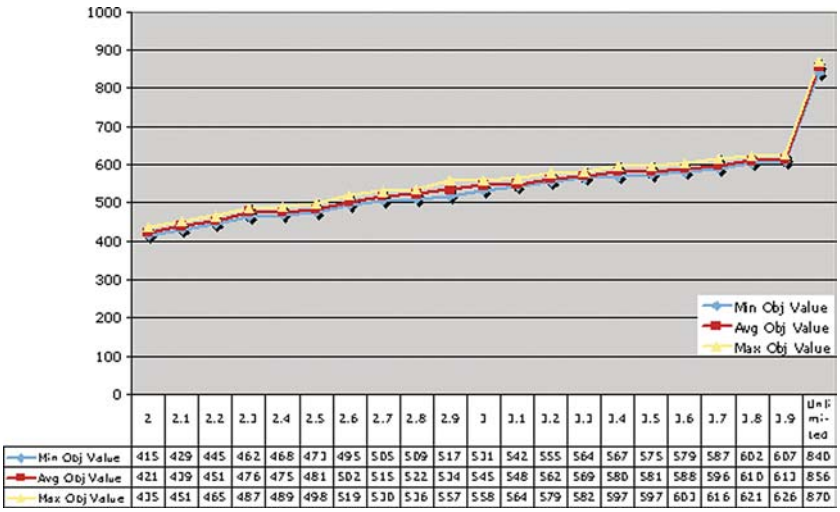
Acknowledgements The Authors wish to acknowledge financial support of Alberta Informatics Circle of Research Excellence (iCORE). We appreciate the support of Pankaj Bhawnani in setting up the case study and the whole ReleasePlanner team for their involvement in running the case study. We would like to thank David Goodladd for performing computations for the case study and also Jim McElroy for his feedback.

Table 6 Set of features showing the groupings, DoM, XoM and Impact for each feature $f(i)$

		Reporting	Validator	IP component	Java brokers	Import/ export component	Stakeholder voting analysis	DB connectivity class	Alternative analysis wizard			
	EoM(C(j))	0.111	0.107	0.193	0.123	0.160	0.117	0.087	0.100			
ID	Feature f(i)	Percentage in size of modification of the component (column) required by implementing the feature (row)								DoM	XoM	Impact
1	Comparative analysis manual versus RP solution	20		10	10	10		10	150	4.392	0.566	2.485
2	Acess to history of sets of alternative solutions	20			25	10		30	75	3.754	0.462	1.734
3	Conformance measure of requirements across alternative solutions	20			15			10	25	1.898	0.359	0.682
4	Comparison between solution	20			15			40	200	5.796	0.359	2.083
5	MS Excel advanced compatibility				20	200		20		4.320	0.327	1.414
6	MS Project compatibility				20	200		20		4.320	0.327	1.414
7	Analysis of compatibility requirements with existing RM tools				20	200		20		4.320	0.327	1.414
8	Generation of solutions based on selected criteria	10		15	15			10		1.600	0.426	0.681
9	Planning across projects	20	30	5	15		20	25	10	3.108	0.592	1.841
10	Replanning capabilities	20							50	1.800	0.200	0.360
11	Splitting of features	10	20	5	5			30	50	2.838	0.538	1.528
12	Accomodation of different skill for human resources	20	20	20	20	20	10	20	30	4.270	0.658	2.808
13	Fuzzy boundaries	20			10			10	25	1.815	0.359	0.652
14	Value–risk trade-off analysis	20						5	10	1.386	0.270	0.374
15	Multiple windows accessibility							10		1.100	0.087	0.096
16	Project data in MS Excel					200		15		3.450	0.233	0.804
17	Dashboard				60			30		2.080	0.199	0.415
18	UI redevelopment	10					10		10	1.331	0.294	0.391
19	Extended reporting component	30						10		1.430	0.188	0.269
20	Explanation component	0						10		1.100	0.087	0.096
21	Visualization of output	50					20	10	20	2.376	0.355	0.843
22	Context-sensitive explanation of terms							10		1.100	0.087	0.096
23	Further development of the validator to give on demand help		20	10	20			10		1.742	0.423	0.737
24	Fine tuning of optimization algorithms			20	5					1.260	0.292	0.368
25	Refactoring			500						6.000	0.193	1.158
26	Caching mechanisms			20	10					1.320	0.292	0.386
27	Candidate feature proposals	5						5		1.103	0.188	0.208
28	Resource estimates	5						5		1.103	0.188	0.208
29	Multiple stakeholder weights	5				5	20	10		1.455	0.398	0.579
30	Improved stakeholder conformance	10		10	10			5	20	1.677	0.483	0.810
31	Individual stakeholder voting feedback	5					10	5		1.213	0.283	0.344
32	Competitor stakeholder voting	10					15	5		1.328	0.283	0.376
33	Stakeholder voting analysis extension	20					100	10		2.640	0.283	0.748

