Applying System Families Concepts to Requirements Engineering Process Definition[⋆]

Amador Durán[1], David Benavides[1], and Jesus Bermejo[2]

[1] Department of Computer Languages and Systems
University of Seville, Reina Mercedes S/N, 41012 Seville, Spain
amador@lsi.us.es,benavides@us.es
[2] Telvent
Tamarguillo, 29, 41006 Seville, Spain
jesus.bermejo@telvent.abengoa.com

Abstract. In this paper, some experiences gained during the definition of a unified, common software development process for several companies in Telvent are presented. Last year, Telvent made the decision of developing a unique software development process which was flexible enough to be adapted to specific practices and needs of the different companies. In this paper we focus mainly on the experiences gained during the definition of the requirements engineering process, al-though many of them are also applicable to other software development processes. One of the most interesting experiences from our point of view is that, al-though the definition process was started using a top-down approach and well-know techniques like data flow diagrams, we eventually end up applying requirements engineering techniques like glossaries, scenarios or conflict resolu-tion for the definition of the requirements engineering process itself. On the other hand, the need of having adaptable processes for the different companies in Tel-vent made us adopt a *process family* approach, i.e. adopting an approach similar to the system families development, thus defining a core process that could be adapted to specific needs of specific companies in a predefined, controlled man-ner. The experiences gained in the definition of the process family were applied to the definition of requirements engineering process for product line development, which is briefly presented in this paper.

Keywords: Requirements Engineering Process, Systems Families

1 Introduction

Definition and adoption of requirements engineering (RE) processes in software development companies is a complex task not easy to accomplish [10,13]. This task is even harder in a company like Telvent, composed of several companies developing a wide spectrum of systems like information systems, real–time control systems or satellite communication systems.

Each company has its own specific needs for developing software, depending on the type of software, the customer characteristics, the standards to be applied, if they are project-oriented or product-oriented, etc.

[⋆] This work is partially supported by the Eureka Σ! 2023 programme, ITEA ip00004 project CAFÉ and by the Spanish CICYT project TIC 2000–1106–C02–01 (GEOZOCO).

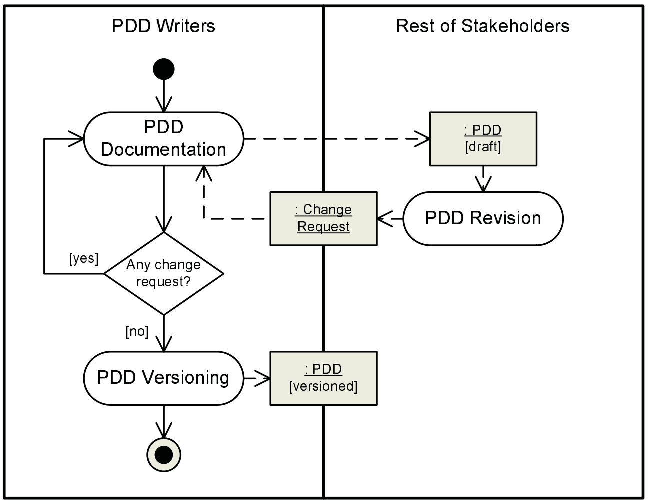

Fig. 1. PDD Iterative Workflow

In 2002, Telvent decided to adopt an ISO–12207–based [9], common software development process that should fit the needs of all its companies thus avoiding the maintenance of several internal standards and saving training costs when some employees had to move from one company to another within Telvent. For that purpose, some research staff from the Department of Computer Languages and Systems of the University of Seville were hired by Telvent as consultants. In this context a process family approach was used where a commin core process was defined. This process serve as base for specific processes in different companies therefore it includes *process variability*

For the definition of the common processes, at least one person from each company, including quality assurance people and consultants from the University of Seville, was selected. Depending on the specific process, i.e. requirements engineering, software design, software testing, etc., one or two persons were selected as responsible for writing a draft version of the corresponding process description document (PDD). Then, the draft version of the PDD was reviewed during meetings in which all stakeholders took part. The people responsible for the PDD registered the proposed changes and then presented an updated version of the PDD in the next meeting. This iterative process ended when no more change proposals were submitted and the PDD was baselined, as shown in the UML activity diagram in figure 1.

