Measurement of the Support in the Development Stages of an Aspect-Oriented Software Product Line Framework

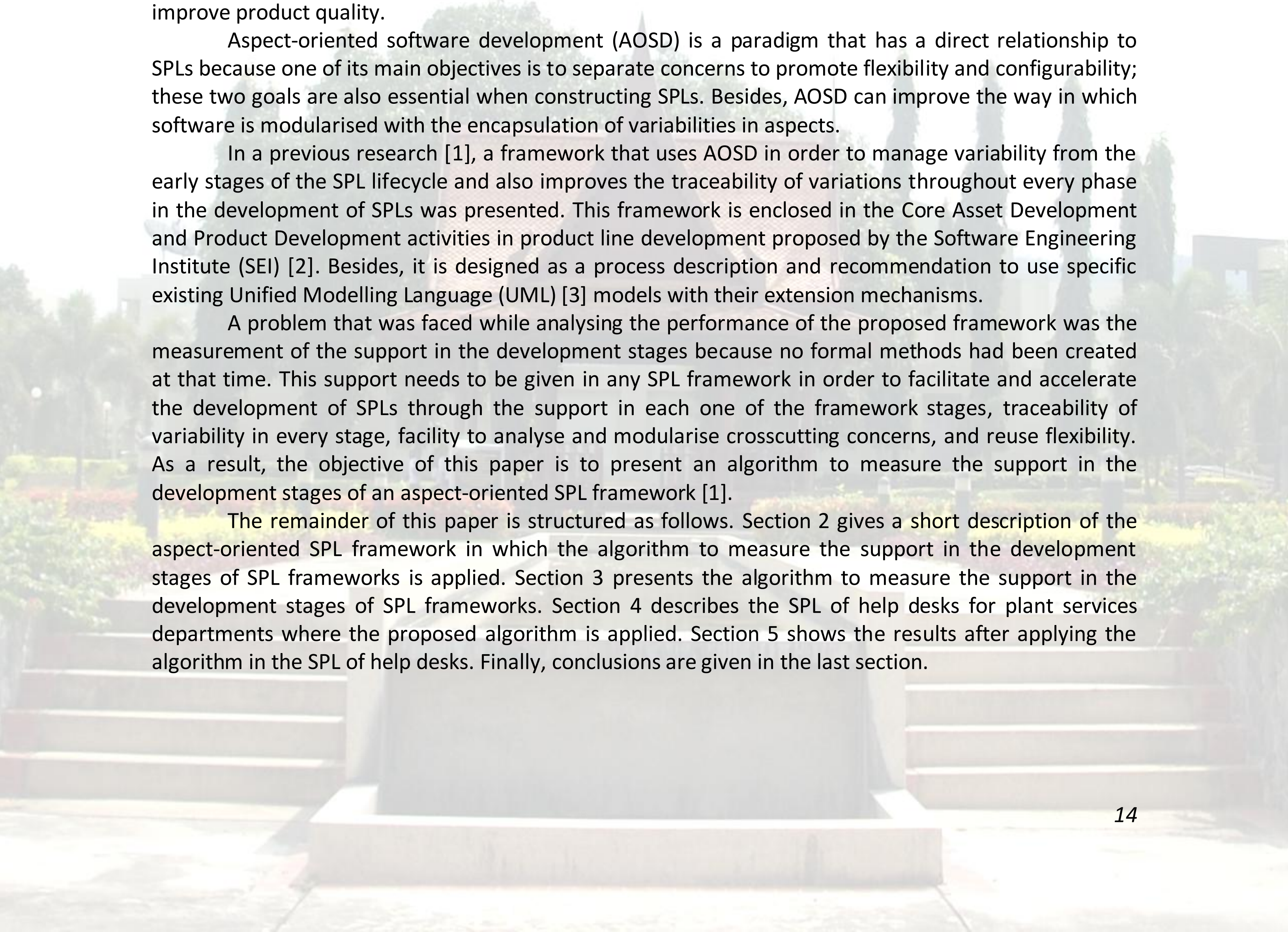
Germán Harvey Alférez Salinas

Abstract

A performance goal of a software product line (SPL) framework is the support in the development stages. In spite of its importance, previous researches do not formalise a way to measure this goal.

This paper presents an algorithm based on four metrics that can be used to formalise the measurement of the support in the development stages of SPL frameworks. Specifically, this algorithm is applied in an aspect-oriented SPL framework that was presented in a previous research [1].

Introduction



SPL engineering is about exploiting commonalities among a set of systems while managing the variabilities among them in order to achieve systematic reuse goals, improve time to market, and improve product quality.

Aspect-oriented software development (AOSD) is a paradigm that has a direct relationship to SPLs because one of its main objectives is to separate concerns to promote flexibility and configurability; these two goals are also essential when constructing SPLs. Besides, AOSD can improve the way in which software is modularised with the encapsulation of variabilities in aspects.

In a previous research [1], a framework that uses AOSD in order to manage variability from the early stages of the SPL lifecycle and also improves the traceability of variations throughout every phase in the development of SPLs was presented. This framework is enclosed in the Core Asset Development and Product Development activities in product line development proposed by the Software Engineering Institute (SEI) [2]. Besides, it is designed as a process description and recommendation to use specific existing Unified Modelling Language (UML) [3] models with their extension mechanisms.

A problem that was faced while analysing the performance of the proposed framework was the measurement of the support in the development stages because no formal methods had been created at that time. This support needs to be given in any SPL framework in order to facilitate and accelerate the development of SPLs through the support in each one of the framework stages, traceability of variability in every stage, facility to analyse and modularise crosscutting concerns, and reuse flexibility. As a result, the objective of this paper is to present an algorithm to measure the support in the development stages of an aspect-oriented SPL framework [1].

