Supporting Software Release Planning Decisions for Evolving Systems

Omolade Saliu and Guenther Ruhe
Laboratory for Software Engineering Decision Support
University of Calgary, 2500 University Drive NW,
Calgary, AB T2N 1N4, Canada
{saliu, ruhe}@cpsc.ucalgary.ca

Abstract

Large-scale software systems constantly change during system evolution for feature enhancement. Most of the features originate from diverse stakeholders that require their needs to be met despite resource and risk constraints. In such large systems, the number of features requested during the different releases of the system typically exceeds the available resources. Release planning involves decision making about what new features or changes to implement during which release of the software. Existing release planning techniques are not targeted at evolving systems; in this case, knowledge about existing software product is core to making meaningful release decisions.

In this paper, we describe ten key technical and non-technical aspects impacting release planning. Based on these aspects, we evaluate seven existing release planning methods. We have also proposed a new release planning framework that considers the effect of existing system characteristics on release planning decisions. Initial realization of this framework focuses on historical defect data to characterize the health of system components. This proposed approach extends the existing solution method called EVOLVE by (i) the proactive analysis of the risk involved in integrating new features into existing components of the system and (ii) identifying the importance of estimating the integration effort for each feature based on system characteristics. An illustrative example is also presented.*

Keywords: Software Evolution, Release Planning, Decision Support, ReleasePlanner®, Effort Estimation, Risk.

1. Introduction

Incremental software development offers sequential releases of software systems with additive functionalities in each increment. This approach allows customers to receive part of a system early. Thus, each increment is a collection of features that forms 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 that of determining what should be in different releases of the software [1].

Release planning for incremental software development assigns features to releases such that most important technical, resource, risk and budget constraints are met. Release planning, therefore, generalizes pure prioritization of features or requirements [22]. Release planning is a very complex problem including different stakeholder perspectives, competing objectives and different types of constraints.

The release planning problem is "wicked" [4]in that the objective is "to maximize the benefit", but it is difficult to give a measurable definition of "benefit". This also means the more you dive into the problem, the more complex it becomes. In this paper, we initially abstract from too many technical details and describe the main components impacting software release planning. We will denote the generic model by ReleasePlanning [A,B,C,D,E,F,G,H,K,L] indicating its dependency from ten different dimensions [A…L]. We have also initiated a release planning approach for evolving software systems.

The rest of the paper is organized as follows: In Section 2, we will briefly describe all these 10 dimensions. In Section 3, we will briefly discuss the contribution of existing release planning methodologies to the ten dimensions; an instantiation of the above generic model is also presented. From the solution presented it becomes clear that current approaches have weaknesses in taking the characteristics of evolving systems into account. In Section 4, we provide a formal approach to evaluate the development risk associated with implementing a feature based on the historical defect involving the components that will host the feature. Section 5 presents an illustrative example. Finally, an agenda for future research in the context of the current work is discussed in Section 6.

2. Key Aspects of Software Release Planning

The most relevant aspect of release planning to be considered when developing methodologies to address the problem are discussed below:

2.1 Scope (dimension A)

It has been observed that planning for only one release (i.e. next one) is not enough [4]; dissatisfactions are highly likely to arise. Some stakeholders would be disappointed seeing their highly ranked features not in the next release, and no planned schedule to have the need met in the future. It is a good idea to plan two or more releases in advance, depending on increment interval (release calendar). Because of feature volatility, however, it does not make sense to plan too many releases ahead. In other to monitor progress within a release, incremental planning within each release is equally necessary.

2.2 Time Horizon (dimension B)

The time horizon of planning refers to the time interval required for a product or part of a product to be released – in essence, the release cycle. There are two fundamental types of release planning problems: (i) release planning with fixed and pre-determined time interval for implementation, and (ii) planning with flexible intervals. In the second problem type, the decision about the length of the interval to implement all the assigned features is not pre-determined. Most release planning approaches available today focus on fixed release cycles. A desirable characteristic is to develop flexible release planning models that would operate in a situation where increment delivery interval cannot be ascertained ahead of time, but dictated by other internal/external factors.

2.3 Objectives (dimension C)

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.

2.4 Stakeholder Involvement (dimension D)

A stakeholder is any individual or organization that has interest in a software development project. Stakeholders can be end users, developers, customers, testers, project managers, and so on. Involvement of stakeholders is of vital importance for the development of realistic and high-quality release plans that address real customer needs. The question of determining the 'right' stakeholders is difficult to answer. In most cases, stakeholders are not sufficiently involved in the planning process. This is especially true for the final users of the system.

2.5 Prioritization Mechanism (dimension E)

This addresses the manner in which the actual priorities of the features should be expressed relative to the rest. This includes the question of an appropriate scale and granularity as well as the actual access rights for stakeholders to vote on features based on their preferences. Typically, prioritization is done without considering efforts, risks or other demands for its implementation. A voting scheme allows all categories of stakeholders to prioritize features by voting on them. Stakeholders may vote to merely prioritize the features based on their values or vote to reflect urgency with which the features are desired in an earlier increment. Some prioritization schemes may be based on the analytic hierarchy process (AHP) [23].

