Chapter 34
A Feature-Driven Method for Adaptive and Perfective Maintenance

Karam Ignaim and João M. Fernandes

Abstract This manuscript discusses whether the refinement of feature models (FMs) can be applied to fulfill the completion of change requests and satisfy requirement modifications through the Software Maintenance Life Cycle (SMLC). It presents a method that uses FMs as an evolvable artifact to handle software changes as part of adaptive and perfective maintenance. When maintaining a product, we should focus on preserving its features and enhancing them. Our method uses FM composition that guarantees structure-preserving refinements to the underlying FM until the base model is fully updated with the necessary change requests. The empirical research reveals that, on average, around 91% of requested product changes are effectively implemented using our method. It also indicates that the feature-based method significantly improves the efficiency and success rate of implementing change requests.

34.1 Introduction

Software maintenance involves the process of modifying a software system after delivery to improve its performance, to fix any new defects, or to adapt the product to the changed environment [7]. The changes to the software may address coding errors or to correct design faults, or substantial additions to correct specification problems or accommodate new requirements [9].

Software maintenance tasks fall into four categories: corrective, preventive, adaptive, and perfective [4]. This work focuses on the last two categories. **Adaptive maintenance** refers to modifications to a software product undertaken after delivery to make it usable in an altered or changing environment, whereas **perfective maintenance** is a modification to a software product made after delivery to improve performance or maintainability. In most situations, changes occur often; therefore,

K. Ignaim (✉)
Department Software Engineering, Al-Balqa Applied University (BAU), Salt, Jordan
e-mail: karam.ignaim@bau.edu.jo

J. M. Fernandes
Centro ALGORITMI/Department Informatics, Universidade do Minho, Braga, Portugal
e-mail: jmf@di.uminho.pt

© The Author(s), under exclusive license to Springer Nature Singapore Pte Ltd. 2025

C. S. In et al. (eds.), *Information Systems for Intelligent Systems*, Smart Innovation, Systems and Technologies 431, https://doi.org/10.1007/978-981-96-1210-9_34

629635_1_En_34_Chapter ☑ TYPESET ☐ DISK ☐ LE ☑ CP Disp.:7/1/2025 Pages: 11 Layout: T1-Standard

UNCORRECTED PROOF

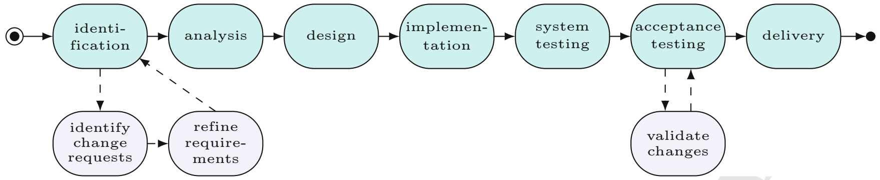

Fig. 34.1 Steps of the proposed method involved in the SMLC

software that does not receive regular adaptive and preventive maintenance quickly becomes outdated. Existing applications updated with new technology may exhibit small performance enhancements in areas such as scalability and speed, but their overall functionality is typically unaffected. Perfective maintenance improves the software. It encompasses the enhancement of existing features, in addition to the incorporation of new ones. Feature models (FMs) are widely used for software product lines (SPLs). An FM identifies the features of an SPL and the possible constraints among them. FMs can express the features that describe a potential product in a compact and comprehensible format. An FM can be updated by adding, changing, or removing features [6] in a consistent manner using FM composition [1].

The goal of this manuscript is to describe a method that uses FMs as a central artifact to manage these changes effectively. The method aims to ensure that software features are preserved and enhanced while accommodating change requests. This research seeks to determine whether the refinement of FMs can be utilized to facilitate the completion of change requests in an efficient manner. The method has been evaluated in a real-world setting by conducting a case study that handles change requests for adaptive and perfective maintenance. The findings suggest that incorporating feature modeling into the software maintenance process can lead to more successful outcomes and ultimately contribute to the overall success of software projects.

The SMLC method consists of seven phases that can all be used in an iterative manner and can be gradually expanded so that customized items and processes can be incorporated (see Fig. 34.1). The cycle is a framework that specifies the phases performed during the maintenance process [8].