The remainder of this paper is structured as follows. Section 2 gives a short description of the aspect-oriented SPL framework in which the algorithm to measure the support in the development stages of SPL frameworks is applied. Section 3 presents the algorithm to measure the support in the development stages of SPL frameworks. Section 4 describes the SPL of help desks for plant services departments where the proposed algorithm is applied. Section 5 shows the results after applying the algorithm in the SPL of help desks. Finally, conclusions are given in the last section.

Aspect-Oriented Framework to Manage Variability in Software Product Lines

Fig. 1 shows the two main activities of an aspect-oriented SPL framework [1], Domain Engineering and Application Engineering, and their mapping with the SEI's activities. Each activity has its own development cycle. The bidirectional rows indicate that it is an interative process and that traceability can be done between any stage and between the two development cycles.

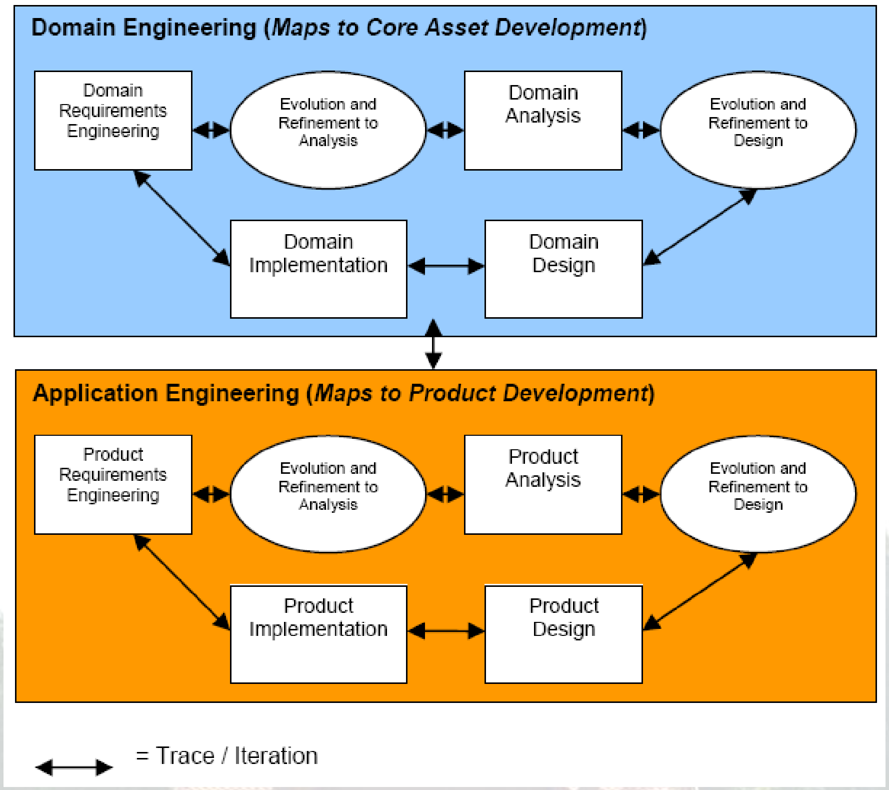

Domain Engineering (*Maps to Core Asset Development*)
Domain Requirements Engineering
Evolution and Refinement to Analysis
Domain Analysis
Evolution and Refinement to Design
Domain Implementation
Domain Design
Application Engineering (*Maps to Product Development*)
Product Requirements Engineering
Evolution and Refinement to Analysis
Product Analysis
Evolution and Refinement to Design
Product Implementation
Product Design
= Trace / Iteration

Figure 1. Aspect-Oriented SPL Framework [1]

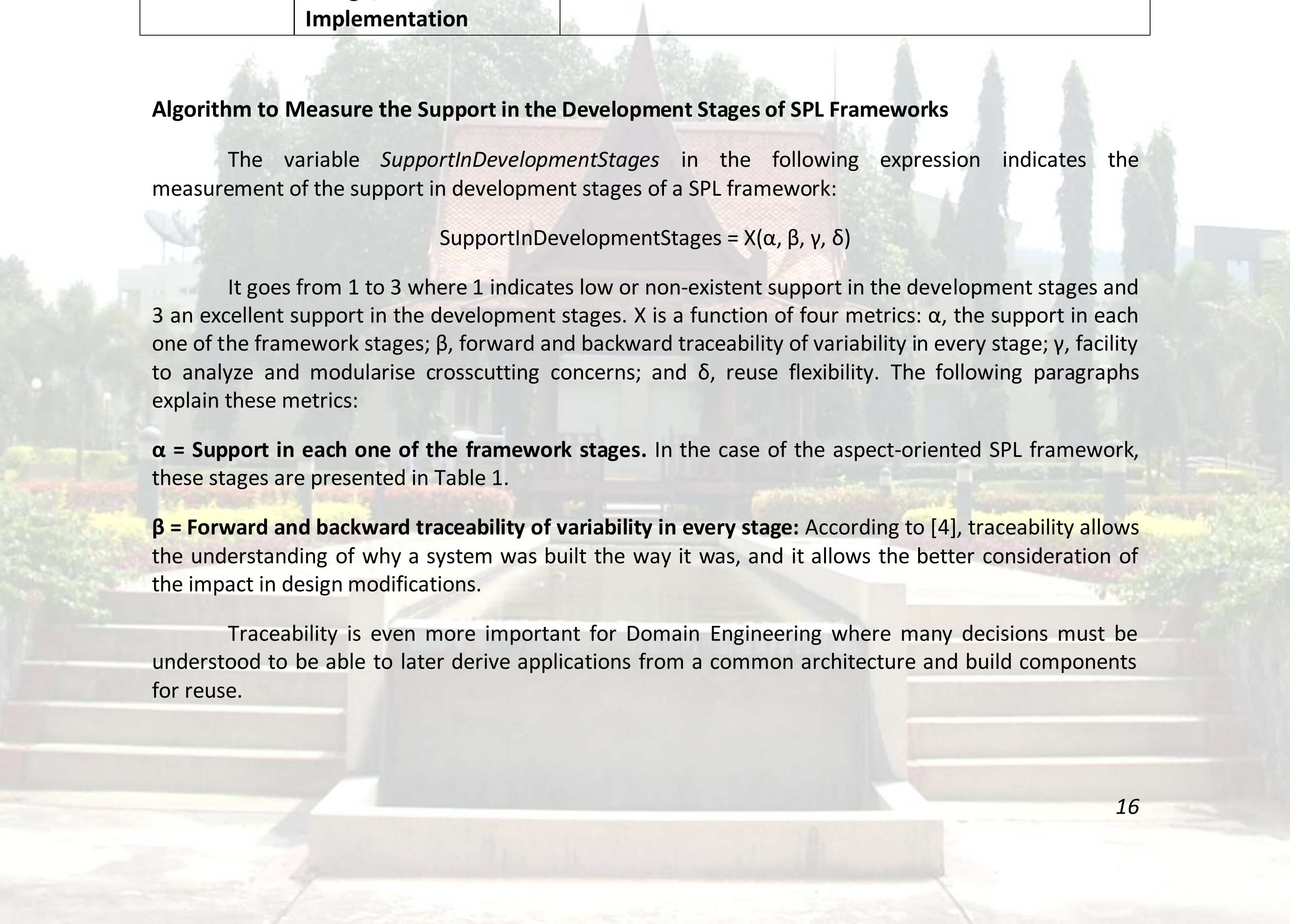
Domain Engineering is the process of SPL engineering in which the commonality and the variability of the product line are created. Application Engineering is the process of SPL engineering in which the applications of the product line are built by reusing artifacts.

