Determining Criteria for Selecting Software Components: Lessons Learned

Juan Pablo Carvallo, *Etapatelecom*
Xavier Franch and Carme Quer, *Universitat Politècnica de Catalunya*

Component selection relies on the suitability and completeness of the criteria used for evaluation. Experiences from determining criteria for several industrial projects provide important lessons.

Software component selection[1] is growing in importance. Its success relies on correctly assessing the candidate components' quality. For a particular project, you can assess quality by identifying and analyzing the criteria that affect it.

For example, imagine that a company needs to acquire a workflow system. To make an informed decision, the organization wants to know the quality of the products available in the market. From the company's viewpoint, a workflow system's quality depends on many factors—for example, services offered (such as mechanisms to notify users about events), nonfunctional characteristics (such as security policies implemented), and deployment characteristics (such as licensing schemes supported).

For a particular selection process, you can organize *selection criteria* into a *criteria catalog*. A CC is built for a *scope*, which can be either a *domain* (workflow systems, mail servers, antivirus tools, and so on) or a *category* of domains (communication infrastructure, collaboration software, and so on). Structurally, a CC arranges selection criteria in a hierarchical tree-like structure. The higher-level selection criteria serve to classify more concrete selection criteria, usually allowing some overlap. They also serve to leverage the CC. The lowest-level selection criteria are observable and measurable properties that exhibit the target scope's components. In our workflow example, some lowest-level selection criteria are `User Notification Mechanisms`, `Support Rules`, `Security Transfer Protocols`, `Encryption Algorithms`, and `Licensing Schemes Supported`. The first two criteria are children of a higher-level one, product `Suitability`, while the third and fourth are linked to `Security`. Table 1 shows some selection criteria from a CC for workflow systems.

It's worth examining how a CC's construction and use fits in the overall decision process as Anthony Finkelstein, George Spanoudakis, and Mark Ryan defined it in their seminal article.[1] According to them, software selection comprises four activities:

1. acquiring and specifying the requirements,
2. understanding the available packages,
3. assessing package compatibility with regard

Table 1

Some selection criteria from a criteria catalog for workflow systems, with the evaluation and comparison of two products

Selection criteria	Requirements	Product 1	Product 2	Comparison
Suitability				
User Notification Mechanisms	As much as possible	Email Message (screen)	Email Task page Message (screen)	Product 2 better
Support Rules	At least by role and by user	By role By group By department By user	By role By user	Both products satisfy requirement
Security				
Security Transfer Protocols	S-HTTP, SSL	S-HTTP	S-HTTP, SSL	Product 1 fails
Encryption Algorithms	DES	RSA, CAST	RSA, CAST, DES, 3DES	Product 1 fails
Business				
Licensing Schemes Supported	Per client	Per concurrent user Per user groups	Per concurrent user	Both products fail

to the requirements, and
4. selecting the "best" available package.

A CC is useful in the evaluation of the components (activity 2 itself), in the matching of those components with regard to the requirements (activity 3), and in making the final decision (activity 4). This process combines quantitative reasoning (for example, using a multicriteria decision-making technique) with qualitative arguments (typically, managerial decisions). Also, some negotiation might be necessary when no product meets some requirements. More details about this process appear elsewhere.[2]

We therefore propose CCs as the best way to deal with selection criteria for software component selection. However, building CCs can be cumbersome and error prone. Here, we present lessons we've learned that ameliorate these problems. These lessons come from various projects in different domains and contexts.[3] We participated in seven quality-related projects that resulted in 10 CCs consisting of hundreds of selection criteria each (120 for the smallest and 510 for the biggest). In addition, we've built many CCs in an academic setting.

Lesson 1: Adopt a balanced CC

Most current proposals for building CCs use a *mixed approach*. In this approach, you start with a basic, solid catalog of selection criteria and extend and adapt the criteria to the target scope's needs. Such an approach's success relies on the *base catalog's* quality.

Problem

Literally hundreds of proposed selection CCs exist, and new ones appear continually, making it hard to choose one over another and even harder to reconcile them. Constructing a suitable base catalog is difficult: it can't be too narrow (useless) or too specific (difficult to adapt to particular projects).

Solution

We propose using a base catalog containing only high-level CCs, applicable to virtually all the quality scopes. We define it as an extension of the catalog introduced in the ISO/IEC 9126-1 standard, which is presented as a general-purpose *quality model*. (More details and discussion of this standard appear elsewhere.[2]) The original standard defines a CC of two levels, composed of six high-level selection criteria (called *characteristics*), divided into 27 second-level selection criteria (*subcharacteristics*). Our base catalog adds 60 new subcharacteristics: 34 in the third level and 26 in a fourth one.