2.6 Technological Constraints (dimension F)

A study of requirements repositories in Telecommunications domain by Carlshmare *et al.* [4] observed that only about 20% of the requirements were singular or independent of each other. They further discussed different types of possible requirements interdependencies. Even when characterized by independent features, RP problem is said to be a difficult problem [1]. This conclusions show that RP models must identify different interdependencies between features. For example, there could be coupling dependency in which case two or more features must be implemented together in the same release; precedence dependency occurs when a Feature X must be implemented before Feature Y or vice-versa. In fact, a top-priority feature may require that a low-priority feature be implemented first. These interdependencies become more complex in a situation where two or more increments are to be planned ahead.

2.7 Resource Constraints (dimension G)

A project manager has to consider various resource constraints while allocating the features or requirements to various releases. Most frequently, these constraints are related to different types of resources, schedule, budget, risk or effort. Resources typically are in one-to-one correspondence to the essential roles in software development and evolution (such as analysis and design, development, testing, documentation).

2.8 System Constraints (dimension H)

Release planning is a central activity to the evolution of software products. RP to select and assign features into increments would not be realistic without considering the effects of these features on existing system. The sources of information may be the existing architecture, code base, change repository and defect history, and so on. These types of information are necessary for impact analysis to assess and quantify the extra effort, risk and complexity trade-offs necessary when incorporating new features into existing system.

2.9 Character and Quality of Solutions Offered (dimension K)

This dimension addresses the question of what is considered and accepted to be a solution of the problem. The range is quite large: A single solution versus a set of alternatives? An *ad hoc* solution that is not necessarily feasible versus a solution that is actually satisfying all (or most) of the constraints? Or a solution with undefined quality versus a solution that achieves a predefined level of optimality?

2.10 Tool Support (dimension L)

Release planning is a human and knowledge intensive process, which has a number of tedious tasks like resource estimation, conflicting stakeholders objectives, all of the other factors in the dimensions discussed above, release plan generation, and decision evaluation. The overheads of such tasks call for intelligent tool support that would be of great value to a project manager required to make release decisions.

3. Evaluation of Release Planning Methodologies

A number of release planning methods have been developed - some are targeted at the release planning problem and others as mere requirements prioritization and selection techniques. Software release planning method that we evaluate in this paper, based on the dimensions above are:

- Estimation-Based Management Framework for Enhancive Maintenance (EMFEM)[1] [17]
- Incremental Funding Method (IFM) [5],
- Cost-Value Approach for Prioritizing Requirements (COVAP)[1] [11],
- Optimizing Value and Cost in Requirements Analysis (OVAC)[1] [10],
- The Next Release Problem (NRP)[1] [1],
- Planning Software Evolution with Risk Management (PSERM)[1] [9], and
- Hybrid Intelligence (EVOLVE*) [19].

[1] The abbreviation used here is merely for convenience of our evaluation table and is not originally named as such.

These seven methods are considered to be representative of the current state-of-the-art in software release planning. Because of space limitation, we will not discuss the details of each method here; appropriate references serve as pointers to interested readers. What we discuss here is an evaluation of the contribution of each method to the release planning dimensions presented above. Table 1 gives a summary of the RP methodologies and their contributions to release planning dimensions.

3.1 Discussion of Comparison Results

A quick look at Table 1 reveals main deficits in the existing release planning methods. We specifically outline three deficits as follows:

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. There are not enough decision support tools that are fully developed and based on sound formalism, except ReleasePlanner®.
3. Release planning has been largely focused on "fixed release intervals" and no current work on planning with flexible time intervals.

Among the three problems identified above, the one concerning system constraints is the most critical in the context of evolving software systems. It is highly unrealistic to plan system evolution without considering history of the underlying system. This constitutes the major gap that this paper seeks to address.

Our proposed approach will extend the hybrid intelligence method called EVOLVE* [7]. We aim to enhance the performance and suitability of ReleasePlanner® for system evolution planning by incorporating historical information about the system.

3.2 ReleasePlanner® - Instantiation of the Generic Model

3.2.1 Overview

ReleasePlanner® (www.releaseplanner.com) is a tool suite that provides a flexible and web-based support for release planning. The tool was developed in the Laboratory for Software Engineering Decision Support (http://www.sengdecisionsupport.ucalgary.ca), University of Calgary, Canada. The overall architecture of this approach called EVOLVE* [19]is designed as an iterative and evolutionary procedure mediating between the real world problem of software release planning, the available tools of computational intelligence for handling explicit knowledge and crisp data, and the involvement of human intelligence for tackling tacit knowledge and fuzzy data. We will briefly discuss the instantiation of the above generic release planning dimensions by showing the values for all of the dimensions, as implemented in EVOLVE*.

3.2.2 *Decision Variables*

Suppose we have a collection of features F = {*f(1), f(2), ... , f(n)*}, the objective of EVOLVE* model is to distribute the features into one of three options: "next release (option 1)", "next but one release (option 2)", and "not yet assigned (option 3)". The problem is characterized by a number of decision variables and constraints that include: coupling and precedence constraints between features, stakeholders' preferences, and resource requirements.