Table 1 describes the development stages of the aspect-oriented SPL framework.

Table 1. Development Stages of the Aspect-Oriented SPL Framework

Activity	Development Stage	Description
Domain Engineering	**Requirements Engineering**	Functional and non-functional requirements common to the entire product line are represented through use cases with variation points that can be used to create product-specific requirements employing extensions and extension points.

	Evolution and Refinement to Analysis	Mapping rules are used to establish relationships between kernel, optional, and variant use cases, and concerns.
	Analysis	The feature, class, and sequence diagrams are created.
	Evolution and Refinement to Design	To facilitate the reuse and understanding of core assets in different products, kernel classes and aspects are grouped in different packages depending on their stereotypes and preserving the relationships that were created in the class diagram at the Domain Analysis phase.
	Design	The refined class model is created here.
	Implementation	Software components and aspects are developed for systematic reuse across the product line.
Application Engineering	**Requirements Engineering, Evolution and Refinement to Analysis, Analysis, Evolution and Refinement to Design, Design, and Implementation**	The product builders instantiate the production plan, recognising the variation points being selected for the given product depending on the variabilities that were discovered and defined as functional and non-functional aspects in every stage of the Domain Engineering activity.



Algorithm to Measure the Support in the Development Stages of SPL Frameworks

The variable *SupportInDevelopmentStages* in the following expression indicates the measurement of the support in development stages of a SPL framework:

$$\text{SupportInDevelopmentStages} = X(\alpha, \beta, \gamma, \delta)$$

It goes from 1 to 3 where 1 indicates low or non-existent support in the development stages and 3 an excellent support in the development stages. X is a function of four metrics: α, the support in each one of the framework stages; β, forward and backward traceability of variability in every stage; γ, facility to analyze and modularise crosscutting concerns; and δ, reuse flexibility. The following paragraphs explain these metrics:

α = Support in each one of the framework stages. In the case of the aspect-oriented SPL framework, these stages are presented in Table 1.

β = Forward and backward traceability of variability in every stage: According to [4], traceability allows the understanding of why a system was built the way it was, and it allows the better consideration of the impact in design modifications.

Traceability is even more important for Domain Engineering where many decisions must be understood to be able to later derive applications from a common architecture and build components for reuse.

γ = Facility to analyse and modularise crosscutting concerns: According to [5], “crosscutting” is how to characterise a concern that spans multiple units of object-oriented modularity - classes and objects. Crosscutting concerns resist modularisation using normal object-oriented constructs, but aspect-oriented programs can modularise crosscutting concerns because “we call a well modularised crosscutting concern an aspect” [6]. In addition, [1] demonstrates that variabilities in SPLs can be effectively encapsulated into aspects.

With the object-oriented approach, crosscutting concerns produce two big problems: scattering and tangling. “Scattering” is when similar code is distributed throughout many program modules. This differs from a component being used by many other components since it involves the risk of misuse at each point and of inconsistencies across all points. Changes to the implementation may require finding and editing all affected code [5].

“Tangling” is when two or more concerns are implemented in the same body of code or component, making it more difficult to understand. Changes to one implementation may cause unintended changes to other tangled concerns [7].

Scattered and tangled code carries out problems such as a weak understanding of the problem, inability to determine how a change in an artifact affects the others, increases the complexity of adding, removing or modifying requirements, and potentially has a high impact in changes–even the smaller changes in requirements can affect a great part of code and design [8].

δ = Reuse flexibility: An important concept related to SPLs is reuse [9]. According to [10], early efforts focused on small-grained reuse of software code. The cost of creation and use of these small-grained assets often outweighed the modest gains. Over the years, reuse technology has evolved to focus on progressively larger-grained assets. Using this approach, reuse can result in remarkable benefits, such as time to market and improved product quality. Fig. 2 shows the reuse evolution, from subroutines to SPLs.

