



**HOW TO ADDRESS MODIFIABILITY
CONCERNS IN THE PROCESS OF
PLANNING FOR THE NEXT RELEASE –
A TRADE-OFF METHOD AND ITS
INITIAL EVALUATION**

Anas Jadallah, Matthias Galster, Mahmood Moussavi,
Guenther Ruhe

Report Number 084/2009

**SOFTWARE ENGINEERING DECISION SUPPORT LABORATORY
UNIVERSITY OF CALGARY**

June, 2009

Table of Contents

Abstract	3
1. Introduction and Motivation	3
2. Related Work	4
2.1 Release Planning for Evolving Systems	4
2.2 Features Dependencies and Modifiability.....	4
2.3 Feature Dependencies and Modeling.....	4
2.4 OO Metrics and Modifiability	5
3. Problem Formalization.....	5
4. Object-Oriented Feature Modeling.....	7
4.1 Features as Classes.....	7
4.2 Interactions between Features in OOFeM	7
5. The NRP-Trade-off Method	8
6. Case Study	12
6.1 Case Study Description: Release Information and Existing System.....	12
6.2 Phase 1	12
6.3 Phase 2	12
7. Discussion of Results.....	15
8. Summary and Future Research.....	16
Appendix	19

ABSTRACT

Planning the next release in the context of software release planning addresses the problem of assigning features to the next release so that technical, resource, risk, and budget constraints are met. However, inevitable changes in stakeholders' needs might require modifications in already implemented features. Accommodating these changes is a non-trivial task as many features have dependencies with other features. Thus, this paper studies the planning for the next release of an evolving system. We introduce a method to adjust baseline release plans for more modifiability by systematically replacing lower value features with features having a higher degree of modifiability and which therefore are expected to increase the overall system modifiability. Our method includes a new approach for feature modeling and assessing the degree of modifiability of systems by applying object-oriented design metrics to the feature domain. As a proof-of-concept, a case study is conducted.

Keywords:

Release Planning, Change Request, Feature Modeling, System Modifiability, Object Oriented Design Metrics, Decision Support.

1. INTRODUCTION AND MOTIVATION

Software systems in market driven product development are subject to continuous modifications, in particular in volatile environments where customers develop their understanding of the system over time. Failing to accommodate these changes can negatively affect the success of the system. A survey over 8000 projects in 350 US companies revealed that one third of the projects were never completed and around 50% succeeded only partially. The major reasons for these failures were changing requirements [36].

A way of addressing this problem other than reactively handling change requests is to analyze the system to be developed proactively and designing it in a way that reduces its modification effort in later stages. However, analyzing the system to be developed requires understanding the features for the system, how they are related to each other and how they impact system modifiability. In our work, a feature is understood as a set of cohesive requirements which provide functionality to the end user of the system [34].

It is well acknowledged in practice and research that features cannot be treated in isolation as there are dependencies between them [10]. Carlshamre et al. have shown that only around 20% of features do not have dependencies with other features [7]. Such dependencies need to be analyzed as they can increase the effort of modifying software systems.

The research presented in this paper is part of an ongoing effort, which addresses changing requirements in software release planning. The first step in this research proposed a method called H2W [18] to handle the problem of “How”, “When”, and “What” of re-planning product releases. The main contributions of this paper are the following:

- A light-weight method called NRP-Trade-off to plan for the next release (NRP), which aims at a trade-off between the value of a release and its impact on modifiability.
- Adaption of the object-oriented (OO) modeling paradigm for the purpose of finding intra- and inter-feature structural properties. These properties are used to predict the impact of new candidate features on modifiability.
- Proof-of concept of the applicability of the method by running a case study.

The paper is structured as follows. Section 2 describes related work. The problem is formalized in Section 3. Section 4 provides a new model for feature representation. This is used to determine trade-offs between feature values and release modifiability in Section 5. Section 6 provides a case study. The validity of the method is studied in Section 7, including a discussion of the case study results. A summary and outlook for future work are provided in Section 8.

2. RELATED WORK

2.1 RELEASE PLANNING FOR EVOLVING SYSTEMS

Software release planning involves decision making on what (new) features or changes to implement in what release. A number of methods have been proposed for in this context (e.g., [2], [19], [29], [35]). Also, it has been argued that change management procedures are needed [23]. However, none of these release planning methods consider the impact of new features on system modifiability.

Saliu et al. developed a release planning framework that considers the effect of existing systems on planning decisions [30]. The framework uses historical defect data to characterize the health of system components in order to decide which features to implement in which release. However, this requires a lot of information which is usually not available at the time of release planning. However, feature characteristics should also be considered at design time when little information about the system is available.

2.2 FEATURES DEPENDENCIES AND MODIFIABILITY

An industrial survey by Carlshamre et al. in the context of release planning found that identifying singular requirements in earlier releases substantially reduces modification effort at a later stage [7]. The survey also related the coupling between requirements in the release planning domain with coupling of objects in OO design: In both cases a high degree of coupling indicates higher sensitivity to changes in other parts of the system. However, the study considered only the coupling between requirements as an indicator for modification effort, whereas release planning assigns features to releases by addressing additional issues (e.g., value, effort, risk). Also, there are other factors besides coupling that impact effort (e.g., cohesion).

Stark et al. [32] explored the relationship between requirements changes and software releases by looking at the volatility of maintenance requirements in a large software system. It has been found that prioritizing maintenance requirements during release planning reduces modification effort for later releases. In order to do proper prioritizations, three new metrics were proposed: planned schedule percentage, degree of requirements volatility, and degree of risk.

2.3 FEATURE DEPENDENCIES AND MODELING

Feature modeling has traditionally been used for domain analysis in software product lines. It captures stakeholder-visible aspects of software products [13]. Such aspects not only refer to the end user but could be of interest for any stakeholder [21]. Feature models hierarchically structure features and define common and variable characteristics in product lines. They also describe relations between features [21], [27], [28].

Fey et al. [13] used inter-feature dependencies to model commonalities and variability in the product line. Similarly, Ferber et al. investigate feature dependencies in product lines to derive variants from legacy product line assets [12]. In both studies, dependencies have not been used in order to assess the degree of modifiability of products.

UML multiplicities in product line feature models have been applied by Riebisch et al. [28]. As multiplicities related to dependencies are less obvious in feature models and sometimes difficult to express, a change in the notation of feature models was suggested, using UML. This lack of semantics in feature models has also been discussed by Bontemps [4].