The modeling approach consists of decision variables *{x(1), x(2), ..., x(n)}* with *x(i)* = *k* if requirement *i* is assigned to release option *k*.

3.2.3 *Technological Constraints*

Two types of technological constraints are considered where features are either in a coupling relation called C or in a precedence relation called P. Both relations are subsets of the product set F*F:

$$x(i) = x(j)\ \forall(i.j) \in C\ \subset \mathrm{F} * \mathrm{F}\ (\textit{Coupling})$$

$$x(i) \le x(j)\ \forall(i.j) \in P\ \ (\textit{Precedence})$$

3.2.4 *Stakeholder*

Stakeholders are extremely important in order to perform realistic and customer-oriented release planning. We assume a set of stakeholder S = {S(1),…,S(q)}. Each stakeholder p is assigned a relative importance $\lambda_p \in (0,1)$ and stakeholder weights are normalized to 1, e.g. $\sum_p \lambda_p = 1$.

Table 1: The comparison of various release planning methodologies

RP Methods / Dimension	PSERM [9]	ENFEM [17]	NRP [1]	OVAC [10]	COVAP [11]	IFM [5]	EVOLVE* [19]
Scope	1 release	1 release	1 release	1 release	1 release	Chunks of small releases	2 releases planned ahead
Time horizon	Fixed release	Fixed release	Fixed release	Fixed release	Fixed release	Fixed release	Fixed release
Objectives	Based on benefit of system changes	Based on benefit of features to customers	Based on weight of customers	Based on value of requirements	Based on customer satisfaction	Based on return on investment	Based on value, urgency, stakeholders weights and satisfaction
Stakeholders involvement	Project manager	Developers, project management	Project manager, customer	Involvement of project manager is implied	Project manager, customer, users	All major stakeholders	All major stakeholders
Prioritization mechanism	Optimization-based	No defined prioritization scheme	Optimization-based	Optimization	AHP	IFM Heuristics	Stakeholders prioritize features; Optimization heuristics used to balance conflicts
Technological constraints	Not available	Not available	precedence	Not available	Not available	Precedence (precursor)	Coupling and precedence
Resource constraints	Cost, risk	Effort	Cost	Cost	Cost	Cost, time-to-market	Effort, risk, schedule
System constraints	Operational risks	Not available	Not available	Not available	Not available	Not available	Not available
Character and quality of solutions	One solution plan	One solution plan	One solution plan by any chosen search algorithm	One solution plan generated	One solution plan provided	One plan spanning many release periods	Several alternative solution plans are provided. These plans are diversified and fulfill some target quality level.
Tool support	Not available	Time-tracking system	Not available	Not available	Not available	Partially available	ReleasePlanner®

3.2.5 *Prioritization of Features*

Each stakeholder assigns an ordinal value, $value(p,i) \in \{1,2,\cdots,9\}$ to each feature based on importance of the feature to the specified stakeholder. Any other scale can be used; the 9-point scale we have used here offers enough flexibility and is tractable.

In addition to the value-driven prioritization, we allow urgency-oriented prioritization as well. For that, each stakeholder p has a chance to represent the urgency with which they want each feature assigned to different releases. That is, the satisfaction that feature i is assigned to option k (k=1, 2, 3). This satisfaction is expressed by $sat(p,i,k) \in \{1,2,\cdots,9\}$; giving each stakeholder a total of nine votes to distribute among three options for each feature. The higher the number of votes assigned to k, the more satisfied the stakeholder would be if the feature is put in the respective option.

3.2.6 *Resource Consumptions*

Each feature consumes different types of resources for its implementation. We assume T resource types. Every feature i requires *an* amount of resources (i.e. $r[i,t]$) of type effort, finance, etc. Every release option k (except option 3) has a certain amount of resource capacity of type t available, that is R[k,t]. Thus, the resource requirements for all features assigned to release k must satisfy the following constraints:

$$\sum_{i:x(i)=k} r[i,t] \leq R[k,t].$$

Considering the influencing factors representation above, the overall satisfaction of assigning feature i to option k=1, 2 is given as:

$$c[i,k] = \sum_p \lambda_p \cdot value(p,i) \cdot sat(p,i,k);$$

3.2.7 *Objective*

The objective is to maximize a function *F(x),*

$$F(x) = \xi_1 \sum_{x(i)=1} c[i,1] + \xi_2 \sum_{x(i)=2} c[i,2] \qquad (1)$$

for all release plans x satisfying the above technological and resource constraints with two parameters $\xi(1)$ and $\xi(2)$ describing the relative importance of options 1 and 2, respectively.

The approach we have just described has been able to address release planning problem dimensions discussed above in the realm of new software systems. However, the ever-changing environment in which software systems operate demands constant evolution. Most organizations like NASA concentrate more on the evolution of existing software systems rather than building new system for every problem. Evolving systems environment affects release planning decisions. Therefore, the characteristics of the software system as reflected in the repository of previous release histories must be considered for a realistic and measurement-based decision making.

4. Release Planning for Evolving Systems

4.1 Software Evolution

Software development proceeds as a series of changes to a base set of system – for new projects the base set may be initially empty [15]. 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 correcting errors that are discovered during operation of the software. The evolution is achieved by modifying parts of the software source code.