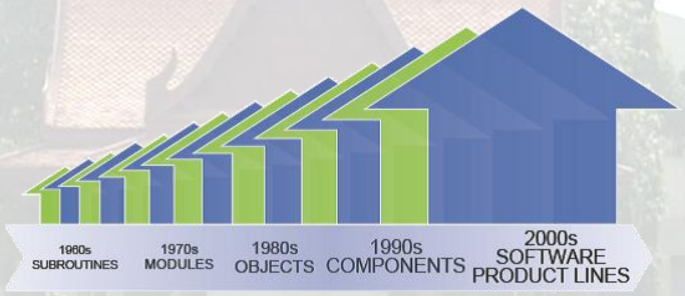
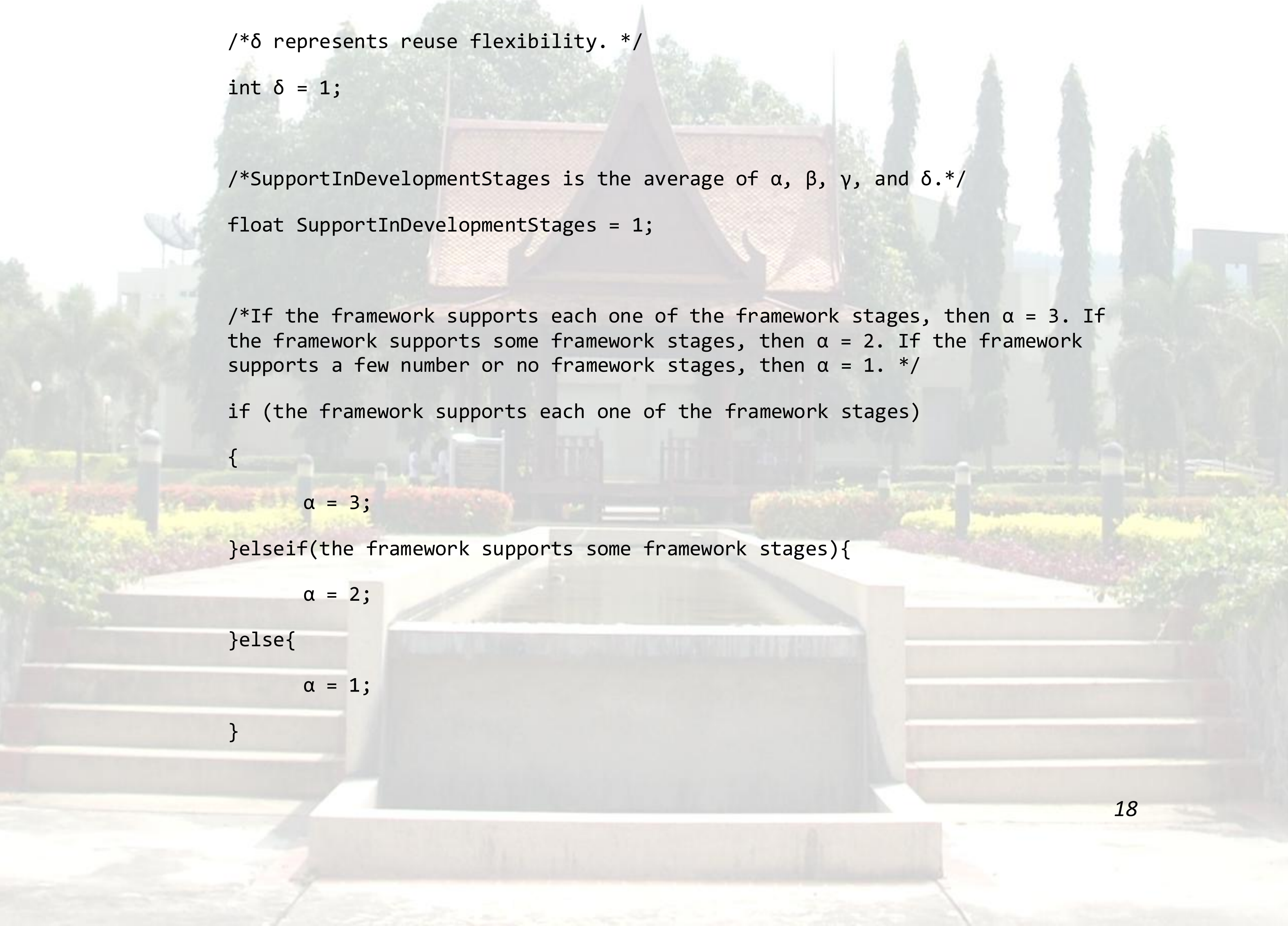


Figure 2. Reuse History: From Ad Hoc to Systematic [11]

Besides, SPLs can maximise the reuse of a wide variety of assets, such as requirements. Moreover, the knowledge and skills of project personnel and the methods used to develop and evolve a system, among other assets, are reusable [12].