34.2 The Method

This section outlines the method proposed in this paper, leveraging FMs for software adaptive and perfective maintenance. Upon receiving a change request, the method endeavors to build the FM that accurately represents the intended change. Our method adopts and adapts one of the compositional mechanisms proposed by Acher [1]. The method uses the insertion operator, to capture the change requests, which are part of the SMLC. The method consists of three steps. The first two steps intertwined with

UNCORRECTED PROOF

629635_1_En_34_Chapter ✓ TYPESET DISK LE ✓ CP Disp.:7/1/2025 Pages: 11 Layout: T1-Standard

the problem identification phase of SMLC, while the third step intertwined with the acceptance test phase of SMLC (see Fig. 34.1).

Figure 34.1 presents an overview of the steps of our method. The upper side of the figure indicates the phases of the SMLC. The lower side of the figure indicates the steps of our methods and specifies the related phase in which the steps take place. The remaining phases of the SMLC work as usually defined. We next outline our method. In **step 1** (inserted into the **problem identification phase**), we identify the change request by collecting data from the change request form. Change request forms play a crucial role in the change control process by providing a standardized way to document, review, and manage alterations to a product. They help ensure that changes are carefully considered, properly evaluated, and implemented in a controlled manner [3]. The collected data is saved as a list of modifications and features. Based on the list, we refine the requirements by evolving the FM using the composition operator in **step 2** (inserted into the **problem identification phase**). At the end of the process, the specified modifications undergo a final validation in **step 3** (inserted into the **acceptance test phase**). To generate acceptance test report, the base FM is checked to ensure that all software features conform to the specified requirements on the change request form. We assume that the maintainers who want to update the base FM (designed during the development of the software product) know the insertion point in the base FM, can design the sub-FM that contains the desired updates in terms of features, and can create an FM that supports all of these updates.

In the following, we give details on each of the three steps.

Step 1. Identify the change request. The requests for a change of the software product are identified by collecting data from the change request form as a *feature list*. The goal is to construct an initial list of features that represent a change request. The maintainers read specific items from the change request form (column 1 of Table 34.1) and then map them into a sub-FM as depicted in Fig. 34.2. Each row represents a mapping between an item of the form and a feature of the sub-FM. For example, in Table 34.1, Change Request Name is mapped to the Root feature, and ID is mapped to the Parent feature. This figure represents a change request with a sub-FM.

Table 34.1 Items of the change request form mapped to the FM

Change request name	Root feature
ID	Child feature
Priority	Child feature
Submitter name	Child feature
Change request type	Child feature

UNCORRECTED PROOF

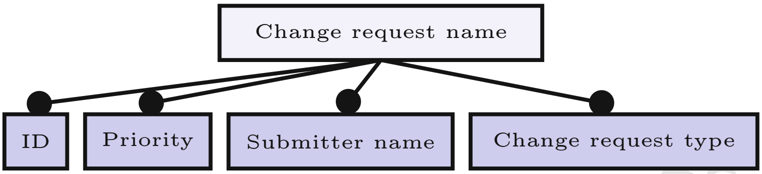

Fig. 34.2 An abstract view of a sub-FM that is designed for a single change request

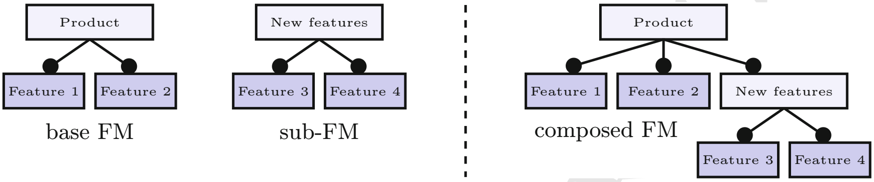

Fig. 34.3 Example of the insert operator

Step 2. Refine the requirements by evolving the FM. The aim is to refine the base FM with the information of a change request designed as a sub-FM in step 1. The maintainers can use the insert operator to refine the base FM. The purpose of the insert operator is to introduce the sub-FM into the base model. In the example shown in Fig. 34.3, a maintainer can extend the base FM associated a software product with the features represented in a sub-FM, upon receiving a change request from a customer. The sub-FM is inserted with mandatory status as a child feature of the base FM. The result is the composed FM, which is a combination of the base FM and the sub-FM.