For many large software systems, the set of features required are typically large that all the features cannot always be accommodated and made available to the customers as market demands. These set of features are therefore grouped into different releases – a task that demands selecting optimal subset of features that satisfies as many stakeholders as possible within the budget, resource and risk constraints. Scaling up to such challenges, therefore, is not an easy task without good release planning that considers the operating environment and existing system base.

4.2 Influence of System Characteristics on Software Evolution Planning

4.2.1 *Background*

The base system of large software consists of interacting components *{C$_1$, C$_2$... C$_m$}*. In order to implement a new feature, one or more of these components must be modified for the integration. Suppose we have a set of features F and set of system components C = *{C$_1$, C$_2$... C$_m$}*. Since one feature could have impact on more than one component, we assume a set-valued mapping Ψ assigning to each feature i a set of impacted components Ψ (i) $\subset$ C.

The process of identifying the components affected by integrating new features is called Feature-Driven Impact Analysis (FDIA), and would largely depend on technical developers that understand the system.

Decision support for planning evolving software system releases needs to consider the historical information about components of a system, rather than recommending features for implementation independent of integration challenges. The repositories of software project and change histories contain valuable information that could be mined to characterize the software components making up the system. In order to plan evolution releases, FDIA must be performed, given the information about components of the system. Figure 1 gives a summary overview of the proposed new release planning approach. The architecture goes some level above the traditional release planning performed independent of system characteristics.

According to Lehman's laws of software evolution [13], software complexity tends to increase while the quality will tend to decrease. Although, the characterization we seek to address here is at individual component (we have loosely defined components here – it could be subsystems, modules etc) level, because fine-grained feature estimation is not influenced by the characteristics of the system as a whole, but those of the individual components. Ramil and Lehman [18]discussed the potentials of coarse-grained historical release information (e.g. subsystems counts, modules counts, components counts, etc.) in predicting software system evolution effort. These counts help prediction at system level, but not individual component level. However, it further justifies our assumption that characterization of each component of a system is possible and would be valuable. The first major challenge is how to characterize the components of a system using the release history? What would constitute the descriptive attributes we can use for characterization? What classification scheme do we adopt?

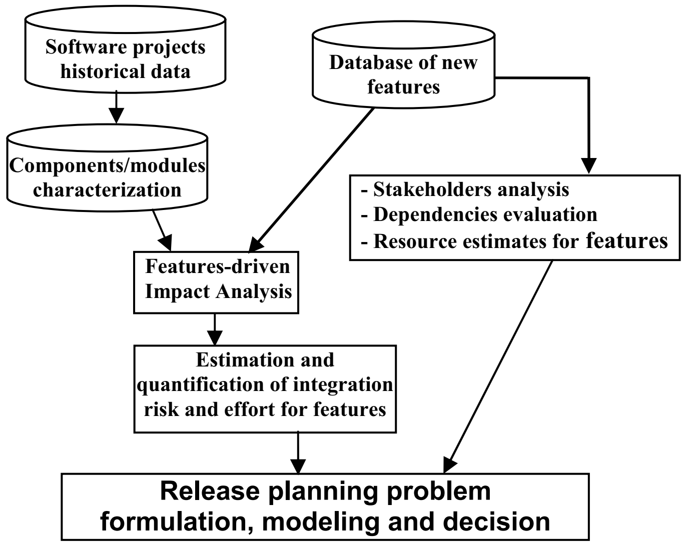
Software projects historical data
Database of new features
Components/modules characterization
- Stakeholders analysis
- Dependencies evaluation
- Resource estimates for features
Features-driven Impact Analysis
Estimation and quantification of integration risk and effort for features
Release planning problem formulation, modeling and decision

Figure 1: Architecture of release planning decision support system

Mens and Demeyer [14] suggest that software metrics may be used to analyze software system with the aim of assessing evolution-critical, evolution-prone and evolution-sensitive parts. Evolution-sensitive refers to parts of the system that can cause problem upon evolution which could result from parts that are interwoven thus demanding more effort to manage impact. This sensitivity of components of the system is of interest in this research. The authors have not proposed methods to measure these two properties of a system, but the definitions could form a starting point in addressing subset of one of the questions we have raised.

Each component of a system should at least be evaluated for complexity, healthiness, operational failure, modifiability, etc. We do not claim that these attributes alone are all-encompassing, but they would form part of our initial focus in this paper. In reality, we will need to derive from the system history, an (m x k) matrix of *m* components and *k* descriptive attributes. Such a matrix will exist for each system operational period between subsequent releases of the software. We will investigate these attributes and other components characterization questions in a future research.

Our main goal for deriving information from existing system components is to use the results during FDIA to measure the integration effort and risk associated with implementing a new feature that modifies such components. Our initial concentration will be on formal assessment of feature risks. In order to account for the risk involved in integrating new features into existing components in light of system characteristics, we hypothesize that:

Hypothesis: *The risk of integrating new feature(s) into any component(s) of a system is a function of the health of the components hosting the features.*

In our definition, the health of component is determined by the operational failure reported against the component during field usage of the software in previous releases. If components are unhealthy, they would be fault-prone and changing any small part of the code can be really risky. Empirical studies [7] reveal that correlation exists between the health of a program code, the quality of resulting product and the functionality that can be added to the system.