In this paper, we focus on the problems we found, the solutions we applied and the experience we gained during the definition of the RE process. The rest of the paper is organized as follows. In section 2, some initial problems are described. In section 3 we present how we addressed the variability in the core RE process so it could be adapted to different needs of the companies in Telvent. In section 4 we briefly described the RE process for product line developed applying some of the experiences described in previous sections. Finally in section 5 we present some conclusions.

2 Initial Problems

In this section we present some of the initial problems we identified at the beginning of the definition process and how they were, totally or partially, solved applying RE techniques.

2.1 Lack of a Common Vocabulary

One of the first detected problems was the lack of a common vocabulary among all stakeholders. Depending on their backgrounds, the type of software they were used to developing or the standards they had had to apply in previous developments, the vocabularies of the involved people were quite different from each other. This problem provoked that many hours in the initial meetings were wasted discussing subtle semantic aspects about some words or phrases.

In order to solve this problem, and after some disappointing meetings, we decided to follow Leite's approach [11] of building a glossary (also know as *lexicon*) at the beginning of the process in order to understand the language of the problem before the problem itself. Items in the glossary should defined not only their corresponding concepts (*notions* in Leite's terminology) but also their interactions (*behavioral responses* in Leite's terminology) with other concepts, as shown in figure 2, where references to other glossary items are underlined[1]. As a matter of fact, the glossary was the first official document in which we started to work.

...

Change request: Request for the modification of any item previously baselined. The author of the change request can be any member of the Software Development Group or any other person working in the corresponding product line. The motivation for the change request can be an error detection in the corresponding product or an enhance suggestion. Change requests must be processed by the Change Control Board.

...

Change Control Board: Group composed of senior programmers from the Software Development Group, the Product Test Manager, the Configuration Control Manager, and the Documentation Manager. The mission of this board is to manage all change requests, perform the corresponding impact analysis and allocate the responsibilities for the change process.

...

Fig. 2. Glossary excerpt

If some stakeholders did not agree about the meaning of a specific concept even after consulting standards glossaries like [8], we voted what we considered the meaning of the concept and the glossary was updated democratically. In this way, we saved time and avoid personal confrontation apart of developing a comprehensive and useful glossary of software development terms.

[1] In the HTML version of the glossary, references to other items were actually hyperlinks.

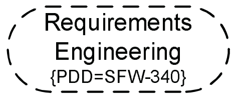

Fig. 3. Adopted notation for complex activities and use of tagged values

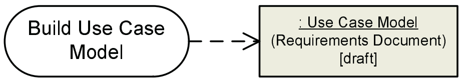

Fig. 4. Adopted notation for *whole/part* object flows

2.2 Lack of Common Process Notation

Another initial problem was the selection of a diagrammatic process notation that was accepted by all stakeholders and helped them have a way of visualizing processes' sequence of tasks and a general overview of process interactions.

Data flow diagrams (DFD) [15] were rapidly discarded because of the impossibility of representing neither sequence of activities nor conditional flows. After discarding DFDs, UML activity diagrams [12] were considered as a tentative notation and eventually adopted with some adaptations from the original notation. These adaptations were the following:

1. In order to know what PDD described the corresponding activity in an activity diagram, we added a *tagged value*[2] in every activity which was described in a PDD, as shown in figure 3. The tagged value was named *PDD* and its value was the corresponding code of the PDD in which the activity was described in more detail.
2. Although in the last version of UML (UML 1.4 [12]), complex activities are depicted by adding a small activity diagram in the lower left corner of the activity icon, the lack of a CASE tool supporting this and other changes introduced in UML 1.4 made us adopt an easier-to-draw notation. In the adopted notation, a complex activity is drawn using a dash line, as show in figure 3.
3. More often than not, output object flows of activities were parts of documents to be generated during some process. In order to show this in the activity diagrams, we adopted the notation for object flows that can be seen in figure 4, in which for *part* objects, the corresponding *whole* object name is also included in the object flow icon right under the name of the *part* object inside parentheses. In the example depicted in figure 4, the Use Case Model object flow is part of the Requirements Document object flow.
4. Those object flows that were shared by two or more processes, i.e. *interface object flows*, were depicted adding a shadow on the lower left corner of the object flow icon, as shown in figure 5.

[2] *Tagged values* are one of the extensibility mechanisms of UML. Tagged values are pairs {*name*=*value*} used to add extra information to model elements.