The Insert Operator. The insertion operator adopted by our method is inspired by the work of Acher et al.[1]. The operator supports a way to insert a sub-FM (i.e., a change request) into the base FM (i.e., a given software product). Syntactically, the insert operator is defined as follows:

```
Insert (root of the sub-FM: Feature, insertion feature point: Feature,
                          operator: Operator)
```

The insert operator takes three arguments. The first one is the feature to be inserted, which is a root feature in the sub-FM. The second one is the targeted feature, which is a feature in the base model where the insertion must occur. Finally, the operator is provided by the maintainers (e.g., and- operator). An empty intersection between the set of features in the base FM and the set of features in the sub-FM is a prerequisite for the insert operator. This condition maintains the well-formed quality of the combined FM, which specifies that each feature has a unique name. To compose models, the insert operator permits the specification of any appropriate FM (i.e., and-, xor-, or or-groups).

UNCORRECTED PROOF

Step 3. Validate the specified modifications. The composed FM is used to perform the acceptance testing on the fully integrated software product. The change reviewer can use the composed FM to validate that all features of the software product conform to the specifications of the change request form. For example, the change reviewer can use the composed Math Kids Game FM (Fig. 34.3b) to check that different items in a change request meet the defined requirements and have been successfully implemented in the software product.

Once the FM has been updated, our approach ensures that previous configuration of software product is still valid once the FM has been updated. This is due to the fact that our approach adopts two main operators to compose FMs [1]. Each operator is described by stating where it is applied, what features are composed, and how the composition is made. Each composition is defined by rules that formally describe the structure of the resulting FM and preserve the set of configurations determined by the input FMs.

34.3 Evaluation: The Case Study

To conduct the case study, the Goal-Question-Metric approach was used [2]. The main **goal** of this case study was to evaluate the effectiveness, efficiency, and stakeholder satisfaction of the proposed method to handle software product change in adaptive and perfective maintenance applied to the Sales Distribution Application (SDA), including FM refinement.

The case study was conducted to address three **Research Questions**:

Q1: How effective is the method for addressing software product change requests in adaptive and perfective maintenance?

Q2: How efficient is the method for software product change requests in adaptive and perfective maintenance?

Q3: How satisfied are the stakeholders with the changes implemented using the method?

These questions provide an understanding of the effectiveness, efficiency, and stakeholder satisfaction of the method for addressing software product change requests in adaptive and perfective maintenance. The efficiency of the method sheds light on the time and resources required for implementing software product changes, allowing for better resource allocation and planning. Stakeholder satisfaction indicates the level of acceptance and approval of the changes made using the proposed method, determining its overall success and impact on the organization.

The **subjects** (i.e., participants) of the study are stakeholders of the SDA, including software maintainers and customers. The participants of the case study were six software maintainers and six customers of SDA.

To evaluate the method, we used the following **criteria**: **(1) Effectiveness**, to assess the ability of the feature-based method to successfully address software product change requests in adaptive and perfective maintenance. It can be measured using metrics such as the percentage of successfully addressed change requests and stakeholder satisfaction scores. **(2) Efficiency** to evaluate the efficiency of the method in

UNCORRECTED PROOF

terms of time and effort required for software product change in adaptive and perfective maintenance. Metrics such as the average time taken to implement changes and a comparison of the effort required can be used to assess efficiency. **(3) Stakeholder Satisfaction** to measure the satisfaction of stakeholders, including customers and software maintainers, with the changes implemented using the feature-based method. It can be measured through usability satisfaction questionnaire, like the IBM Usability Questionnaire for computer systems [5]. To adapt this questionnaire to our method, we rephrased the questions by replacing the word 'system' with 'method':

1. Overall, I am satisfied with how easy it is to use this method.
2. It was simple to use this method.
3. I could effectively complete the tasks and scenarios using the method.
4. I was able to complete the tasks and scenarios quickly using the method.

The perceptions of the respondents were given on a 7-point Likert scale, from 1 (strongly agree) to 7 (strongly disagree).