Table 7 Structure of the five diversified alternative solutions for three levels of impact thresholds

ID	Impact=2.0					Impact=3.9					No impact				
	Alternative					Alternative					Alternative				
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
1	3	3	3	3	3	3	3	3	3	3	1	1	1	1	1
2	3	3	3	3	3	3	3	3	3	3	1	1	3	1	1
3	3	3	3	3	3	1	1	3	1	1	1	1	1	1	1
4	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
5	3	3	3	3	3	3	3	3	3	2	2	2	2	2	3
6	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2
7	3	3	3	3	3	3	3	3	3	3	1	1	1	1	1
8	3	3	2	3	3	2	2	1	3	2	2	2	2	2	2
9	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
10	1	1	1	1	1	1	1	1	1	1	3	3	3	3	3
11	3	3	3	3	3	3	3	3	3	3	1	1	1	1	1
12	3	3	3	3	3	3	3	3	3	3	3	3	3	3	2
13	3	3	3	3	3	3	2	3	3	3	2	2	2	2	2
14	1	2	1	2	2	1	1	1	1	1	1	1	1	1	1
15	1	2	1	2	1	1	1	1	1	1	1	2	1	1	1
16	3	3	3	3	3	2	3	2	3	3	2	2	2	2	2
17	3	3	3	3	1	1	1	1	1	1	1	1	1	1	1
18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
19	2	1	2	2	2	2	2	2	1	3	3	3	1	3	3
20	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
21	3	3	3	3	3	2	3	3	2	3	3	3	3	3	3
22	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
23	2	2	3	3	3	2	2	2	2	2	2	2	2	2	2
24	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
25	3	3	3	3	3	3	3	3	3	3	1	1	1	1	1
26	2	2	3	2	3	2	2	2	2	2	2	2	2	2	2
27	2	2	2	3	2	2	3	2	2	2	2	2	2	3	1
28	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
29	3	3	3	1	2	1	1	2	1	1	1	1	1	1	1
30	3	3	3	3	3	3	3	2	2	3	2	2	2	2	2
31	2	1	2	2	2	1	2	1	2	2	2	2	2	1	2
32	3	3	2	2	3	3	1	1	2	1	1	1	1	2	2
33	3	3	3	3	3	3	2	3	3	3	2	2	2	2	2
Value	435	420	419	417	415	626	615	609	607	607	870	866	858	845	840



	2	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9	3	3.1	3.2	3.3	3.4	3.5	3.6	3.7	3.8	3.9	Unlimited
Min Obj Value	415	429	445	462	468	471	495	505	509	517	531	542	555	564	567	575	579	587	602	607	840
Avg Obj Value	421	439	451	476	475	481	502	515	522	534	545	548	562	569	580	581	588	596	610	613	856
Max Obj Value	435	451	465	487	489	498	519	530	536	557	558	564	579	582	597	597	603	616	621	626	870

Fig. 5 Sensitivity between the impact capacity threshold and the objective function value