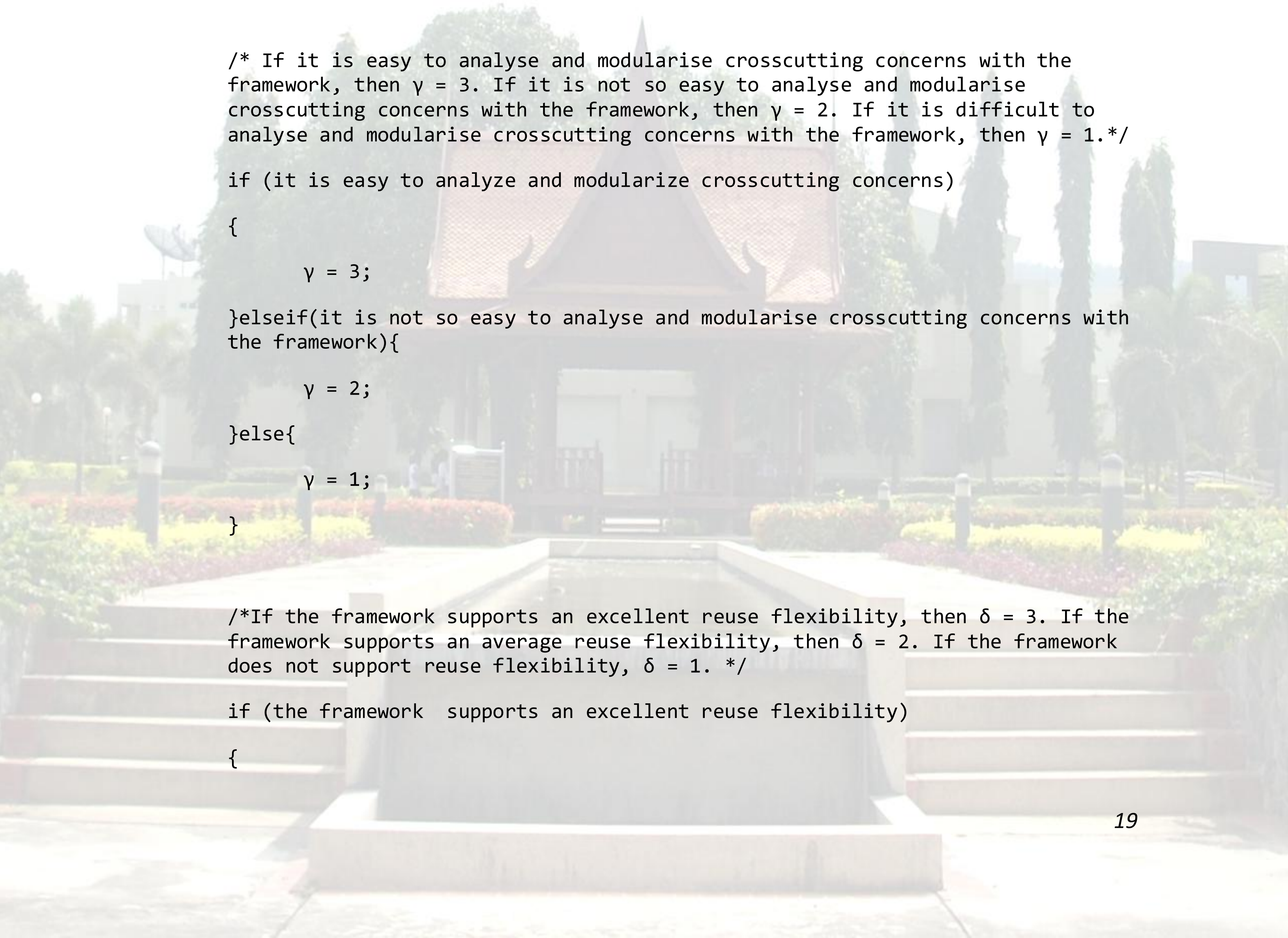
The algorithm to measure the support in the development stages of SPL frameworks is given in Fig. 3.

```
SupportInDevelopmentStagesCalculation(){
```



```
/*SupportInDevelopmentStages measures the support in the development stages.*/

float SupportInDevelopmentStages = 1;


/*α represents the support in each one of the framework stages.*/

int α = 1;


/*β represents traceability of variability in every stage.*/

int β = 1;


/*γ represents the facility to analyze and modularise crosscutting concerns.
*/

int γ = 1;


/*δ represents reuse flexibility. */

int δ = 1;


/*SupportInDevelopmentStages is the average of α, β, γ, and δ.*/

float SupportInDevelopmentStages = 1;


/*If the framework supports each one of the framework stages, then α = 3. If
the framework supports some framework stages, then α = 2. If the framework
supports a few number or no framework stages, then α = 1. */

if (the framework supports each one of the framework stages)

{

      α = 3;

}elseif(the framework supports some framework stages){

      α = 2;

}else{

      α = 1;

}
```

```
/*If the framework provides an excellent traceability of variability in every
stage, then $\beta = 3$. If the framework has an average traceability of variability
in every stage, then $\beta = 2$. If the framework has a poor traceability of
variability in every stage, then $\beta = 1$.*/

if (the framework provides an excellent traceability of variability in every
stage)

{

      $\beta = 3$;

}elseif(the framework provides an average traceability of variability in every
stage){

      $\beta = 2$;

}else{

      $\beta = 1$;

}


/* If it is easy to analyse and modularise crosscutting concerns with the
framework, then $\gamma = 3$. If it is not so easy to analyse and modularise
crosscutting concerns with the framework, then $\gamma = 2$. If it is difficult to
analyse and modularise crosscutting concerns with the framework, then $\gamma = 1$.*/

if (it is easy to analyze and modularize crosscutting concerns)

{

      $\gamma = 3$;

}elseif(it is not so easy to analyse and modularise crosscutting concerns with
the framework){

      $\gamma = 2$;

}else{

      $\gamma = 1$;

}


/*If the framework supports an excellent reuse flexibility, then $\delta = 3$. If the
framework supports an average reuse flexibility, then $\delta = 2$. If the framework
does not support reuse flexibility, $\delta = 1$. */

if (the framework  supports an excellent reuse flexibility)

{
```