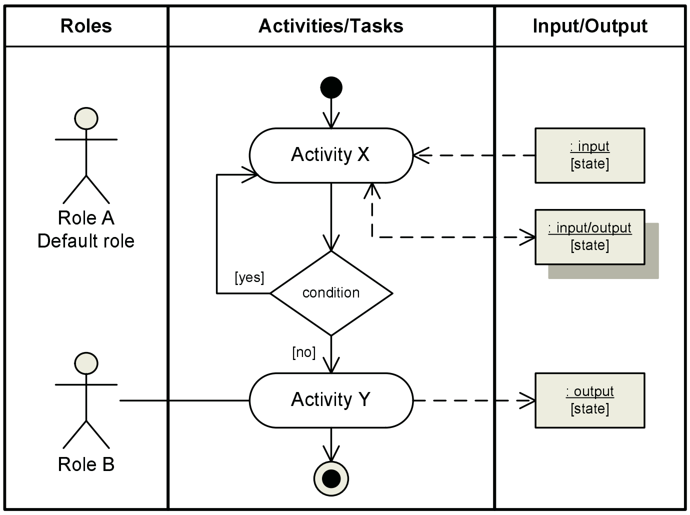

Fig. 5. Adopted notation for swimlanes

5. When many actors participated in an activity diagram, the use of more than three or four different swimlanes made the diagrams very complex to be drawn and difficult to read. On the other hand, a high number of object flows in a diagram resulted in too much crossing lines. In order to avoid these two problems, a different use of swimlanes was adopted.
   We decided to use only three swimlanes. The first swimlane was used for including *roles*, i.e. actors performing or participating in some activity in the diagram. The second swimlane was used for activities and tasks. The third one was used for input/output object flows. In this way, the resulting diagrams were much more easier to be created and read (see figure 5). If a role participated in an activity, an association between the role and the activity was added to the diagram.
   In the case a role participated in all activities in a diagram, the role was considered as a *default role* and associations between the default role and all the activities were not drawn, making the diagram easier to read.
   For example, in figure 5 role *A* participates in activities *X* and *Y*, while role *B* participates only in activity *Y*.

2.3 Describing Process Interactions

Although the use of enhanced UML activity diagrams increased communication and agreement among all stakeholders, having a *big picture*, i.e. an overview of all processes and their interactions, was still a problem. For that purpose, one of the stakeholders took the responsibility of developing a *process map*.

The process map should show all ISO–12207 high–level processes and their interactions, i.e. the products they interchanged with each other, usually documents or software.

After some weeks of work, the final result was an A0 size sheet with dozens of icons and crossing lines which was difficult to understand. In order to make the process map more usable, different views of it were developed. Each view was focused on only one of the processes and their interactions, thus simplifying the initial process map.

Anyway, since the processes in the process map were the ISO–12207 high level processes, i.e. development, operation, maintenance, management, configuration management, etc. (see [9]) for more details), it was still very difficult to know what was the interaction between processes in specific situations.

In other words, if a new employee were hired, performing a specific task taking into account all process interactions would make necessary to read several documents carefully and deduced the implied interactions. Something that would probably take too much time.

In order to avoid these problems, we consider the use of *scenarios* [11], i.e. descriptions of interactions in a given situation. For example, *what happens when a customer applies for a change and the requirements document is not baselined yet?*, *what if the requirement document is already baselined?*, *what are the roles implied?*.

After considering different scenario description techniques (including different approaches for use case descriptions like [2] or [6]), we eventually chose Leite's scenarios because of their simplicity and expressiveness (see [11] for a detailed description).

We introduced some adaptations to the original notation. For example, we dropped the *exceptions* section after some initial descriptions because it seemed unnecessary for our purposes. We added a *variations* section in the scenario description in order to describe possible alternatives to process enactment. We also introduced substeps into conditional blocks in order to ease reading. As proposed by Leite, glossary items in the scenario text are underlined and references to other scenarios appear in upper case. See figure 6 for an example.