The hypothesis stated above is not a widely investigated subject and therefore opens discussion for several empirical investigations. Validation of this hypothesis requires access to real historical project data. The concepts we explore in this paper are some of the usage scenarios of such project data.

4.2.2 Features risk modeling based on components health

Quantification of potential development risk level of each feature cannot be done in isolation. This could be determined through impact analysis that is based on informed decisions about previous histories of the different components making up the architecture of the system. Unfortunately, the likelihood (probability) of requirements failure has largely been based on ability of experts to give *a priori* risk probabilities to each requirement. The risk-based requirement decision developed at NASA [6] and the evolution planning in [9] follow this route. We argue that such probability values are difficult to elicit from experts, but benefiting from historical defect data of the system under consideration, we can compute risk probabilities based on evidence.

Since we desire risk estimation at feature level, information about health of component is more relevant. Health parameters can be derived from post-release defect analysis which reflects operational failure of components during actual usage rather than testing reports. This evaluation should span several periods depending on how many evolutionary cycle the system has gone through. A *period* is the time between successive releases during which the system was with users. These histories are kept in change management systems (CMS) and version control systems (VCS).

Suppose we have a software system that contains *m* components and has gone through *p* releases, several failures must have been experienced that affect each component. Examining the change management repository, we will be able to find the number of defects (defect count may not be the only attribute) issued against each component as shown in Table 2. We will not consider new features that are introduced during a release, but operational defect data reported by the users of the system. The fact that our targeted defect data are not those resulting from development testing, but the actual field failures reported by users should validate the underlying assumptions of relationship between defect data and operational failure.

Table 2: Evolution of components operational defect for a software system

Component ID	**Operational Period #1 (Number of defects)**	**Operational Period #2 (Number of defects)**	**…**	**Operational Period #P (Number of defects)**
C_1	d_{11}	d_{21}		d_{p1}
C_2	d_{12}	d_{22}		d_{p2}
…	…	…	…	…
C_m	d_{1m}	d_{2m}		d_{pm}

The health of each component during specific period *p* can be calculated as a ratio of the number of defects affecting the component and the total defect during the period, as follows:

$$H_p(C_i) = \frac{D_p(C_i)}{\sum_{j=1}^{m} D_p(C_j)} \tag{2}$$

In Equation (2) $H_p(C_i)$ denotes the health of the component at period *p*, and the value is given by the ratio of number of defects against C_i to the total number of defects in that period for all components *m*. $D_p(C_i)$ is the defects count for component C_i in period *p*. Since $H_p(C_i)$ is a probability value; it is guaranteed that $0 \le H_p(C_i) \le 1$, and we can compare health of components within the same period. Our health calculation is independent of the number of components present in any particular period because system components can be added or removed as the system evolves.

The current health $H(C_i)$ of each existing component of the system is taken as a weighted-average of the health over all previous evolution periods. We conjecture here that operational failure of components during the most recent operational period is more critical than in previous periods and should carry more weight. Thus, the more recent the operational period the higher the probability that the faults resulting in failure still exist in the components, and the higher the weight μ_p $(0 \le \mu_p \le 1)$ we attach to the period. The period weights are normalized to 1, that is $\sum_P \mu_p = 1$.

The weighted-average of health over all periods for each component is given by:

$$H(C_i) = \frac{\sum_{p \in P} \mu_p \cdot H_p(C_i)}{|P|} \tag{3}$$

From Equation (3), all the components of the system can be assigned health value $H(C_i)$ $(0 \le I(C_i) \le 1)$, which reflects the risk of modifying the component for new feature integration. The higher the value of $H(C_i)$ the more unhealthy the component is and thus the riskier it is to modify the component to integrate new feature. It should be noted that there is no computation overhead since the health assessment process would be done once. Since our interest is to assign risk values to the features to be developed the component health is just a stepping stone.

A problematic feature is the one that is at risk of causing failure. The likelihood of failure occurring in implementing the feature is derived from the health *H(C)* of all the components it would be integrated with. Thus, we define the likelihood-of-risk of feature f_k as:

$$R(f_k) = H(C)$$

In most cases, implementation of a feature would require modification of several components. Therefore, we must carry out feature-driven impact analysis (FDIA) to identify various components of the system to be affected by the implementation of the new feature. The health of these other components identified would also impact the risk level of the features to be integrated.

In general, for *m* number of components to be modified by integrating feature f_k, the total likelihood-of-risk of the feature is:

$$R(f_k) = 1 - \prod_{i=1}^{m} \left(1 - H(C_i)\right) \tag{4}$$

The impact of the combined health values is never less than either of the combined probabilities while also ensuring that the likelihood of risk of features are normalized ($0 \le R(f_k) \le 1$) for comparative evaluation.