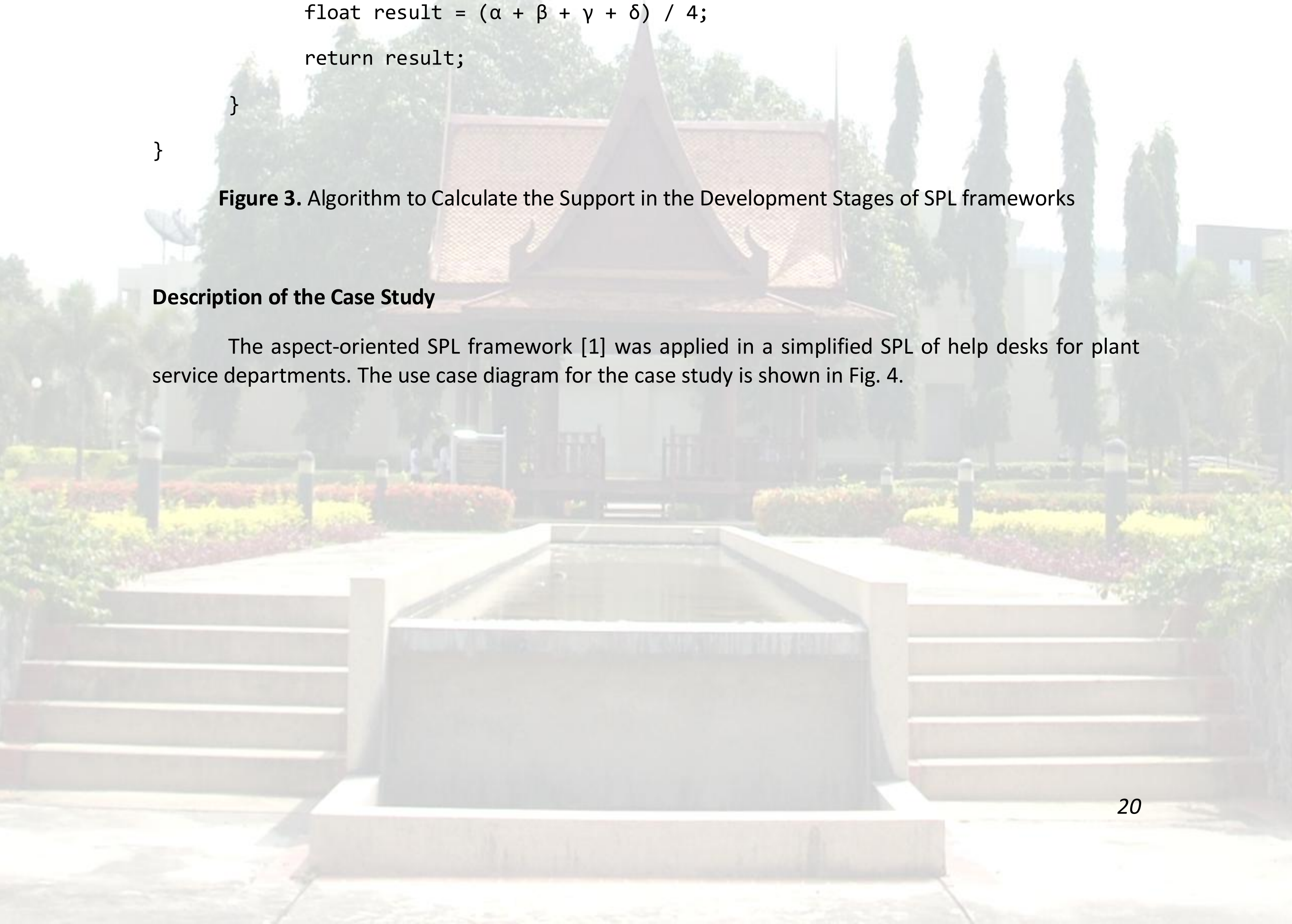


```
                δ = 3;
        }elseif(the framework provides an average reuse flexibility){
                δ = 2;
        }else{
                δ = 1;
        }

        /*The result of X function is assigned to SupportInDevelopmentStages.*/
        SupportInDevelopmentStages = X(α, β, γ, δ);

        /*Calculation of the four parameters’ average: α, β, γ, δ.*/
        float X(int α, int β, int γ, int δ)
        {
                float result = (α + β + γ + δ) / 4;
                return result;
        }
}
```

Figure 3. Algorithm to Calculate the Support in the Development Stages of SPL frameworks

Description of the Case Study

The aspect-oriented SPL framework [1] was applied in a simplified SPL of help desks for plant service departments. The use case diagram for the case study is shown in Fig. 4.