3 Introducing Variability in the Common RE Process

A high–level model of the RE process was developed and agreed among all stakeholders (see figures 7 and 8). In this RE process, three main subprocesses were identified, namely requirements development, requirements negotiation and requirements management. The responsibilities of each process the following:

- Requirements Development: this is the most important subprocess and it is responsible for the elicitation, analysis, verification and validation of requirements. It is composed of four activities forming what we call the *requirements pipeline* whose responsibilities are:
  - Requirements Elicitation: this activity is responsible of eliciting requirements from customers and users and producing a draft version of requirements. It is the most complex one due to the needed human interaction. The usual techniques are interviews, meetings, observation, documentation analysis, etc.
  - Requirements Analysis: this activity is responsible for analyzing elicited requirements in order to identify conflicts. If conflicts are identified, they must be solved by the Requirements Negotiation subprocess. The usual technique is requirements modelling.

Title: Register a new Customer Request before requirements are baselined.
Goal: Make requirements according to customers needs.
Context: The project has started and the Requirements Document is not baselined yet.
Resources: Requirements Document, Requirements Management Tool, Configuration Management Tool.
Roles: Customer, Requirements Engineer, Project Manager, Configuration Management Manager, Software Quality Assurance Group.

Steps:

1. A Customer informs of a new Customer Request.
2. The Requirements Engineer registers the new request using a Requirements Management Tool.
3. The Requirements Engineer performs an Impact Analysis of the Customer Request.
4. The Requirements Engineer informs the Project Manager of the results of the Impact Analysis of the Customer Request.
5. Depending on the Impact Analysis, the Project Manager makes the decision of accepting the Customer Request, rejecting the Customer Request or Organizing a Meeting of the Change Control Board.
6. If the Customer Request is eventually accepted, then
   6.1 The Customer Request is incorporated into the Requirements Document as a new Requirement.
   6.2 The Configuration Management Manager manages the new version of the Requirements Document.
   6.3 The Software Quality Assurance Group performs Requirements Verification on the new Requirement.
7. If the Customer Request is eventually rejected, then
   7.1 The Customer Request is incorporated into the Requirements Document as a Rejected Change Request.
   7.2 The Configuration Management Manager manages the new version of the Requirements Document.

Variations:

1. A Customer may inform of a new request by phone (to a previously specified contact person in our company), by fax, by email (to a contact person) or personally.

2. Depending on the maturity of the Requirements Engineering process at the company developing the project, the Requirements Management Tool can be a spreadsheet tool like Excel or an actual Requirements Management Tool like DOORS or Requisite Pro.

3. Depending on the facilities provided by the Requirements Management Tool, the Requirements Engineer can perform the Impact Analysis manually or using the Requirements Management Tool. In both cases, a previous Traceability policy has to be considered in the Software Development Plan.

4. Depending on the company policy (or the project policy), if the impact of the Customer Request is considered to be non significant by the Requirements Engineer, steps 4 and 5 can be skipped, so the Project Manager is not directly informed of the Customer Request.

6.2/7.2 Depending on the Configuration Management Plan of the project, these steps can be performed automatically without the intervention of the Configuration Management Manager.

Fig. 6. Scenario Example

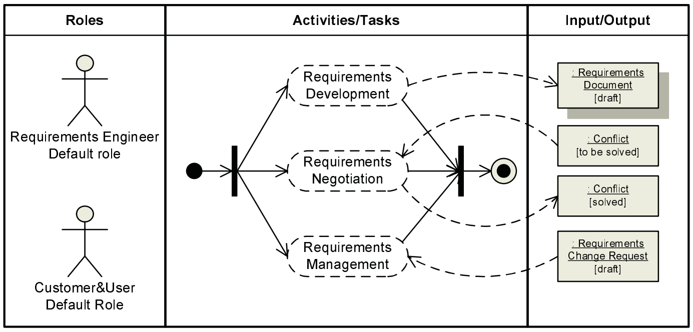
Roles
Activities/Tasks
Input/Output
Requirements Engineer Default role
Customer&User Default Role
Requirements Development
Requirements Negotiation
Requirements Management
: Requirements Document [draft]
: Conflict [to be solved]
: Conflict [solved]
: Requirements Change Request [draft]

Fig. 7. High–level Requirements Engineering Process

- Requirements Verification: this activity is responsible for verifying analyzed requirements in order to detect defects. If defects are found, they must be solved by the requirements writers in the Requirements Elicitation activity. The usual techniques are checklists, formal reviews, inspections, etc.
- Requirements Validation: this activity is responsible for validating verified requirements, thus confirming that they are consistent with the intentions of customer and users. As in the Requirements Verification activity, if conflicts are found, they must be solved by the Requirements Negotiation subprocess. The usual technique is user interface prototyping.