An extension to Equation (4) requires explicit consideration of the severity of each risk as a multiplicative factor in the equation. A possible computation of this could be based on the criticality of the components to be modified. In any system, some parts are mission-critical and have tremendous reliability requirements, while other components may only be minor functionalities that are rarely used or have no negative effect on mission goals. This means that if a bug was inadvertently introduced into one of these low priority components while implementing a new feature, 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 the consequence on the system will likely be far greater and more profound. The criticality rating can be translated to severity of the risk, if desirable.

4.2.3 Release planning model and decision support for evolving systems

At this early stage of the research where initial proposals have been made to consider various information sources in performing release planning for system evolution we still maintain our objective function described in Equation (1). The major added value is that risk assessments would be measurement-based and not mere gut-feelings. The risk information would be valuable in decisions to select feature(s) for implementation. In Section 5, we will discuss a case study example to illustrate the added value in terms of historical evidence-based feature risk assessment.

The long term goal of this research is to improve the decision model described in Equation (1) in order to factor major challenges of evolving systems into such emerging models. In the interim, our efforts would be concentrated on validating the hypothesis put forward in this paper by collecting and experimenting with historical system data as discussed. Without such extensive validation using diverse environment-independent datasets, we cannot categorically draw conclusions from the metrics discussed here. Results from such validation would further enhance the current state of release planning decision support.

5. A Telecom Example

In order to illustrate the effect of risk assessment based on system constraints, we discuss an example taken from a trial project consisting of 13 features for a telecommunication company. The planning process is for the next two releases of the company TELECOM's product development. This project is an abridged version of the original project with 25 features that we discussed in our previous work [20]. The project now has four stakeholders with different concerns. The list of these required features, estimated resource demands and available capacity bounds are given in Table 3. These resources are in the form of effort and capital requirements. The following two technological constraints were identified in the project:

- USEast Inc. Feature 1 is coupled with USEast Inc. Feature 2, and
- China feature 1 must precede China Feature 2.

Table 3: Features, resource consumption and available capacities for the two releases

ID	Feature Name & Group	Developer (person-days)	Testers (person-days)	Documentation (person-days)	Capital Requirement ($K)	Risk Level
	Group 1: China Telecom					
1	16 sector, 12 carrier BTS for China	1330	400	400	1500	0.768
2	China Feature 1	440	200	60	500	0.18
3	China Feature 2	310	190	40	200	0.54
	Group 2: General Features					
4	Common Feature 01	450	200	0	50	0.24
5	Common Feature 02	350	150	50	0	0.44
	Group 3: Strategic Features					
6	Software Quality Initiative	600	400	5	0	0.865
7	Expand Memory on BTS Controller	205	75	20	200	0.33
8	FCC Out-of-Band Emissions	620	200	10	200	0.14
	Group 4: Cost Reduction					
9	Next Generation BTS	1300	500	200	150	0.92
10	Cost Reduction of Transceiver	470	200	60	1000	0.796
	Group 5: USEast Contractual					
11	Pole Mount Packaging	930	400	150	500	0.63
12	USEast Inc. Feature 1	800	400	100	0	0.08
13	USEast Inc. Feature 2	850	250	50	25	0.08
	Total resource consumption:	**8655**	**3565**	**1145**	**4325**	
	Available capacity - Release 1	**4500**	**2200**	**500**	**2600**	
	Available capacity - Release 2	**4000**	**1500**	**500**	**2000**	

In this case study, five essential components have been identified by the software designers, programmers and project manager to be impacted by these 13 features. Using the scanty change history data maintained by the organization, we characterized each of the five components using the health measure discussed in Section 4 of this paper. Table 4 shows the five components and their health measures.

Table 4: Impacted component characterization for health

ID	Components	Health Measure
1	DOM Time Server	0.24
2	DOM Messaging	0.54
3	DOM Scheduler	0.78
4	DOM Monitor	0.18
5	CEM Object Manager	0.08

For every feature in Table 3, different impacted components were identified by the designers. Their recommendations were consistent with our assumption that some features could result in modification of multiple components. The estimated risk values for the features are given in the last column of Table 3. Because of space limitation we have not shown the steps in calculating the risk value for each feature, but Figure 2 shows the components that would be modified to realize each feature. It is therefore straightforward to calculate these risk values directly from the health of components by using Equation (4).

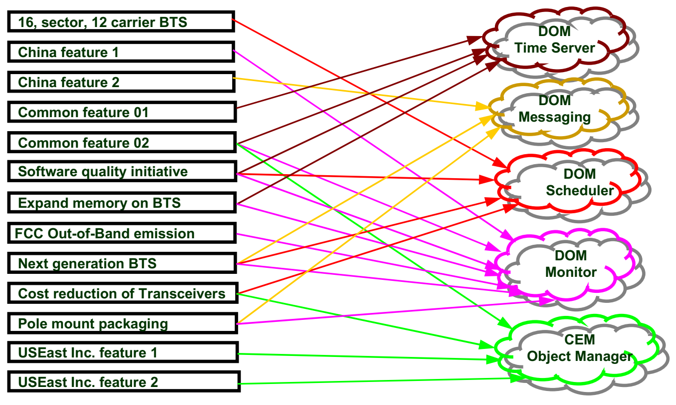

Figure 2: Feature and their associated components in existing system

All the data shown in Table 3 were entered into ReleasePlanner® and the four stakeholders (with different weights based on importance) log in to the tool to vote on the features. In this case study, we initially generated 5 different alternative release plans that are maximally diversified. Diversification of plans refers to a situation in which two solutions are considered more different as more individual features are assigned to different releases. These 5 alternative plans are shown in Table 5. It should be noted that all the alternative plans are in the range of 95% quality as per the objective function. The numbers 1 and 2 in the rows of each plan indicate the releases to which the feature in the corresponding column is assigned. Number 3 indicates postponed features.