Based on the selected criteria, the following hypotheses can be formulated:

- Null hypothesis (Hp_0): The feature-based method does not provide a significant improvement in the performance (i.e., effectiveness and efficiency) of software maintainers in addressing change requests.
- Alternative hypothesis (Hp_1): The feature-based method provides a significant improvement in the performance (i.e., effectiveness and efficiency) of software maintainers in addressing change requests.
- Null hypothesis (Hs_0): There is no significant improvement in stakeholder satisfaction with the feature-based method.
- Alternative hypothesis (Hs_1): There is a significant improvement in stakeholder satisfaction with the feature-based method.

We wish to refute the null hypotheses with the greatest possible significance and prove the alternative hypotheses.

Additionally, the following dependent variables were considered:

- **Effectiveness** represents the percentage of change requests that were effectively implemented using the proposed method.
- **Efficiency** represents the percentage of change requests that were efficiently implemented using the proposed method.
- **Stakeholder Satisfaction** measures the level of satisfaction expressed by stakeholders with the changes implemented using the method.

These dependent variables capture the key aspects in the **effectiveness**, **efficiency** and **stakeholder satisfaction** of the proposed method for software product change in adaptive and perfective maintenance. By analyzing and measuring these variables, the case study helps in evaluating the impact and performance of the method in achieving successful change implementation and stakeholders satisfaction.

UNCORRECTED PROOF

629635_1_En_34_Chapter ✓ TYPESET DISK LE ✓ CP Disp.:7/1/2025 Pages: 11 Layout: T1-Standard

Table 34.2 Change requests for the sales distribution application during the adaptive and perfective maintenance

CR1	Incorporate mechanisms for gathering consumer feedback, ratings, and reviews into the application (customer feedback and reviews)
CR2	Develop a mobile application or optimize the existing one for mobile devices, allowing sales representatives to use the application on smartphones and tablets while traveling (mobile accessibility)
CR3	Expand the reporting and analytics capabilities to gain a deeper understanding of sales performance, product performance, customer behavior, and market trends (improved reporting and analytics)
CR4	Integrate sales forecasting and demand planning capabilities into the application to assist businesses in making accurate sales projections, optimizing inventory levels, and planning production or procurement accordingly (sales forecasting and demand planning)
CR5	Provide a self-service portal for customers, allowing them to view product catalogs, place orders, track order status, and access their purchase history independently (customer self-service)
CR6	Enable businesses to tailor the application's user interface, colors, logos, and branding elements to their company's identity (customization and branding)

Experimental Environment. The study was conducted as part of adaptive and perfective maintenance for the SDA, which was already developed by the developers in the company. Table 34.2 shows the change request for the SDA issued by stakeholders. The change requests were chosen based on their impact on stakeholder satisfaction and the overall success of the change implementation. Due to limited time and resources, we chose only six change requests. However, we wanted to focus on the most critical areas that would have the biggest impact on stakeholder satisfaction and overall success.

The **participants** in the study were software maintainers and customers of the SDA application. The participants of the study were selected based on their expertise in software maintenance and their familiarity with the SDA. They included both software maintainers who were responsible for handling change requests for the SDA, as well as customers of the application who regularly interacted with the software. This diverse group of participants provided valuable insights into the effectiveness of the proposed method and its impact on both the maintenance team and the customers.

Following the steps of the proposed method, the following are the primary activities that resulting in producing the **generated artifacts** during the execution of the case study: First, the software maintainers (1) identified the change request and prepared an *initial list of features*. The list was used to design the *sub-FM* that represents the change request. In the second step, the software maintainers (2) refined the requirements by evolving the base FM with the sub-FM, resulting in the *composed FM*. Finally, they (3) validated the specified modifications, resulting in a *check list* that can be used by reviewers to validate that the modified features of the software product conform to the specifications of the change request form (i.e., to validate that the modifications have been successfully implemented).