UML has also been utilized by Griss et al. [15] who use the reuse-driven software engineering business (RSEB), a model-driven approach to software reuse based on OO software engineering. RSEB utilizes UML to model systems in the product line domain. Classes from UML are redefined as features; however, the internal structure of classes is not adopted.

Even though conventional feature models for the product lines model dependencies between features, they do not investigate modifiability based on these dependencies. Also, the internal structure of features is not considered from a modifiability perspective. As opposed to feature models in the product line domain which might focus on any stakeholder; we focus on the end user to represent features. Moreover, variable and common characteristics of software products as modeled in product lines are not required in our feature model for release planning.

2.4 OO METRICS AND MODIFIABILITY

In the context of object oriented metrics and system modifiability, Li et al. collected maintenance effort data from two commercial systems over three years [22]. It was found that the maintenance effort can be predicted from a set of OO metrics. The most important metrics are Depth of Inheritance Tree (DIT), Number of Children (NOC), Message Passing Coupling (MPC), Lack of Cohesion between Methods (LCOM), and Weighted Method Complexity (WMC).

In the context of analyzing the impact of design metrics on a system's modifiability, Deligiannis et al. [11] studied the impact of design heuristics on the quality and maintainability of OO design, (i.e., assuming a specified system design and analyzing its impact). The study showed that OO design heuristics can substantially predict system maintainability: a more maintainable design at an earlier stage can reduce maintenance effort at a later stage. It was also found that coupling between classes is the most significant factor with regard to maintainability, besides cohesion between methods of the class.

More studies on the relation of OO metrics and modifiability can be found in ([5], [6], [8], [20], [24]).

3. PROBLEM FORMALIZATION

Our research is conducted in the context of the next release problem, initially stated by Bagnall et al. [2]. Different from its original formulation, we consider the next release of an evolving system. In addition, we look at multiple criteria to determine which features are recommended for the next release. Special emphasis is set on the impact on modifiability.

The next release planning process is formulated for a time period denoted by $[T1, T2]$. The implementation process of the planned release starts at point in time $t = T1$. The planned release date is $t = T2$. Each feature $f(n)$ is characterized by a set of attributes. Attribute $effort(n)$ describes the (estimated) effort for implementation of feature $f(n)$.

In addition to effort, we consider $risk(n)$ and $value(n)$ that represents an (estimated) risk and an (estimated) projected value of the feature, respectively. To keep the method and the demand for estimation simple, all attributes are defined on a nine-point scale as defined in Table 1. The different values are assumed as being the result of stakeholder (and expert) evaluation as described in [29]. Experts can estimate the level of risk for features from previous project history. For estimation of effort, a variety of known methods and techniques potentially can be applied.

Table 1. Nine-point scale for evaluating effort, risk and value of features

Risk(n), alue(n), normEffort(n)	Interpretation
1	Extremely low
3	Low
5	Average
7	High
9	Extremely high

At the beginning of the re-planning process, we assume a set of existing features $F = \{f(1) \dots f(N)\}$. These features represent the legacy of the evolving system. For the next release, a set of new features, $F_{new} = \{f(N+1) \dots f(M)\}$ is considered. We describe the release plan by a Boolean vector x with

$$x(n) = 1 \text{ iff feature } f(n) \text{ is selected for implementation in the next release } (n = N+1..M) \quad (1)$$

With given total cumulative capacity of CAP, the effort constraint becomes

$$\sum_{n=N+1..M} x(n) \text{ effort}(n) \leq \text{CAP} \quad (2)$$

The added value of assigning the new features to the next release is defined by

$$\text{Value}(x) = \sum_{n=N+1..M} x(n) \text{ value}(n) \quad (3)$$

In our study we are looking for next release plan with two competing criteria:

- Criterion 1: Inclusion of a set of features being best in priority in terms of their balance between value, risk and effort.
- Criterion 2: Inclusion of features, which are attractive in terms of their impact on modifiability of the system.

For the second criterion, we look into intra- and inter-structural properties of features. For that purpose, we perform object-oriented feature modeling described in the next section. The final problem statement therefore becomes:

Perform a trade-off analysis between the added value provided by the new features and their predicted impact on modifiability.

4. OBJECT-ORIENTED FEATURE MODELING

We have developed a new approach for object-oriented feature modeling (OOFeM). OOFeM utilizes concepts known from UML structure diagrams. Structure diagrams in UML model the static structure of a system [14] in contrast to behavioral diagrams (such as use-case diagrams or sequence diagrams) which show the dynamic behavior between objects. In particular, we utilize the concept of class diagrams as one particular type of structure diagram. The motivation behind translating features and their dependencies into a UML class-diagram-like model is to utilize structural capabilities proven from class diagrams, which use metrics such as coupling and cohesion [8]. The use of these metrics for estimating feature modifiability will be discussed later.

4.1 FEATURES AS CLASSES

A class notation in UML consists of three sections: class name, class state or attributes, and class methods. Since a feature is a set of cohesive requirements [38], in the context of OOFeM, the class name is simply mapped to the feature name or feature identifier, as illustrated in Figure 1.

Similarly, we defined requirements in OOFeM as system axioms, i.e., certain properties of the system that must be preserved in order to provide the desired functionality. As such, requirements in OOFeM exist as Boolean states which are either met or not (e.g., requirement “Effort estimation” is considered to be either met or not). In OOFeM, requirements are not expressed as full sentences (like requirements in requirements specifications); only key terms of the actual requirement are included in the feature description.

The methods of a class, which describe the dynamic behavior of classes in UML, are translated into (testable) operationalisms. Operationalisms [39] (as the analogy to methods in OO) are defined as entities that make requirements operational and enable the feature to meet the requirements. Operationalisms are specific to features and cannot occur in other features or spread across features, except in an inheritance hierarchy. Operationalisms always address particular requirements in a feature and ensure that axioms are preserved. For example, the operationalism “Select project data” helps achieve the requirement “Archiving project data”. This operationalism does not occur in other features but only in the feature “Data storage”. These operationalisms are testable to determine if a requirement (or axiom) is met or not.

4.2 INTERACTIONS BETWEEN FEATURES IN OOFeM

OOFeM implements the following types of dependencies:

- requires: $f(1)$ requires $f(2)$ means that $f(1)$ does not function without $f(2)$, i.e., $f(1)$ must be assigned to the same release as $f(1)$ or a previous release. On the other hand, $f(2)$ does not depend on $f(1)$.
- couples: $f(1)$ couples $f(2)$ means that $f(1)$ and $f(2)$ have to be assigned to the same release.
- precedes: $f(1)$ precedes $f(2)$ means that $f(1)$ has to be implemented before $f(2)$.