Analysis of the five alternative plans is necessary in order to make informed decision about the most promising plan taking into account the inherent uncertainty of the problem. Instead of just generating one solution, we present a portfolio of five (qualified) solutions which are maximally diversified. The idea here is that, we are not interested in making decisions for the human decision maker but to offer decision support to aid his judgment. By starting with clear alternative plans together with the risks, the project manager (i.e. decision maker) is offered the needed support to evaluate different trade-off and make decisions as appropriate. In the sequel, we look at possible analysis a project manager could perform in the context of the case study solution plans generated.

On a first assessment, plans 2, 4, and 5 are not so attractive since they have some features postponed, but an analysis of resource consumption of the plans reveals that resource usage of the plans are not well balanced. If there are resource bottlenecks, then the other two plans 1 and 3 would not have been able to schedule all features within available resources. In order to mitigate risk, the distribution must be balanced, but this is not the case for plans 2, 4, and 5. The satisfaction of the stakeholders with the generated plan must be considered in making decisions; the tool support makes it possible to see their preferences through the voting patterns.

Table 5: Structure of the five initial plans generated

	16 sector, 12 carrier BTS	China Feature 1	China Feature 2	Common Feature 01	Common Feature 02	Software Quality Initiative	Expand Memory on BTS Controller	FCC Out-of-Band Emissions	Next Generation BTS	Cost Reduction of Transceiver	Pole Mount Packaging	USEast Inc. Feature 1	USEast Inc. Feature 2
	1	2	3	4	5	6	7	8	9	10	11	12	13
	Features and their scheduled releases												
Plan 1	1	1	1	1	1	2	1	1	2	1	2	2	2
Plan 2	2	1	1	1	1	1	1	1	1	1	2	3	3
Plan 3	2	1	1	1	1	1	1	1	1	1	1	2	2
Plan 4	2	1	1	1	2	1	1	1	3	1	2	1	1
Plan 5	3	1	1	2	1	1	1	1	2	1	2	1	1

Since we cannot make decision for the decision-maker, we will briefly concentrate on analyzing plans 1 and 3 in the context of balancing risk distribution. Plan 3 has 10 of the 13 features (i.e. 77% of total features) scheduled for release 1 and 3 features scheduled for release 2. On the other hand, plan 1 has 8 of the 13 plans scheduled for release 1 (62% of total features), meaning lesser number of features will be delivered first. On a careful analysis, we discovered that it is safe to postpone Feature 9 (Next Generation BTS) because more time is actually needed to understand how to go around mitigating the risk or negotiating the feature with stakeholders because of the high risk level, and uncertainty it might bring to the project plan. The most risky of all the features are 9, 6, 10, 1, and 11 in decreasing order. Plan 3, has scheduled 4 of these 5 highly risky features in release 1 while leaving only one risky feature in release 2. In the case of Plan 1, two high risk features are scheduled for release 1, and three other risky features scheduled for release 2. We cannot categorically state that either of the plans is better. It depends on the problem context and goals of the decision maker in the project. Decisions regarding the plan to adopt for implementation are left for the decision maker.

One interesting result in this case study is not just the assessment of risk made possible but the fact that those assessments are based on measures that consider intricacies of the base system. Several other scenario-playing analyses are possible with ReleasePlanner®, but our focus is to explain the improvement in the current work via the example.

In our previous work [20] using the entire version of this case study example, the project manager merely identified features that are thought to be risky and assigned values to them on a scale of 0 to 10. But using the model-based assessment discussed in this paper, some of those relative assessments have been discovered to be far from reality. For example, feature #6 (Software Quality Initiative) and feature #11 (Pole Mount Packaging) were previously considered to be non-risky features. But identification of impacted components when implementing these features and the health of the components now reveals different risk quantification. These two features constitute some of the very high risk features in the project. A model-based risk assessment that concentrates more on the historical data of the system components forces the designers, programmers or project managers to perform critical analysis before making resource or risk estimations.

6. Conclusion and Future Work

This paper discusses current state of release planning. The shortcomings of the current approaches discussed have necessitated our work in this paper which is targeted at software evolution planning. The risk evaluation measures we have discussed serve as potential improvements to release planning decisions in evolving systems. Our approach is novel in its component level estimations which has not formed the focus of major research. Without component level estimation and characterization, we cannot perform realistic FDIA.