– Requirements Negotiation: this subprocess is responsible for solving all conflicts identified during the Requirements Development subprocess. The solved conflicts are fed back into the Requirements Elicitation activity, the head of the requirements pipeline.

– Requirements Management: this subprocess is responsible for the management of the Requirements Engineering process. Their main responsibilities are requirements change request management and traceability.

Apart from the variations section of process scenarios like the one in figure 6, which allows the introduction of a certain amount of variability in the RE process, once a high–level model of the RE process was developed and agreed among all stakeholders (see figures 7 and 8), it was needed to include different needs from different companies in Telvent. Following a product line approach [1], we identified different *features* which could be necessary in order to tailor the core RE processes for the different companies in Telvent.

3.1 Elicitation Techniques Variability

One of the first features identified as variants were elicitation techniques, requirements documents templates and (single) requirements templates. Some companies were used to apply interviews as the only elicitation technique while others used questionnaires, group meetings or even video–conference for widely distributed projects.

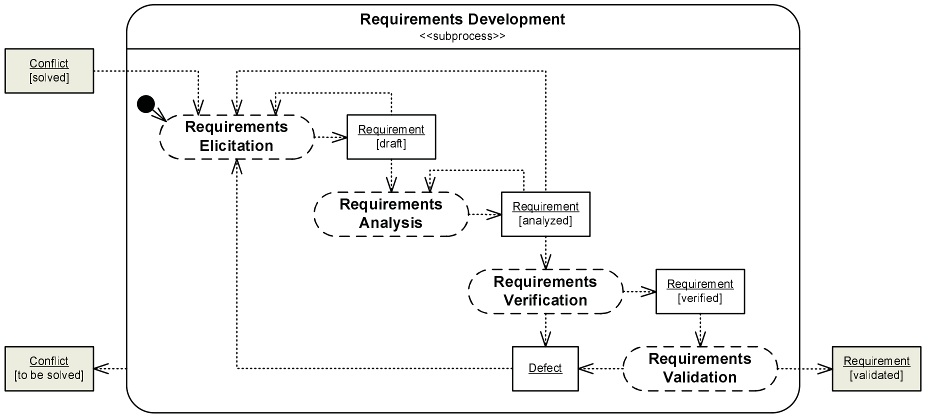

Fig. 8. Requirements Development Subprocess

For requirements analysis, some companies preferred structured techniques like entity-relationship diagrams and data flow diagrams while other preferred object-oriented techniques.

In order to express these possible variants we adopted stereotyped UML class diagrams similar to the notation proposed in [14]. For example, the hierarchy of elicitation techniques in figure 9 is presented as a feature of the requirements elicitation activity. When a company has to tailor the requirements elicitation activity, some of the elicitation techniques must be chosen. In this case, an important issue about feature selection was the selection criteria. Some heuristics were also developed in order to help project managers chose the right feature, i.e. the right elicitation technique. Those heuristics were based on the previous work by Davis and Hickey [4].

3.2 Roles Variability

Another configurable *feature* of the RE process was the roles of the different Telvent personnel performing the RE process. Role names and responsibilities had to be adapted to the different backgrounds of the Telvent companies[3].

There was also the possibility of determining for every project if a role was played simultaneously by another person playing another role or by a specific person playing one role only.

For example, in figure 10 the Requirements Verifier role can be played by the person playing the Requirements Engineer role, by the person playing the SQA role o by a person playing the Requirements Verifier role specifically.

3.3 Documentation Variability

Another important aspect of the RE process variability was the documentation to be delivered. Depending on the customer, sometimes the requirements document should be

[3] Actually, role names are in the way to be universally adopted in all Telvent companies, although there is an adaptative period of 2 years. Role names were one of the concepts which generated more discussion when building the glossary of software development terms (see section 2.1).