These dependencies are transferred onto dependencies in the UML class diagrams [25] as specified in Table 2. For the notation of dependencies in UML we refer to [14].

Table 2. Feature dependencies.

Feature dependency	UML artifact
<requires>	Association (arrow)
<ouples>	Association with cardinalities
<precedes>	Inheritance

Figure 1 shows an example of OOFeM. The features are taken from the process of planning for a release planning tool, called ReleasePlanner™ [26]. The model shows all three types of dependencies: *f(8)* requires *f(1)* and *f(3)* because *f(1)* and *f(3)* generate the data which is to be stored. *f(1)* and *f(3)* mutually require each other (stakeholder voting is not useful without planning, and planning or ranking of features cannot be done without voting), and *f(3)* precedes *f(6)* (with regard to time). The model shows a complete representation of features in OOFeM, including their requirements and operationalisms.

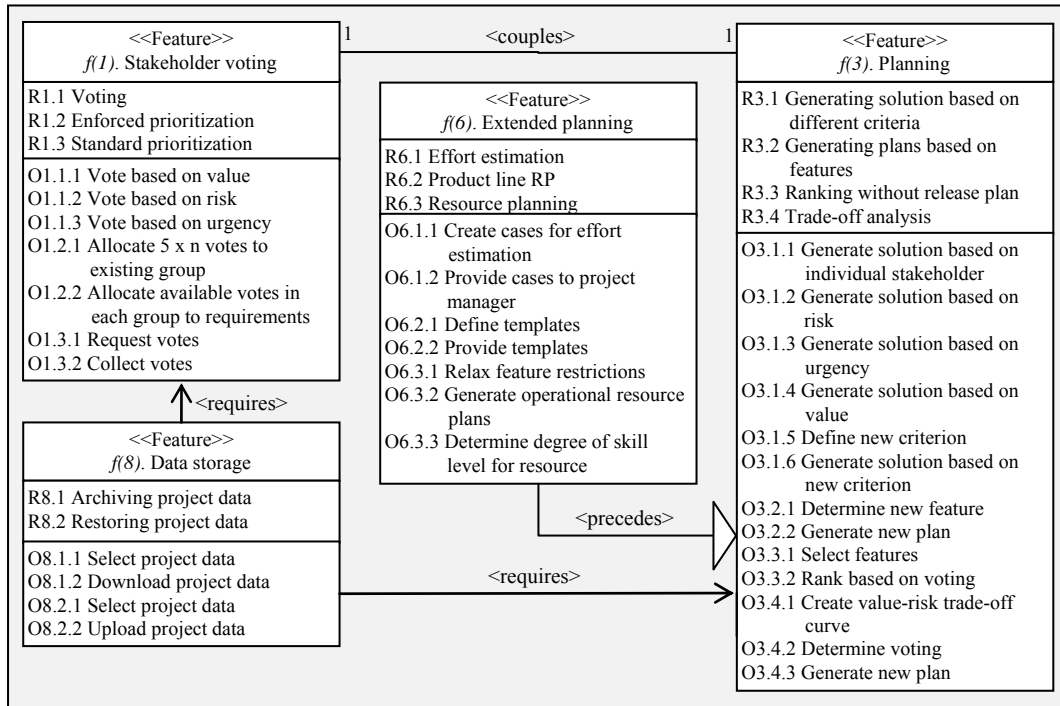


Figure 1. Example of OOFeM

5. THE NRP-TRADE-OFF METHOD

In this section, we present our method called NRP-Trade-off. The method consists of two phases. In Phase 1, initial planning for the next release is done. At this stage, the impact of features on modifiability is ignored. This keeps phase 1 sufficiently simple. For that purpose, the initial three feature attributes are taken into account: Value, effort, and risk (measured on a nine-point ordinal scale). The best plan is created from the features being closest to the ideal point (being the one with highest values and lowest risk and effort). The result of Phase 1 is a plan denoted by *x1*.

The idea of minimization of the distance from ideal point [33] is illustrated in Figure 2 for the case of two criteria (being effort and value in this case). For simplicity again, we treat the ordinal scale like a ratio scale and determine the Euclidean distance to the ideal point. To determine the initial next release plan, greedy optimization [9] is performed. In Figure 2 the ideal point is given by the coordinates of a feature with (value, effort) = (9,1). For features with associated points a, b, c, d, e, and f, the ranking of feature would be b » d » e » f » c » a, where » stands for “closer to ideal point”.

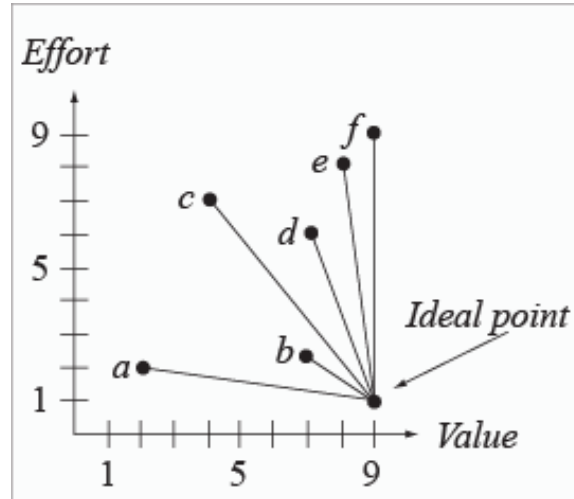


Figure 2. Minimization of distance to ideal point

In Phase 2, the initial plan $x1$ is refined with the objective of becoming more attractive in terms of its impact on modifiability. To achieve this, feature dependencies are analyzed more in-depth by using OO design metrics extracted from the OOFeM. These metrics are utilized to predict modifiability for each feature and later used to analyze how individual features impact system modifiability.

This paper does not define new modifiability metrics, but applies existing ones. In order to better identify the necessary metrics to collect, we used the Goal Question Metric (GQM) approach [3]. GQM helped us elicit the goal of our study and find questions as drivers for finding the relevant metrics. The goal is to improve (purpose) the modifiability (issue) of future releases (object) from the project manager’s viewpoint.

We identified the following questions which characterize how the goal should be attained:

- What is the current modifiability of a feature?
- What is the current modifiability of a release?
- Is the modifiability of a given feature lower or higher compared to other features in the feature set?