UNCORRECTED PROOF

629635_1_En_34_Chapter ☑ TYPESET ☐ DISK ☐ LE ☑ CP Disp.:7/1/2025 Pages: 11 Layout: T1-Standard

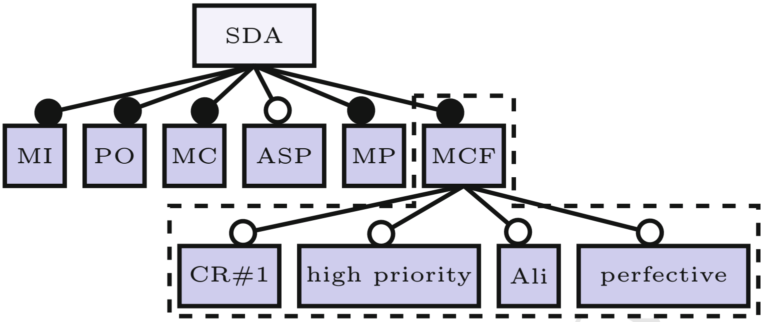

Fig. 34.4 Composed FM of the sales distribution application (SDA) of the case study. MI: manage inventory, PO: place order, MC: manage customer, ASP: automated sales process, MP: manage pricing, MCF: manage customer feedback

In detail, the scenario of responding to each change request in the case study was as follows: the request for a change of the SDA (e.g.,**Change Request 1** in Table 34.2) is identified by collecting data from the change request form as a feature list. By the end of this step, a sub-FM that represents a single change request has been designed (Fig. 34.4b). After structuring the sub-FM, it is time to refine the base FM (Fig. 34.4a) with the information of **Change Request 1** designed previously as a sub-FM (Fig. 34.4b) to produce the *composed FM* (Fig. 34.4a, b). In the last step, change reviewers can use the composed FM (Fig. 34.4a) to check that all the features of the SDA were updated consistently to conform to the specifications of **Change Request 1**. The steps of the case study were repeated for each change request presented in Table 34.2. This allows us to thoroughly assess the performance of software maintainers and customer satisfaction with each request and evaluate its feasibility within the given context. Considering the limitations of space and to ensure optimal readability, we have only selected and presented the composed FM that includes the sub-FM of **Change Request 1**.

34.4 The Analysis and Interpretation

Based on the performance of software maintainers in fulfilling the change requests using both the feature-based method and the SMLC method, as well as the customers' questionnaire, we acquire metrics. Then, the null hypotheses presented early in this manuscript can be rejected, and the alternative hypotheses can be accepted. It is worth mentioning that the performance (effectiveness and efficiency) metrics of software maintainers for the SMLC method (see third column of Tables 34.3 and 34.4) were derived from existing records meticulously maintained by the company's quality team.

UNCORRECTED PROOF

629635_1_En_34_Chapter ✓ TYPESET DISK LE ✓ CP Disp.:7/1/2025 Pages: 11 Layout: T1-Standard

Table 34.3 Effectiveness for our method and the SMLC

	CR1	CR2	CR3	CR4	CR5	CR6	Mean
Our method	88	89	95	94	89	89	90.7
SMLC	79	83	90	89	88	85	85.7

Table 34.4 Efficiency for our method and the SMLC

	CR1	CR2	CR3	CR4	CR5	CR6	Mean
Our method	25	17	22	27	28	3	20.3
SMLC	2	1	17	18	28	25	15.2

Table 34.3 shows a summary of the Effectiveness values, obtained from the execution of the change requests presented in Table 34.2. The dependent variable Effectiveness of each software maintainer was measured as the quotient of the right number of change requests that the software maintainer identified by the total number of change requests. As presented in last column of Table 34.3, the mean value of Effectiveness for our feature-based method (90.7) is greater than the mean value for SMLC (85.7). This indicates that the proposed method aids in improving the implementation of change requests while preserving the feature structure of the application.

Similarly, Table 34.4 summarizes the values of the Efficiency. These values were calculated as the ratio between the number of right change requests that the software maintainer identified and the total time spent on the identification (number of right identified change requests/time). The results show that the mean value for the feature-based method (20.3) is greater than the mean value for SMLC (15.2).

The results of Tables 34.3 and 34.4 enable researchers to reject the null hypotheses (Hp_0 and Hs_0) and accept the alternative ones (Hp_1 and Hs_1). The rejection of the hypotheses indicates that the proposed method can handle the implementation of change requests effectively and efficiently while minimizing the risk of introducing inconsistencies or breaking existing features. It also proves that there is a significant improvement in the performance of software maintainers in addressing change requests using the feature-based method compared to the SMLC.