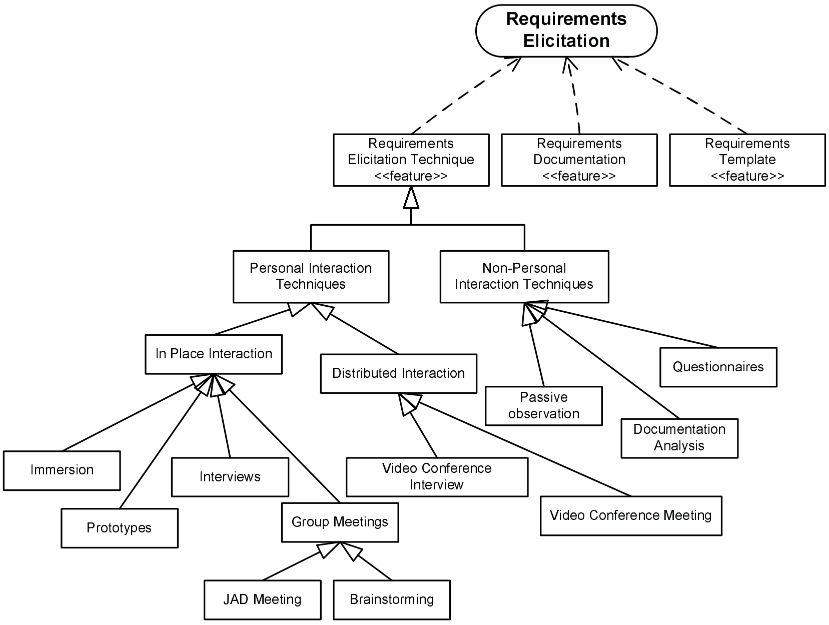

Fig. 9. Requirements Elicitation Techniques Feature Hierarchy

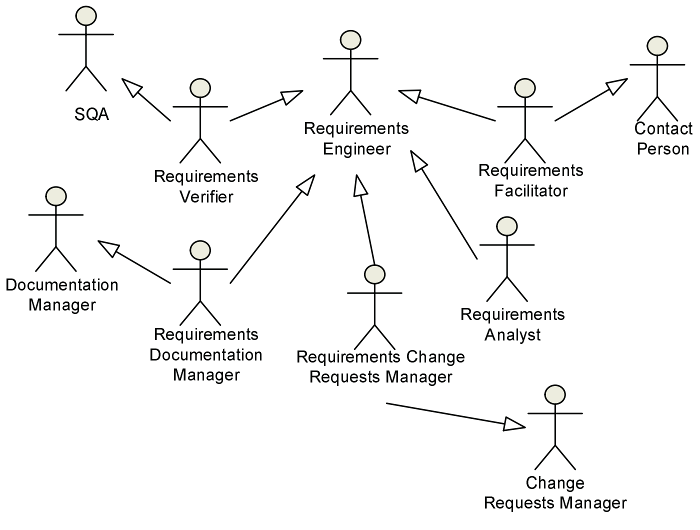

Fig. 10. Requirements Engineering Role Hierarchy

written using IEEE–830 [7], MÉTRICA [3] (Spanish Government Methodology, similar to SSADM) or other official standards. Keeping the same core RE process, concrete products were tailored in order to be compliant with standards like those previously mentioned.

The way how requirements were written, i.e. a requirements template [5], was also considered as a possible variable feature of the RE process.

4 Definition of a RE Process for Systems Families Development

One of the most important middle–term goals of Telvent at the beginning of the definition of common software development processes was to adopt a product line approach [1] in some of their companies. From the experience gained in the definition of the "process family" for RE, some results were extrapolated into an RE process family for product line development. An overview of the new RE core process for product line development is shown in figure 11, where key activities are highlighted and the Requirements Development subprocess has been flattened for the sake of simplicity.