However, the successful implementation and validation of the work is dependent on the availability of historical defect and change data, which are kept by mature organizations like NASA. Our future research will focus on diverse historical system data collection. We will seek to confirm whether the defect data is suitable for determining the health of system components in all environments. It is possible that defects count and recentness of defects may not be enough for evaluating the health of components but it is easier to collect by an organization. In fact, our future research is to ultimately characterize components of a system based on their modifiability to accommodate new features. Modifiability would be a function of attributes like health, complexity, understandability, criticality of components and so on. We are currently investigating the possibilities of developing a generic and more sophisticated taxonomy scheme for classifying modifiability of system components. This would lead to an environment-independent metric definition based on product release history data.

Improvement of release planning model with an objective to incorporate all influencing factors identified (e.g. risk, value, urgency etc.) for measurement-based decision support will be investigated. The subject of integration effort estimation would also constitute part of our future research.

Acknowledgment

The authors would like to thank the Alberta Informatics Circle of Research Excellence (iCORE) for its financial support of this research.

References

[1] Bagnall, A. J., Rayward-Smith, V. J., and Whittley, I. M., "The Next Release Problem," Information and Software Technology, 43 (14), pp. 883-890, 2001.

[2] Bahsoon, R. and Emmerich, W., "ArchOptions: a Real Options-based Model for Predicting the Stability of Software Architectures," In the Proceedings of the 5th Workshop on Economics-Driven Software Research (EDSER-5), Portland, USA, 2003, pp. 35-40.

[3] Borg, A., Karlsson, J., Olsson, S. and Sandahl, K., "Supporting Requirements Selection by Measuring Feature Use", to appear at the 10th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'04), Riga, Latvia, June 7-8, 2004.

[4] Carlshamre, P., "Release Planning in Market-Driven Software Product Development: Provoking an Understanding," Requirements Engineering 7, pp. 139-151, 2002.

[5] Denne, M. and Cleland-Huang, J., "The Incremental Funding Method: Data Driven Software Development," 21(3), pp. 39-47, 2004.

[6] Feather, M. S. and Cornford, S. L., "Quantitative risk-based requirements reasoning", Requirements Engineering, 8(4), 2003, pp. 248-265

[7] Graves, T. L. , Karr, A. F., Marron, J. S. and Siy, H., "Predicting Fault Incidence Using Software Change History," IEEE Transactions on Software Engineering, vol. 26, pp. 653-661, 2000.

[8] Greer, D. and Ruhe, G., "Software Release Planning: An Evolutionary and Iterative Approach," Information and Software Technology, 46(4), pp. 243-253, 2004.

[9] Greer, D., "Decision Support for Planning Software Evolution with Risk Management," Proceedings of the 16th International Conference on Software Engineering and Knowledge Engineering, Banff, Canada, pp. 503-508, 2004.

[10] Jung, H.-W., "Optimizing Value and Cost in Requirements Analysis," IEEE Software, pp. 74-78, 1998.

[11] Karlsson, J. and Ryan, K., "A Cost-Value Approach for Prioritizing Requirements," IEEE Software, 14(5), pp. 67-74, 1997.

[12] Karlsson, J., Wohlin, C., and Regnell, B., "An Evaluation of Methods for Prioritizing Software Requirements," Information and Software Technology, 39(14-15), pp. 939-947, 1998.

[13] Lehman, M. M., "Laws of Software Evolution Revisited," Proceedings of the 5th European Workshop on Software Process Technology (EWSPT'96), Nancy, October 1996, pp. 108-124.

[14] Mens, T. and Demeyer, S., "Future Trends in Software Evolution Metrics," In Proceedings of the 4th International Workshop on Principles of Software Evolution (IWPSE'01), Vienna, Austria, 2001, pp. 83-86.

[15] Mockus, A. and Weiss, D. M., "Predicting risk of software changes," Bell Labs Technical Journal, 5(2): pp. 169 – 180, 2000.

[16] Nejmeh, B. A. and Thomas, I., "Business-Driven Product Planning Using Feature Vectors and Increments", IEEE Software, vol. 9, no. 6, pp. 34 – 42, 2002.

[17] Penny D. A., "An Estimation-Based Management Framework for Enhancive Maintenance in Commercial Software Products," In Proceedings of International Conference on Software Maintenance (ICSM), pp. 122-130, 2002.

[18] Ramil, J. F. and Lehman, M. M., "Metrics of Software Evolution as Effort Predictors - A Case Study," In Proceedings of International Conference on Software Maintenance (ICSM '00), San Jose, California, USA, 2000, pp. 163-172.

[19] Ruhe, G. and Ngo-The, A., "Hybrid Intelligence in Software Release Planning," International Journal of Hybrid Intelligent Systems, Vol. 1(2004), pp. 99-110, 2004.

[20] Ruhe, G. and Saliu, M. O., "The Science and Practice of Software Release Planning," IEEE Software (submitted).

[21] Ruhe, G., "Software Engineering Decision Support – Methodology and Applications," In: Tonfoni and Jain (Eds) Innovations in Decision Support Systems, pp. 143-174, 2003.

[22] Ruhe, G., "Software Release Planning," appears in: "Advances in Software Engineering and Knowledge Engineering'".

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

[24] Sneed, H. M., "A Cost Model for Software Maintenance & Evolution," In Proceedings of the International Conference on Software Maintenance (ICSM'03), Chicago, IL, USA, pp. 264-273.