References

1. Bagnall AJ, Rayward-Smith VJ, Whittley IM (2001) The next release problem. Inform Software Tech 43(14):883–890
2. Ruhe, G (2005) Software Release Planning. In: Handbook of software engineering and knowledge engineering, vol. 3. World Scientific Publishing
3. Saliu O, Ruhe G (2005) Supporting software release planning decisions for evolving systems. In: Proceedings of 29th IEEE/NASA software engineering workshop, Greenbelt, MD, USA, 6–7 April
4. Saaty, TL (1980) The analytic hierarchy process. McGraw-Hill, New York
5. Penny DA (2002) An estimation-based management framework for enhancive maintenance in commercial software products. In: Proceedings of international conference on software maintenance (ICSM'02), Montreal, Canada, 3–6 October, pp. 122–130
6. Denne M, Cleland-Huang J (2004) The incremental funding method: data driven software development. IEEE Softw 21(3):39–47
7. Karlsson J, Ryan K (1997) A cost-value approach for prioritizing requirements. IEEE Softw 14(5):67–74
8. Jung H-W (1998) Optimizing value and cost in requirements analysis. IEEE Software, pp. 74–78
9. Greer D (2004) Decision support for planning software evolution with risk management. In: Proceedings of 16th international conference on software engineering and knowledge engineering (SEKE'04), Banff, Canada, pp. 503–508
10. Ruhe G, Ngo-The A (2004) Hybrid intelligence in software release planning. Int J Hybrid Int Syst 1(2):99–110
11. Jilles van-Gurp Bosch J, Svahnberg M (2000) Managing variability in software product lines. In: Proceedings of LAC 2000, Amsterdam
12. Carlshamre P (2002) Release planning in market-driven software product development: provoking an understanding. Requirements Eng 7:139–151
13. Graves TL, Karr AF, Marron JS, Siy H (2000) Predicting fault incidence using software change history. IEEE Trans Softw Eng 26(7):653–661
14. Eick SG, Graves TL, Karr AF, Marron JS, Mockus A (2001) Does code decay? Assessing the evidence from change management data. IEEE Trans Softw Eng 27(1):1–12
15. Gall H, Hajek K, Jazayeri M (1998) Detection of logical coupling based on product release history. In: IEEE international conference on software maintenance. Washington DC, pp. 190–198
16. Porter AA, Selby RW (1990) Empirically guided software development using metric-based classification trees. IEEE Softw 7(2):46–54
17. Mockus A, Weiss DM (2000) Predicting risk of software changes. Bell Labs Tech J 5(2):169–180
18. Gall H, Jazayeri M, Klosch RR, Trausmuth G (1997) Software evolution observations based on product release history. In: Proceedings of international conference on software maintenance (ICSM'97), Bari, Italy, Oct. 1–3, pp. 160–166
19. Yu TJ, Shen VY, Dunsmore HE (1988) An analysis of several software defect models. IEEE Trans Softw Eng 14(9):1261–1270
20. Lehman MM (1980) On understanding laws, evolution and conservation in the large program life cycle. J Syst Softw 1(3):213–221
21. Van Scoy RL (1992) Software development risk: opportunity, not problem. Software Engineering Institute, Pittsburgh, (CMU/SEI-92-TR-30, ESC-TR-92-030)
22. Ohlsson MC, Andrews AA, Wohlin C (2001) Modelling fault-proneness statistically over a sequence of releases: a case study. J Softw Maintenance Evol Res Pract 13:167–199
23. Mens T, Demeyer S (2001) Future trends in software evolution metrics. In: Proceedings of 4th international workshop on principles of software evolution (IWPSE'01), Vienna, Austria, pp. 83–86
24. Lehman MM (1996) Laws of software evolution revisited. In: Proceedings of 5th European workshop on software process technology (EWSPT'96), Nancy, pp. 108–124
25. McCabe TJ (1976) A complexity measure. IEEE Trans Softw Eng 2(4):308–320
26. Halstead MH (1979) Elements of software science. Elsevier, Holland
27. Schneidewind NF, Hoffman HM (1979) An experiment in software error data collection and analysis. IEEE Trans Softw Eng 5(3):276–286
28. Dayani-Fard H (2003) Quality-based software release management. Queen's University, Canada, Kingston, Ontario, PhD Thesis
29. Christel MG, Kang KC (1992) Issues in requirements elicitation. SEI, Carnegie Melon University, Pittsburgh, CA, CMU/SEI-92-TR-12
30. Wolsey LA, Nemhauser GL (1998) Integer and combinatorial optimization. Wiley, New York