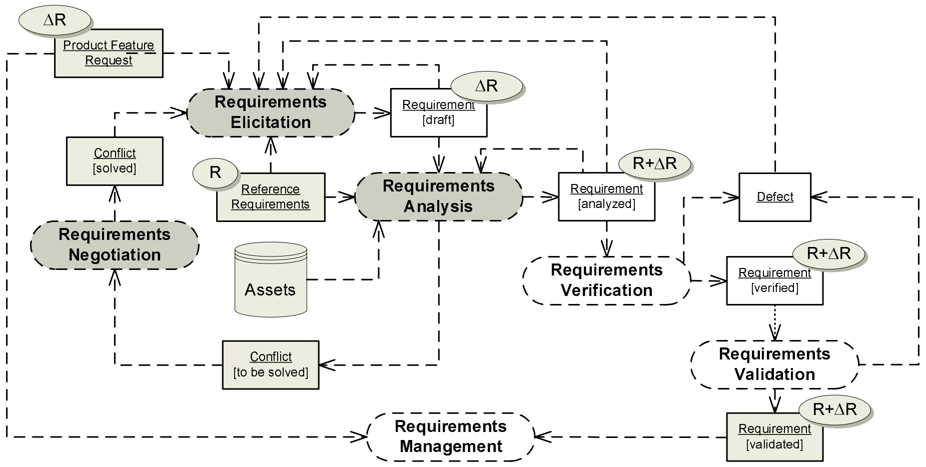

Fig. 11. RE Process for Systems Families Development

The main difference between the single–product RE process and the product line oriented RE process is the introduction of the Reference Requirements and the set of reusable assets as inputs of the Requirements Elicitation, Analysis and Negotiation activities.

In this new RE process, Requirements Elicitation is responsible for identifying new product features, but taking into account reference requirements, so the degree of freedom is substantially reduced.

Requirements Analysis is responsible not only for the identification of conflicts in new requirements (delta requirements), but also for the identification of conflicts between new requirements and reference requirements, which can be a much harder work than for a project–oriented RE process. What is more, in order to identify conflicts, assets in the product line must be taken into account, making the activity more complex.

On the other hand, solving conflicts, i.e. Requirements Negotiation, is now a critical activity. Conflicts must be solved not only from a logical point of view, but also taking into consideration economical and market issues.

5 Conclusions

The main lesson we have learned after our experience is that defining a common process (software development process, RE process or other kind of process) for different organizations with different needs is a complex task that can be seen as a RE problem where RE techniques can be successfully applied. From an abstract point of view, our task was to develop a product – a RE process embedded in a system/software development process – that must satisfy different, sometimes contradictory, stakeholders' needs. If we had taken the reflexive RE approach from the beginning, instead of trying to impose a common RE process in a top–down fashion, we would have saved time and effort.

Applying the same solutions for the definition of an RE process family, we have started the definition of a RE process family for product line development, part of which has been briefly presented in this paper.

Another interesting lesson is that, sometimes, a product families approach can help in requirements negotiation. If an agreement is not possible, maybe we can develop a small product family satisfying incompatible stakeholders' needs.

References

1. P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison–Wesley, 2002.
2. A. Cockburn. *Writing Effective Use Cases*. Addison–Wesley, 2001.
3. CSJ. Metodología de Planificación, Desarrollo y Mantenimiento de Sistemas de Información. MÉTRICA Versión 3 (Borrador). Borrador, Consejo Superior de Informática, 2000. Disponible en `http://www.map.es/csi/pg5m42.htm`.
4. A. Davis and A. Hickey. Learn how to select the "right" requirements elicitation technique. In *Tutorial at RE'02*, 2002.
5. A. Durán, B. Bernárdez, A. Ruiz, and M. Toro. A Requirements Elicitation Approach Based in Templates and Patterns. In *WER'99 Proceedings*, Buenos Aires, 1999.
6. A. Durán, A. Ruiz, R. Corchuelo, and M. Toro. Supporting Requirements Verification Using XSLT. In *Proceedings of the IEEE Joint International Requirements Engineering Conference*. IEEE CS Press, 2002.
7. IEEE. Recommended Practice for Software Requirements Specifications. IEEE/ANSI Standard 830–1998.
8. IEEE. IEEE Standard Glossary of Software Engineering Terminology. IEEE Standard 610.12–1990, Institute of Electrical and Electronics Engineers, 1990.
9. ISO/IEC. Information Technology–Software Life Cycle Processes. International Standard 12207 : 1995, International Organization for Standarazition, 1995.
10. G. Kontoya and I. Sommerville. *Requirements Engineering: Processes and Techniques*. Wiley, 1997.
11. J. C. S. P. Leite, H. Hadad, J. Doorn, and G. Kaplan. A Scenario Construction Process. *Requirements Engineering Journal*, 5(1), 2000.
12. OMG. Unified Modeling Language, v1.4. Technical report, September 2001.
13. I. Sommerville and P. Sawyer. *Requirements Engineering: A Good Practice Guide*. Wiley, 1997.
14. Jilles van Gurp, Jan Bosch, and Mikael Svahnberg. On the notion of variability in software product lines. In *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01)*, 2001.
15. E. Yourdon. *Modern Structured Analysis*. Prentice–Hall, 1989.