Table 34.5 summarizes the average of rating scale answers related to the representative sample of customers and software maintainers, who were asked to fill the IBM questionnaire (available on https://github.com/karamignaim/change-request) to determine the overall perception of the usability of the proposed method and SMLC. The mean value of stakeholder responses showed a significant positive impact on change requests, indicating that the feature-based method efficiently implements change requests and meets modification requirements. The results shown in Table 34.5, in which the mean of the proposed method (1.92) is less than the mean of SMLC (2.83), allows the software maintainers to reject the null hypotheses (Hp_0 and

UNCORRECTED PROOF

Table 34.5 Stakeholder satisfaction for customers (C) and software maintainers (M), according to their answers to the IBM questionnaire

	C1	C2	C3	C4	C5	C6	M1	M2	M3	M4	M5	M6	Mean
Our method	2	3	2	2	2	1	2	3	2	1	2	1	1.92
SMLC	4	3	2	3	3	2	3	3	3	2	4	2	2.83

A value is better if it is closer to 1

Hs_0) and prove the alternative ones (Hp_1 and Hs_1). Generally, the values give a positive indicator that the method can help companies gain high customer satisfaction in responding to change requests.

The proposed method enhances company maintenance, addressing change requests efficiently and effectively, thereby increasing customer satisfaction, adaptability, and preserving positive relationships.

34.5 Discussion

The study proposes a feature-driven method for SMLC to fulfill change requests and satisfy requirement modifications. This method is effective in adaptive and perfective maintenance, improving efficiency and effectiveness. The refinement of the feature method enhances the use of refactoring patterns in software maintenance.

The study demonstrates the effectiveness of feature-based methods in completing change requests and improving software quality. It emphasizes the practicality of these methods in real-world scenarios and suggests the need for automation to streamline processes. This shift aligns with industry trends and best practices, reducing errors, increasing productivity, and delivering higher-quality software products. The SDA case study underscores the benefits of feature-driven methods and automation in software development.

34.6 Conclusions

Maintenance is essential for software development since it facilitates the evolution of software. This paper introduces a feature-based method to handle adaptive and perfective maintenance. The adaptive and perfective maintenance of a product is performed in a systematic way through a set of steps that are incorporated into the phases of SMLC. We have successfully used FMs and applied the FM composition operator to help maintainers handle change requests in a way that preserves the underlying FM structure. This goes on until the base FM is fully updated with all of the needed change requests. We have demonstrated our method on real-world

UNCORRECTED PROOF

product change requests. The results show that the feature-based method can assist companies in responding effectively and efficiently to change requests in adaptive and perfective maintenance with high customer satisfaction.

References

1. Acher, M., Collet, P., Lahire, P., France, R.: Composing feature models. In: 2nd International Conference on Software Language Engineering (SLE), pp. 62–81. (2009). 10.1007/978-3-642-12107-4_6
2. Caldiera, G., Basili. V., Rombach, H.D.: Encyclopedia of software engineering. In: The Goal Question Metric Approach. pp. 528–532. Wiley and Sons (1994)
3. Chemuturi, M.: In: Requirements Engineering and Management for Software Development Projects. Springer (2013). https://doi.org/10.1007/978-1-4614-5377-2
4. Fernandes, J.M., Machado, R.J.: Requirements in Engineering Projects. Springer (2016). https://doi.org/10.1007/978-3-319-18597-2
5. Kim, G.J.: Human-Computer Interaction: Fundamentals and Practice. CRC Press (2015)
6. Oliveira, R.P., Almeida, E.S.: Requirements evolution in software product lines: an empirical study. In: 9th Brazilian Symposium on Components, Architectures and Reuse Software (SBCARS), pp. 1–10. (2015). 10.1109/SBCARS.2015.11
7. Sommerville, I.: In: Software Engineering. 9th edn. Pearson (2011)
8. Taskesenlioglu, S., Ozkan, N., Erdogan, T.G.: Identifying possible improvements of software development life sycle (SDLC) process of a bank by using process mining. Int. J. Software Eng. Knowledge Eng. **32**(4), 525–552 (2022). https://doi.org/10.1142/S0218194022400010
9. Tripathy, P., Naik, K.: In: Software Evolution and Maintenance: A Practitioner's Approach. Wiley and Sons (2014)

UNCORRECTED PROOF

629635_1_En_34_Chapter ✓ TYPESET DISK LE ✓ CP Disp.:7/1/2025 Pages: 11 Layout: T1-Standard