Based on these questions, NRP-Trade-off uses the following four OO design metrics adjusted to the feature domain as Feature Oriented Modifiability Metrics (FOMM):

1. Feature Complexity Metric (*FComp*)

FComp indicates how complex a feature is to modify in isolation. Based on Bunge’s definition of complexity of things [37], we determine *FComp* as follows:

$$FComp(n) = \sum_{i=1..TR(n)} effort(r(n,i)) \tag{4}$$

2. Feature Inheritance Metric (*FInh*)

FInh is derived from the precedence relationship. The greater the number of descendants (or preceding features) the greater the effort needed to test the children when a modification happens to a feature. The deeper a tree is in the hierarchy, the greater the number of operationalisms that may be reused, making it more complex to predict their behavior [8], [24]. *FInh* is calculated as follows:

$$FInh(n) = \sum_{i=1..M, i \neq n} isDescendant(f(i), f(n)) \quad (5)$$

3. Feature Coupling Metric (*FCoup*)

FCoup is derived from the <requires> and <couple> relationship between features. Coupling is a special case of requiring (i.e., mutual requiring). The higher the coupling, the higher the sensitivity to changes in other features and therefore maintenance becomes more difficult. Also, the higher the coupling, the more rigorous the testing needs to be, which increases the total modification effort [5], [16], [20]. For calculating *FCoup*, we use the coupling and requiring associations where the feature is either a client or supplier for other features:

$$FCoup(n) = \sum_{i=1..M} isCoupled(f(i), f(n)) \quad (6)$$

where

$$isCoupled(f(i), f(n)) = \begin{cases} 1 & \text{if } f(i) \text{ req/coup } f(n) \\ 1 & \text{if } f(n) \text{ req/coup } f(i) \\ 0 & \text{otherwise} \end{cases}$$

4. Feature Cohesion Metric (*FCoh*)

FCoh is derived from requirements and operationalisms in a feature. *FCoh* is the degree to which operationalisms of the same feature address the same requirements. The more operationalisms which address the same requirement, the more cohesive the feature, which means less effort is needed to maintain its requirements [8], [20]. *FCoh* is calculated as follows:

$$FCoh(n) = \frac{\sum_{i=1}^{TO(n)} \sum_{j=1}^{TR(n)} isAddr(o(n,i), r(n,j))}{TO(n) \times TR(n)} \quad (7)$$

where

$$isAddr(o(n,i), r(n,j)) = \begin{cases} 1 & \text{if } o(n,i) \\ & \text{addresses } r(n,j) \\ 0 & \text{otherwise} \end{cases}$$

Therein, $TO(n)$ is the total number of operationalisms in $f(n)$ and $o(n,i)$ is operationalism $o(i)$ of $f(n)$.

Modifiability is a system property. What matters is how the implementation of one feature impacts the other features. Thus, the aforementioned four metrics cover inter- and intra-feature structural properties which are combined in order to estimate the system (release) modifiability. To ensure comparativeness between the different values, we normalize all metrics. The Euclidean distance to the ideal point (i.e., $Dist(n)$) is calculated again, this time with respect to attributes supposed to be relevant for the features impact on modifiability. The ideal point is given by a feature with extremely low coupling, extremely low inheritance, extremely low complexity, and extremely high cohesion. Thus, $Dist(n)$ is calculated as:

$$Dist(n) = \sqrt{a^2 + b^2 + c^2 + d^2} \quad (8)$$

where

$a = \text{normalized } FCom(n)$

$b = \text{normalized } FInh(n)$

$c = \text{normalized } FCoup(n)$

$d = (1 - FCoh(n))$

Features with a low $Dist(n)$ to the ideal point are considered more attractive in terms of their predicted impact on modifiability. Thus, the modifiability indicator ($Mod(n)$) is defined as

$$Mod(n) = 1 - normDist(n) \quad (9)$$

Therein, $normDist(n)$ is the normalized $Dist(n)$.

In Phase 2 the baseline plan is re-evaluated in consideration of modifiability. The pseudo-code description for conducting the trade-off analysis between (high) value and (high) modifiability is shown in Figure 3.

```
Precondition 1: Set of features  $F_b = \{fb(1) \dots fb(P)\}$  in baseline plan (contains features which might be replaced).  
Precondition 2: Set of features  $F_r = \{fr(1) \dots fr(Q)\}$  originally rejected from baseline plan and now available to replace low modifiable features in baseline plan.  
  
Initialize replacedFeaturesList to empty  
Initialize replacingFeaturesList to empty  
Initialize cumDecreaseInValueList to empty  
Initialize cumIncreaseInModList to empty  
Initialize numberOfChanges to zero  
  
 $F_b.sortInDecreasingOrderBy(\text{value})$   
 $F_r.sortInDecreasingOrderBy(\text{modifiability})$   
  
For  $p = P$  to  $1$ ,  $q = 1$  to  $Q$   
If  $Mod(fr(q)) > Mod(fb(p))$  and  $\text{effort}(fr(q)) \leq \text{effort}(fb(p))$   
     $\text{numberOfChanges} = \text{numberOfChanges} + 1$   
     $\text{replacedFeaturesList.add}(fb(p))$   
     $\text{replacingFeaturesList.add}(fr(q))$   
  
     $\text{cumDecreaseInValueList}[\text{numberOfChanges}] =$   
         $\sum_{n=1.. \text{numberOfChanges}} \text{value}(\text{replacingFeaturesList}[n]) -$   
         $\sum_{n=1.. \text{numberOfChanges}} \text{value}(\text{replacedFeaturesList}[n])$   
  
     $\text{cumIncreaseInModList}[\text{numberOfChanges}] =$   
         $\sum_{n=1.. \text{numberOfChanges}} Mod(\text{replacingFeaturesList}[n]) -$   
         $\sum_{n=1.. \text{numberOfChanges}} Mod(\text{replacedFeaturesList}[n])$   
Endif  
Endfor  
 $\text{cumDecreaseInValueList.normalizeByValues}()$   
 $\text{cumIncreaseInModList.normalizeByMod}()$   
  
Determine  $k = \text{numberOfChanges}$  such that the two points  $(k, \text{cumDecreaseInValueList}(k))$  and  $(k, \text{cumIncreaseInModList}(k))$  are closest to each other.// This represents the best compromise of the number of features to be replaced.
```

Figure 3. Trade-off analysis: pseudo code

6. CASE STUDY

6.1 CASE STUDY DESCRIPTION: RELEASE INFORMATION AND EXISTING SYSTEM

In this section, we present a case study based on real-world data from planning a project for the decision support tool ReleasePlannerTM [26]. ReleasePlanner is an evolving system with a code base written in PHP, Java and C++ in the size of more than 80K SLOC. It has grown over a period of five years. To keep analysis tractable, we consider a subset of the whole system. More precisely, the case study includes:

- A set of 25 existing features, $F = \{f(1), \dots, f(25)\}$ listed in Table 4 (first column) in the Appendix;
- A set of 15 new features, $F_{\text{new}} = \{f(26), \dots, f(40)\}$ being candidate features next product release;
- All candidate features have assigned value, effort, and risk estimated as shown in Table 4 (second column);
- Next release effort capacity, $CAP = 250$;
- Average number of developers, $\text{AvgNDev} = 5$;
- Release start time $T1 = 0$;
- Release end time $T2 = 50$.