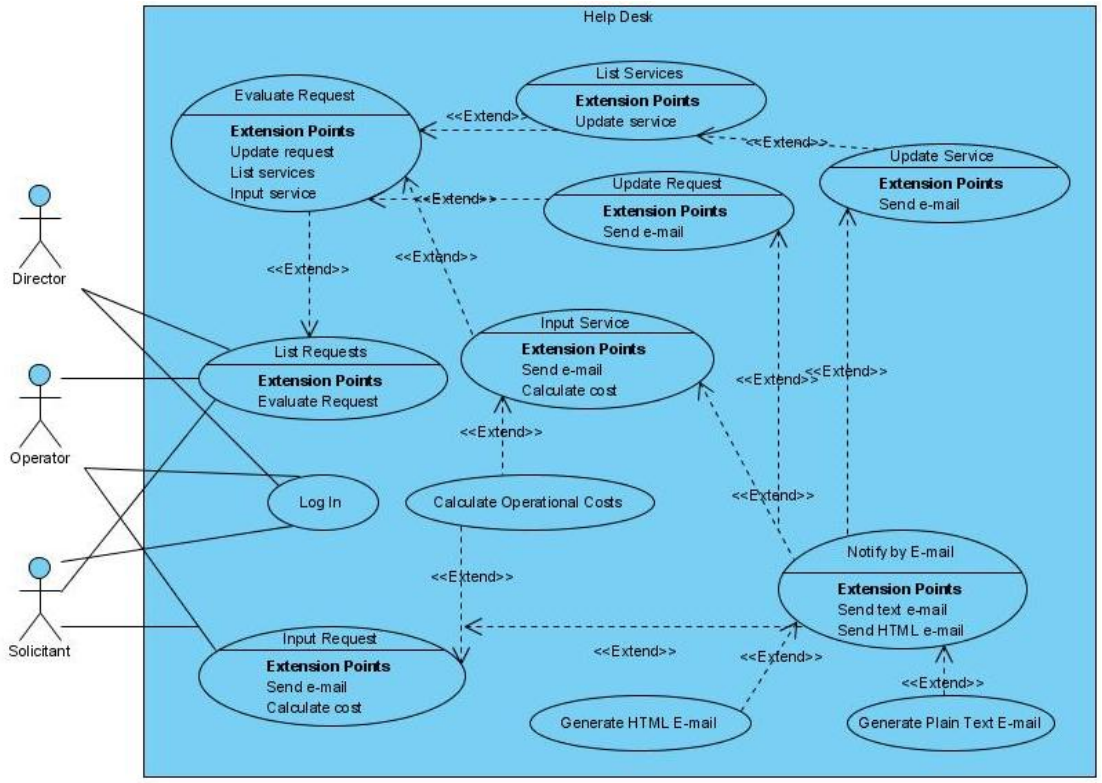


Figure 4. Use Case Diagram for the SPL of Help Desks

There are three roles: Director - he/she is the director of the help desk department. He/she only lists requests; Operator – he/she inputs requests and their specific services, lists requests and services, and updates requests and services; and Solicitant - he/she can submit requests using a Web interface and list his/her requests with their respective services. These three actors must log in into the system to do operations over requests and services.

Some products of the product line require the calculation of the operational costs when there is an input request or input service operation.

Besides, some products require concurrency control for the input request and input service operations (this is a non-functional requirement).

On the other hand, some products of the product line measure the performance (this is a non-functional requirement) of the input request, list requests, input service, and list services operations. In order to take this measure, some products calculate the execution time of these operations in milliseconds and others in microseconds. This calculations help to improve the levels of service quality.

Finally, some products notify by e-mail to the solicitant when a new request or service has been input or updated, or to the operator when a solicitant has input a request from the Web interface. These e-mails can be generated in plain text or in HTML format.

Support in the Development Stages' Results

The algorithm to calculate the support in the development stages (Fig. 3) was applied in the SPL of help desks (section 4). Table 2 shows the results.

Table 2. Framework Goal, Metrics, and Results of the Algorithm to Calculate the Support in the Development Stages Applied in a SPL of Help Desks that was Built Using an Aspect-Oriented SPL Framework

Framework Goal	**Metrics**	**Results**	
Support in the development stages	Support in each one of the framework stages	3	3
	Traceability of variability in every stage	3	
	Facility to analyse and modularise crosscutting concerns	3	
	Reuse flexibility	3	

In the case study, the framework supported each one of the development stages by the easy to follow instructions, such as the ones in the Evolution and Refinement to Analysis phase to establish relationships between kernel, optional, and variant use cases, and concerns. As a result, the support in each one of the framework stages is equal to 3 ($\alpha = 3$).

Besides, the framework provided an excellent traceability support from the very beginning in the construction of the case study. Forward and backward traceability could be easily done through the UML models, conversion rules, and a table of kernel concerns vs. crosscutting concerns with variants. So, traceability of variability in every stage is equal to 3 ($\beta = 3$).

Also, it was very easy to analyse and modularise crosscutting concerns because the aspect-oriented framework allows the encapsulation of variability in aspects and as a result the analysis and modularisation was drastically improved, without scattered and tangled code; these problems were solved with the use of aspects for measuring performance, calculating the operational costs, controlling the concurrency, and notifying by e-mail. As a result, the “facility to analyze and modularize crosscutting concerns” metric is equal to 3 ($\gamma = 3$).

In addition, variability is easily implemented and managed using aspects that can be plugged or unplugged from the SPL in order to create new customised products. As a result, reuse flexibility is equal to 3 ($\delta = 3$).

In conclusion, the support in the development stages is equal to 3:

$$(\alpha + \beta + \gamma + \delta) / 4 =$$

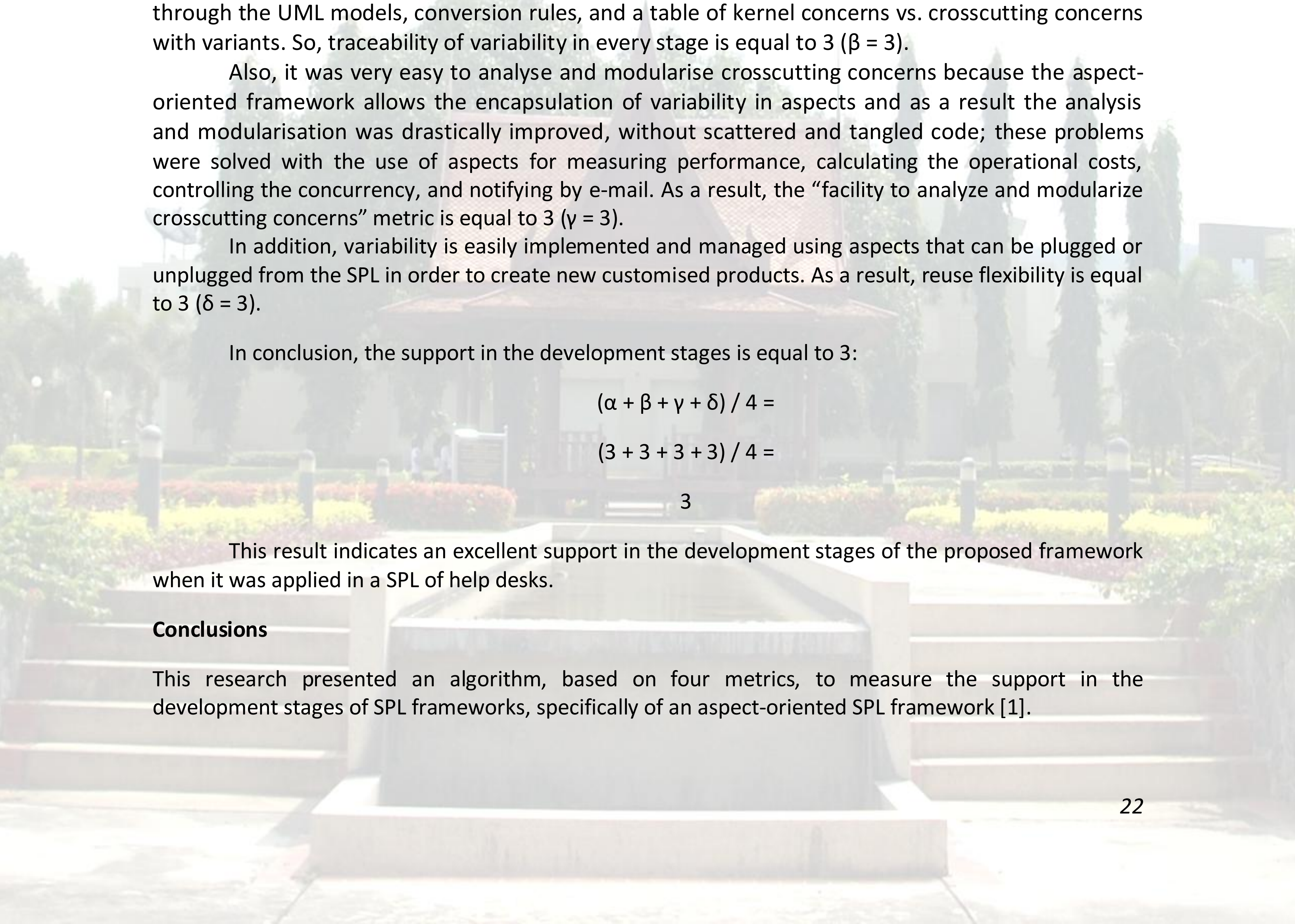
$$(3 + 3 + 3 + 3) / 4 =$$

$$3$$

This result indicates an excellent support in the development stages of the proposed framework when it was applied in a SPL of help desks.

Conclusions

This research presented an algorithm, based on four metrics, to measure the support in the development stages of SPL frameworks, specifically of an aspect-oriented SPL framework [1].



The result of the proposed algorithm goes from 1 to 3 where 1 indicates non-existent support in the development stages and 3 an excellent support in the development stages. In this study, the aspect-oriented framework for SPLs got the highest score in the support of development stages.

Finally, this research allows software companies to apply the proposed algorithm in different SPL frameworks and choose the one that offers the best support in the development stages.

Works Cited

[1] Germán Harvey Alférez and Poonphon Suesaowaluk, "An Aspect-Oriented Product Line Framework to Support the Development of Software Product Lines of Web Applications," Proceedings of the South East Asia Regional Computer Conference 2007 (SEARCC 2007), Thailand. November 18-19, 2007

[2] Paul Clements and Linda Northrop, "Software Product Lines: Practices and Patterns," Addison-Wesley, 2002.

[3] Object Management Group, Inc., "Unified Modeling Language," url: http://www.uml.org, [Accessed 8 January 2008]

[4] Michael Schlick and Andreas Hein, "Knowledge Engineering in Software Product Lines," Proceedings of the European Conference on Artificial Intelligence (ECAI 2000), Workshop on Knowledge-Based Systems for Model-Based Engineering, August 22 , 2000, Berlin, Germany.

[5] Xerox Corporation, Palo Alto Research Center, "Frequently Asked Questions about AspectJ," url: http://www.eclipse.org/aspectj/doc/released/faq.html, [Accessed 16 February 2007].

[6] Gregor Kiczales, *et al*, "An Overview of AspectJ," Proceedings of the European Conference on Object-Oriented Programming (ECOOP), 2001. Springer-Verlag, 2000.

[7] Mik Kersten, "AspectJ: The Language and Development Tools," url: http://www.parc.com/research/projects/aspectj/, [Accessed 2 April 2007].

[8] Siobhán Clarke, *et al*, "Separating Concerns throughout the Development Lifecycle," Proceedings of ECOOP '99 Workshop, 1999. Springer-Verlag, 1999.

[9] Linda Northrop, "Reuse that Pays," Proceedings of the 23rd International Conference on Software Engineering (ICSE'01), 2001. IEEE Computer Society, 2002.

[10] John Bergey, Matt Fisher, Brian Gallagher, Lawrence Jones, and Linda Northrop, "Basic Concepts of Product Line Practice for the DoD," Technical Note CMU/SEI-2000-TN-001, Software Engineering Institute, Carnegie Mellon University, 2000.

[11] Marco Sinnema, *et al*, "COVAMOF: A Framework for Modeling Variability in Software Product Families," Lecture Notes in Computer Science, vol. 3154/2004, pp. 197-213, 2004

[12] Lisa Brownsword, Paul Clemens, and Ulf Olsson, "Successful Product Line Engineering: A Case Study," Proceedings of the Software Technology Conference, Salt Lake City, April, 1996.

About the Author:

At the time of publication, Germán Harvey Alférez Salinas is serving as Lecturer in the Faculty of Engineering and Technology at Montemorelos University, Mexico. Harvey Salinas also taught at Mission College and was Department Head for its CIS programme. At present he also serves as adjunct lecturer for Mission College.