6.2 PHASE 1

Phase 1 determines a ranking of the candidates features based on their distance to the ideal point in consideration of effort, risk, and value. The results of this process are documented in Table 4 (second column). At this point, $f(26)$ is amongst the most attractive features for inclusion into the existing system (due to its high value), and $f(38)$ and $f(39)$ represent the least attractive feature (due to their low value).

Greedy optimization is applied to determine the collection of features proposed at the end of Phase 1 for inclusion into the next release. In this process, the effort capacity constraint needs to be considered all the time. This results in plan $x1$ with the following 10 features to be accommodated in the next release (baseline plan): $F_b = \{f(26), f(27) \dots f(35)\}$. Five features were rejected for the next release: $F_r = \{f(36) \dots f(40)\}$. The relative added release value of $x1$ is estimated by the sum of the values of the individual features being $\text{Value}(x1) = 76$.

The proposed plan $x1$ has not been analyzed in terms of its impact on modifiability. It is obviously extremely difficult to make reliable predictions for this question at all. Under the premise of being lightweight, we will perform a trade-off analysis predicting the impact on modifiability from performing object-oriented feature modeling in OOFeM.

6.3 PHASE 2

OOFeM was designed in ArgoUML [1] which allows partial automation of calculating the modifiability as these models can be imported into SDMetrics [31]. This allows us to partially automate the calculation of the four metrics for the features under consideration. For more information on this OOFeM, please refer to [17]. A screenshot of SDMetrics with case study data is shown in Figure 4.

The feature oriented modifiability metrics (FOMM) were used to evaluate features in terms of their estimated impact on modifiability of the system. Table 5 (first column) shows the results of this computation, again measured as the distance to the ideal point.

For this case study, we assume that the system has an initial level of modifiability at time T1 for the existing features. This level of modifiability is not of interest when calculating the change in modifiability based on new features. Similar to the release value the system modifiability is assumed to be calculated by adding up individual features modifiability indicators.

Table 5 (first column) provides a step-by-step description of how the normalized modifiability (normMod(n)) was calculated for each feature. After calculating Dist(n), these distances were normalized (i.e., normDist(n)). Table 5 also shows the ranking of baseline features (which might be replaced) and features initially rejected from the baseline and available to replace low value features in the baseline.

The screenshot shows the SDMetrics software interface. At the top, there is a toolbar with various icons for navigation and analysis. Below the toolbar, there are tabs for 'Metric Data Tables', 'Histograms', 'Kiviat diagrams', and 'Catalog'. The 'Metric Data Tables' tab is active. Below the tabs, there is a 'Select element type' dropdown menu set to 'Class'. Below that, there are 'Sort' and 'Highlight' options. The 'Sort' options are 'No sort' and 'No sort', and the 'Highlight' option is 'nothing'. Below these options is a table with the following data:

Name	Setters	Getters	NOC	NumDesc	NumAnc	DIT	CLD
untitledModel.Flexible Voting Criteria	0	0	0	0	0	0	0
untitledModel.Integrating Ms Excel with data/soln	0	0	1	1	1	1	1
untitledModel.Ms Project Compatability	0	0	0	0	2	2	0
untitledModel.Stakeholders voting analysis	0	0	0	0	5	4	0
untitledModel.Re-planning Capabilities	0	0	0	0	2	2	0
untitledModel.Product Line Release Planning	0	0	0	0	0	0	0

Figure 4. Screenshot of SDMetrics with case study data

Figure 5 illustrates the exchange of features. Initially, features f(35) and f(36) were exchanged. Figure 6 shows that this decreases the estimated (normalized) feature value, but increases the (normalized) feature modifiability. The same is true for all subsequent iterations of feature replacements. All iterations are summarized in Table 3. E.g., f(35) is replaced by f(36) resulting in .43 increase in modifiability (.62-.19) and decrease in value by 1 (5-4).

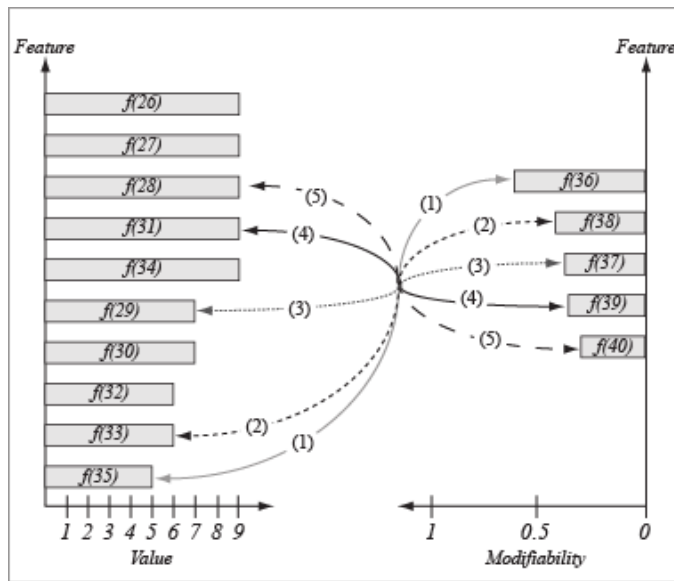


Figure 5. Exchange of features towards modifiability

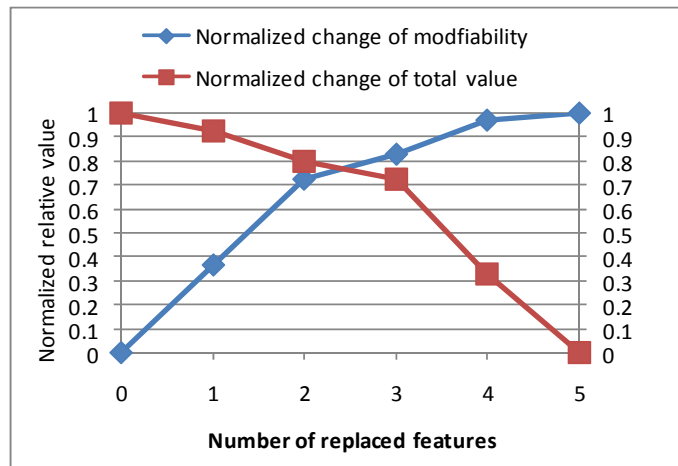


Figure 6. Trade-off analysis between added modifiability and decrease of plan value

The trade-off curve does not answer the question which exchange option should be taken. Assuming that the intersection of the two normalized curves represents the best compromise, it was recommended that features $f(35)$ and $f(33)$ are replaced by features $f(36)$, and $f(38)$.

Table 3. Evolution of features replacement

Number of features eliminated from baseline	Set of features to be replaced	Set of replacing features	Added normalized modifiability	Decrease in Value
1	{35}	{36}	0.43	1
2	{35, 33}	{36, 38}	0.43+ 0.43	1+2
3	{35, 33, 29}	{36, 38, 37}	0.43+0.43+0.09	1+2+1
4	{35, 33, 29, 31}	{36, 38, 37, 39}	0.43+0.43+0.09+0.19	1+2+1+6

Number of features eliminated from baseline	Set of features to be replaced	Set of replacing features	Added normalized modifiability	Decrease in Value
5	{35, 33, 29, 31, 28}	{36, 38, 37, 39, 40}	0.43+ 0.43+0.09+0.19+0.03	1+2+1+6+5

7. DISCUSSION OF RESULTS

From running the case study, we have made a number of observations:

- Considering modifiability when planning for the next release is important. Otherwise, only end-user concerns in terms of the release value are taken into consideration. If the modifiability of the release is totally ignored, the maintenance effort would potentially increase for later releases. Here, modifiability is not only of interest for the developer or project manager, but also for the customer: Higher modifiability leads to lower maintenance costs and faster delivery of new functionality. This becomes even more relevant when taking into consideration that many features requested by the customer and implemented in the system are actually never used.
- The method allows easy and quick planning for the next release. In the case study we planned with 40 features. Easy and quick planning would still be guaranteed for at least 40 + 20 features.
- The efforts for data collection for determining the final plan for the next release are affordable.
- The light-weight-nature of this procedure is intentional. Associated with that, different heuristics were used in the process. The results do not claim to (and in fact, can not) be 100% reliable. The evaluation of the quality of the results is a topic of future research.

We do not claim external validity for the observations above. Substantially more empirical evaluation would be needed for that. There are also limitations with regard to the internal validity of the results. This is caused by simplifications which were made in the underlying model:

Firstly, we aggregate all metrics into one value for the modifiability indicator of a feature and eventually the modifiability of a release. However, in reality, different metrics might have different degrees of impact on the modifiability. This would depend on the individual feature and the system under study. Moreover, modifiability of the feature itself depends on the type of change. One change might be easier to accommodate than another change. Thus, question-based evaluation techniques for assessing the modifiability of features or the system could supplement the metrics in order to judge the modifiability of a feature in a particular change scenario. Also, the selected metrics aggregated into feature modifiability indicator might not be exhaustive.

Secondly, we assume that feature modifiability has an additive impact on system modifiability. In order to determine the modifiability of the system, we sum up individual feature modifiability indicators. However, the actual effort of change implementation not only depends on the feature itself, but also on other features involved.

Thirdly, we assume that all features in the existing system are completed and no feature is in progress. For planning the next release, a subset of the features in the existing system might be in progress. This is a simplification which allows planning in a reasonable amount of time and computational effort and also replacing features in an easy manner.

Fourthly, it might not be sufficient to describe feature dependencies based on requires, couples and precedes relationships. We do not model dependencies such as “either or”, “at least one”, or “at most one” as suggested in the context of software product lines. However, these dependencies would make the application of OO design metrics difficult and probably would require the definition of new metrics.

Finally, to keep the planning method light-weight, we applied a greedy optimization to deliver good, but not necessarily optimal results.

8. SUMMARY AND FUTURE RESEARCH

Software release planning has significant impact on the success or failure of software product development. A systematic approach is likely to generate plans achieving higher business value. On the other hand, modifiability of releases determines the costs of future changes and thus the costs of future releases. Both problems are very important, but inherently difficult. This motivates our method NRP Trade-off.

The main contribution of this research is the provision of a methodology that helps address the next release problem for planning the next release under the consideration of system modifiability (a factor that has not been considered before when planning for the next release). The emphasis is on an easy-to-use procedures to incorporate modifiability concerns into the planning process. We model features and dependencies of newly added features in the OOFeM to compute metrics, which have proven to be predictors for release modifiability.

NRP-Trade-off is relying on a number of assumptions which need to be verified before its application. The meaningfulness of the results will increase with the conformance to the stated assumptions. Nonetheless, the method has initially proven useful. It is not only applicable for planning, but also for more frequent re-planning.

There are two fundamental options to improve quality of results. The first one (related to Phase 1) is to generate better baseline plans. This can be done, for example, by applying rigorous planning and optimization techniques as described in [29]. A second option is to stronger invest into prediction of modifiability. In both cases, the potential for better results concurs with additional effort, both for data collection and for running the process.

A more comprehensive empirical validation is one of the topics for future research. Also, an extension of the feature model towards an input for testing will be investigated. Requirements as Boolean states within features allow determining how well a feature has been met and what its implementation status is. This status can be used to identify features which are in progress and not finished - one of the limitations of our work discussed in Section 7. Moreover, the impact of increasing modifiability on non-functional properties, such as performance, has not been investigated yet. We plan to perform studies in which we relate modifiability also in consideration of non-functional properties.

Acknowledgement

Part of the work presented was financially supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada under Discovery Grant no. 250343-07. The authors would also like to thank Jim McElroy and Kornelia Streb for editorial input on the paper.

Reference

- [1] ArgoUML tool, <http://argouml.tigris.org>.
- [2] Bagnall, A. J., Rayward-Smith, V. J. and Whitley, I. M., “The Next Release Problem”, *Information and Software Technology*, vol. 43, 2001, pp. 883-890.
- [3] Basili, V., Caldiera, G., Rombach, D., “The Goal Question Metric Approach”, in *Encyclopedia of Software Engineering*, Marciniak, J. J. (Ed.), New York, NJ: John Wiley & Sons, pp. 528-532, 1994.
- [4] Bontemps, Y., Heymans, P., Schobbens, P., Trigaux, J., “Semantics of FODA Feature Diagrams”, Proceedings of the Workshop on Software Variability Management for Product Derivation, Boston, MA, 2004, pp. 48-58.
- [5] Briand, L., Daly, J., Wuest, J., “A Unified Framework for Coupling Measurement in Object Oriented Systems”, *IEEE Transactions on Software Engineering*, vol. 25, 1999, pp. 91-121.
- [6] Brito, F., “The MOOD Metrics Set”, Proceedings of. ECOOP’95 Workshop on Metrics, Aarhus, Denmark, 1995.
- [7] Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., and Natt, J., “An Industrial Survey of Requirements Interdependencies in Software Product Release Planning”, Proceedings of the 5th International Symposium on Requirements Engineering, 2001, Toronto, Canada, pp. 84-91.
- [8] Chidamber, S. R. and Kramer C. F., “A Metrics Suite for Object Oriented Design”, *IEEE Transactions on Software Engineering*, vol. 20, 1994, pp. 476-493.
- [9] Cormen, T. H., Leiserson, C. E. and Rivest, R. L., *Introduction to Algorithms*, Clifford Stein. The MIT Press, Cambridge, MA, 2007.
- [10] Dahlstedt, A. and Persson, A., “Requirements Interdependencies: State of the Art and Future Challenges”, in *Engineering and Managing Software Requirements*, Aurum, A., Wohlin C. (Eds.), Berlin: Springer, pp. 95-116, 2006.
- [11] Deligiannis, I., Shepperd, M., Roumeliotis, M., Stamelos, I., “An Empirical Investigation of an Object-Oriented Design Heuristic for Maintainability”, *Journal of Systems and Software*, vol. 65, 2003, pp. 127-139.
- [12] Ferber, S., Haag J., Savolainen, J., “Feature Interaction and Dependencies: Modeling Features for Reengineering a Legacy Product Line”, Proceedings of the 2nd International Conference on Software Product Lines, San Diego, CA, 2002, pp. 235-256.
- [13] Fey, D, Fajta R., Boros, A., “Feature Modeling: A Meta-model to Enhance Usability and Usefulness”, Proceedings of the 2nd International Conference on Software Product Lines, San Diego, CA, 2002, pp. 198-216.
- [14] Fowler, M. and Scott, K., *UML Distilled: Applying the Standard Object Modeling Language*, Addison-Wesley, Boston, MA, 1997.
- [15] Griss, M., Favaro, J., D’Alessandro, M., “Integrating Feature Modeling with the RSEB”, Proceedings of the International Conference on Software Reuse, Victoria, BC, 1998, pp. 76-85.
- [16] Harrison, R., Counsell, S., Nithi, R., “An Evaluation of the MOOD Set of Object-Oriented Software Metrics”, *IEEE Transactions on Software Engineering*, vol. 24, 1998, pp. 491-496.
- [17] Jadallah, A., <http://www.ucalgary.ca/~agjadall/ICSM2009>.
- [18] Jadallah A., Al-Emran A., Moussavi M., Ruhe G., “The How? When? and What? For the Process of Re-Planning for Product Releases”, Proceedings of the International Conference of Software Process, Vancouver, BC, 2009, in press.
- [19] Jung, H. W., “Optimizing Value and Cost in Requirements Analysis”, *IEEE Software*, vol. 15, 1998, pp. 74-78.
- [20] Lee, Y., Chang, K., “Reusability and maintainability metrics for object-oriented software”, Proceedings of the 38th Annual Southeast Regional Conference, Clemson, South Carolina, 2000, pp. 88-94.

- [21] Lee, K., Kang, K. C., Lee, J., "Concepts and Guidelines of Feature Modeling for Product Line Software Engineering", Proceedings of the 7th International Conference on Software Reuse, Austin, TX, 2002, pp. 62-77.
- [22] Li, W., Henry, S., "Object-oriented Metrics that Predict Maintainability", *Journal of Systems and Software*, 1993, pp. 111-122.
- [23] Loconsole, A., "Empirical Studies on Requirement Management Measures", in Proceedings of the 26th International Conference on Software Engineering, Edinburgh, UK, 2004, pp. 42-44.
- [24] Lorenz, M., Jeff, K., *Object Oriented Software Metrics*, Prentice Hall Publishing, Englewood Cliffs, NJ, 1994.
- [25] OMG, "OMG Unified Modeling Language (OMG UML), Superstructure," Version 2.2, February 2009.
- [26] ReleasePlanner, Expert Decisions Inc., www.releaseplanner.com.
- [27] Riebisch, M., "Towards a More Precise Definition of Feature Models", Proceedings of the Workshop on Modelling Variability for Object-Oriented Product Lines", Darmstadt, 2003, pp. 64-76.
- [28] Riebisch, M., Boellert, K., Streitferdt, D. Philippow, I., "Extending Feature Diagrams with UML Multiplicities", Proceedings of the 6th Conference on Integrated Design and Process Technology", Pasadena, CA, 2002, pp. 1-7.
- [29] Ruhe, G. & Saliu, M. O., "The Art and Science of Software Release Planning", *IEEE Software*, vol. 22 2005, pp. 47-53.
- [30] Saliu, O., Ruhe G., "Supporting Software Release Planning Decisions for Evolving Systems", Proceedings of the 29th Annual IEEE/NASA Software Engineering Workshop, Greenbelt, MD, 2005, pp. 14-26.
- [31] SDMetrics, The Software Design Metrics tool for the UML, <http://www.sdmetrics.com/>.
- [32] Stark, G., Skillicorn, A., Ameele, R., "An Examination of the Effects of Requirements Changes on Software Maintenance Releases", *Journal of Software Maintenance: Research and Practice*, 1999, pp. 293-309.
- [33] Steuer, R.E., *Multiple Criteria Optimization: Theory, Computation, and Application*, John Wiley & Sons, New York, NJ, 1986.
- [34] Turner, C., Fuggetta, A., Lavazza, L., Wolf, A., "A Conceptual Basis for Feature Engineering", *Journal of Systems and Software*, vol. 49, 1999, pp. 3-15.
- [35] van den Akker, M., Brinkkemper, S., Diepen, G. and Versendaal, J., "Software Product Release Planning through Optimization and What-If Analysis", *Information and Software Technology*, vol. 50, 2008, pp. 101-111.
- [36] van Lamsweerde, A., "Requirements Engineering in the Year 00: a Research Perspective", Proceedings of the 22nd International Conference on Software Engineering (ICSE' 2000), Limerick, Ireland, 2000, pp. 5-19.
- [37] Wand, Y., Weber, R., "An Ontological Evaluation of Systems Analysis and Design Methods," in *Information Systems Concepts: An In-depth Analysis*, Falkenberg, E. D., Lindgreen, P. (Eds.), Amsterdam: Elsevier Science, 1989.
- [38] Zhang, W., Mei, H., Zhao, H., "A Feature-Oriented Approach to Modeling Requirements Dependencies," Proceedings of the 13th IEEE International Conference on Requirements Engineering, Paris, France, 2005, pp. 273-284.
- [39] Zhang, W., Mei H., Zhao H., "Feature-driven Requirements Dependency Analysis and High-level Software Design," *Requirements Engineering*, vol. 11, 2006, pp. 205-220.

APPENDIX

Table 4. Existing and new system features

Existing System Features		New Features							
<i>f(n)</i>	Feature Name	<i>f(n)</i>	Feature Name	Value (n)	Effort (n)	normEfort (n)	Risk (n)	Dist (n)	normDist (n)
<i>f1</i>	Flexible Voting Criteria	<i>f26</i>	Professional UI Development	9	10	1	1	0	0
<i>f2</i>	Stakeholders' Requirements Requests	<i>f27</i>	Visualization of Output	9	12	2	3	2.24	0.2
<i>f3</i>	Stakeholders' Resource Estimates	<i>f28</i>	User Planning Dashboard	9	15	2	5	4.12	0.37
<i>f4</i>	Multiple Stakeholders' Weight	<i>f29</i>	Extended Reporting	7	8	1	4	3.61	0.32
<i>f5</i>	Generating Criteria-based Solutions	<i>f30</i>	Documentation	7	38	6	2	5.48	0.49
<i>f6</i>	Comparative Analysis of Solutions	<i>f31</i>	Conetxt Sensitive Help	9	9	1	6	5	0.45
<i>f7</i>	Comparative Analysis of Solutions	<i>f32</i>	OnDemand Help	6	29	5	4	5.83	0.52
<i>f8</i>	Improved Stakeholders' Conformance	<i>f33</i>	Stakeholders Notification	6	45	8	2	7.68	0.69
<i>f9</i>	Competitor Stakeholders Voting	<i>f34</i>	Improve web hits	9	50	9	1	8	0.71
<i>f10</i>	Stakeholders' Voting Analysis	<i>f35</i>	Stakeholders Feedback	5	17	3	6	6.71	0.6
<i>f11</i>	Explanation Component	<i>f36</i>	Splitting Features over Releases	4	20	3	7	8.06	0.72
<i>f12</i>	Finetune Optimization Algorithm	<i>f37</i>	Multiple windows accessible	6	41	7	6	8.37	0.75
<i>f13</i>	DOORS Compatability	<i>f38</i>	Ms Excel Integration	3	34	6	9	11.18	1
<i>f14</i>	ROI Analysis of RP trial account	<i>f39</i>	MS excel Compatibility	3	30	5	6	8.77	0.78
<i>f15</i>	Attracting more testimonials	<i>f40</i>	Ms Project Compatability	4	30	5	6	8.12	0.73
<i>f16</i>	Records of Solutions Hisotry								
<i>f17</i>	RP Training Tutorials								
<i>f18</i>	Conformance measure for requirements								
<i>f19</i>	Multi-project Planning								
<i>f20</i>	Enclusive Effort Estimation Support								
<i>f21</i>	Re-planning Capabilities								
<i>f22</i>	Product line release planning								
<i>f23</i>	Elimination of open source code								
<i>f24</i>	Caching mechanisms								
<i>f25</i>	Scalability and Failover recovery								

Table 5. Baseline features modifiability calculation and ranking for trade-off analysis

Ranking and Calculating Features Modifiability										Ranking New Features for Trade-off Analysis					
<i>f(n)</i>	FCoup (n)	normFCoup (n)	FInh(n)	normFInh (n)	normFCComp (n)	normFCoh (n)	Dist(n)	normDist (n)	normMod (n)	Baseline Features (To be Replaced)		Replacing Features			
										<i>f(n)</i>	value (n)	normMod (n)	<i>f(n)</i>	value (n)	normMod (n)
<i>f36</i>	0	0	0	0	0.4	0.77	0.46	0.38	0.62	<i>f26</i>	9	0.17	<i>f36</i>	4	0.62
<i>f34</i>	0	0	0	0	0.34	0.68	0.47	0.38	0.62	<i>f27</i>	9	0.49	<i>f38</i>	3	0.43
<i>f30</i>	1	0.25	2	0.4	0.18	0.67	0.6	0.49	0.51	<i>f28</i>	9	0.27	<i>f37</i>	6	0.38
<i>f27</i>	1	0.25	1	0.2	0.24	0.52	0.62	0.51	0.49	<i>f31</i>	9	0.16	<i>f39</i>	3	0.35
<i>f32</i>	0	0	0	0	0.58	0.65	0.68	0.55	0.45	<i>f34</i>	9	0.62	<i>f40</i>	4	0.3
<i>f38</i>	0	0	0	0	0.6	0.65	0.7	0.57	0.43	<i>f29</i>	7	0.29			
<i>f37</i>	0	0	0	0	0.68	0.66	0.76	0.62	0.38	<i>f30</i>	7	0.51			
<i>f39</i>	1	0.25	1	0.2	0.6	0.58	0.8	0.65	0.35	<i>f32</i>	6	0.45			
<i>f40</i>	1	0.25	0	0	0.82	0.94	0.86	0.7	0.3	<i>f33</i>	6	0			
<i>f29</i>	1	0.25	0	0	0.28	0.21	0.87	0.71	0.29	<i>f35</i>	5	0.19			
<i>f28</i>	0	0	0	0	0.16	0.11	0.9	0.73	0.27						
<i>f35</i>	1	0.25	0	0	0.9	0.66	0.99	0.81	0.19						
<i>f26</i>	0	0	5	1	0.2	1	1.02	0.83	0.17						
<i>f31</i>	0	0	0	0	0.74	0.29	1.03	0.84	0.16						
<i>f33</i>	1	0.25	0	0	1	0.33	1.23	1	0						