Selection criteria: Characteristics/subcharacteristics and attributes				Description
Functionality				ISO/IEC 9126-1 Capability of the software product to provide functions that meet stated and implied needs when the software is used under specified conditions
	Suitability			ISO/IEC 9126-1 Capability of the software product to provide an appropriate set of functions for specified tasks and user objectives
	Accuracy			ISO/IEC 9126-1 Capability of the software product to provide the right or agreed results or effects with the needed degree of precision
		Verifiableness		Provision of the software product of resources to allow the tracking and verification of its right or agreed results or effects
			History Control	Capability of the software product to provide a history of the changes on the data managed
			Data Versioning	Capability of the software product to store/provide versions of the data managed
			Logging Capabilities	Provision of the software product of logging mechanisms
		Effectiveness		Provision of the software product of mechanisms to determine the amount of right or agreed results or effects
			Self Tests Results	Provision of the software product of mechanisms to perform direct tests of the right or agreed results or effects
			Published Tests	Capability according to third-party reports of right or agreed results or effects of the software product
	Interoperability			ISO/IEC 9126-1 Capability of the software product to interact with one or more specified systems
		Direct Interoperability		Capability of the software product to directly interact with specified systems
			By Means of Protocols	Capability of the software product to directly interact with other systems by means of supported protocols
			By Means of APIs	Capability of the software product to directly interact with other systems by means API libraries provided
		Indirect Interoperability		Capability of the software product to interact with other systems by means of indirect mechanisms
	Security			ISO/IEC 9126-1 Capability of the software product to protect information and data so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them
		Application Security		Capability of the software product to provide mechanisms to prevent the accidental or deliberate unauthorized access to system functionality
			Provided by the Application	Provision of the software product of mechanisms to prevent the accidental or deliberate unauthorized access to the product functionality
			Provided by Third Parties	Provision of third-party organizations of mechanisms to prevent the accidental or deliberate unauthorized access to the product functionality
		Data Security		Capability of the software product to provide mechanisms to prevent the accidental or deliberate unauthorized access to the data managed by the software product
			Stored Data	Provision of the software product of mechanisms to prevent the unauthorized access to the data stored by the product
			Transmitted Data	Provision of the software product of mechanisms to prevent the unauthorized access to the data transmitted by the product
	Functionality Compliance			ISO/IEC 9126-1 Capability of the software product to adhere to standards, conventions, or regulations in laws and similar prescriptions relating to functionality

Figure 1. An excerpt of our base catalog: decomposition of the ISO 9126-1 `Functionality` **characteristic.**

For example, figure 1 shows the part of the base catalog corresponding to the `Functionality` characteristic. The standard decomposes `Functionality` into five subcharacteristics. Our extension decomposes three of them into 17 new subcharacteristics.

Observations

We've used the original ISO catalog in all our projects, with more than satisfactory results (we've used 100 percent of the characteristics and approximately 80 percent of the subcharacteristics in all the cases). The only subcharacteristics that we haven't always used are the six referring to compliance that decompose each characteristic of the ISO catalog (such as `Functionality Compliance`).

Forty-two percent of the new selection criteria of our base catalog appear in all the CCs we've built. We decided to maintain the rest of the selection criteria because they're a kind of checklist for not forgetting any potentially relevant aspect. An example of such a subcharacteristic is `Functionality/Security/Data Security/Transmitted Data`, which isn't of interest for scopes that don't require data transmission.

The `Suitability` subcharacteristic isn't decomposed in our base catalog. This is because we've observed that selection criteria that decompose this subcharacteristic aren't usually reused in other scopes except for very closely related ones.

We don't intend the base catalog to be static. In fact, its current form is the result of an evolution that occurred during our first projects, and it will likely grow with new selection criteria.

Some methodological support for extending the base catalog to get the selection criteria appears elsewhere.[2]

Consequences

Consider our base catalog as your CC's starting point. Look at its subcharacteristics and decide which ones make sense for the target scope and are of interest in your project's context. Decompose these subcharacteristics until you obtain the CC, which generally occurs when you obtain selection criteria that correspond to observable and measurable properties of the scope (called *attributes* in the ISO standard).

Lesson 2: Recognize the nontechnical selection criteria's importance

Nontechnical selection criteria such as administrative, economic, or political criteria are often significant in software selection, sometimes even more important than technical selection criteria.

Problem

The catalogs available in most proposals don't include nontechnical selection criteria. Specifically, this is true of ISO/IEC 9126-1. Not considering these criteria compromises the undertaken activity's success. On the other hand, considering them apart from technical selection criteria requires managing two different sets of criteria that are actually similar in nature.

Solution

We propose enlarging the base catalog with nontechnical selection criteria structured in the same way as technical ones. We provide a hierarchy of 141 subcharacteristics and attributes that decompose three high-level characteristics (`Supplier`, `Business`, and `Product`). As a result, we obtain a comprehensive base catalog that we call the *extended ISO catalog*, which integrates technical and nontechnical selection criteria. Figure 2 shows the two highest levels of the nontechnical part of the catalog, including a decomposition of the `Supplier/Reputation` subcharacteristic and a few metrics.

Observations

Nontechnical selection criteria have been important in most of our projects. For example, a university rejected a technically adequate candidate in a requirements management tool selection project because of the poor evaluation of the selection criteria belonging to the `Supplier/Support` subcharacteristic.

Most nontechnical selection criteria don't depend on the domain. So, we've included many of them in the CC, more than technical selection criteria. We even defined nontechnical selection criteria at the attribute level, including metrics. As a result, when we used this nontechnical part in our projects, we pruned those selection criteria that didn't apply to the project at hand.

The extended ISO catalog's technical and nontechnical parts overlap somewhat. For instance, the `Time in Market` attribute of the `History` nontechnical subcharacteristic also falls under the `Maturity` technical subcharacteristic. Also, there are synergies and conflicts among technical and nontechnical attributes; for example, the `Parameterization & Customization` attributes influence `Operability` ones.

Consequences

Consider the nontechnical part of the extended ISO catalog that's in the CC under construction. Remove those selection criteria that don't apply, and eventually modify the definition of some criteria and even include others that aren't part of the catalog.

Lesson 3: Define precisely your selection framework

ISO/IEC 9126-1 is defined on a *quality framework* that embraces many concepts.

Consider the nontechnical part of the extended ISO catalog that's in the criteria catalog under construction.

Selection criteria: Characteristics/subcharacteristics and attributes				Description	Metrics
Supplier				Characteristics of the supplier that can influence the quality of the software product	
	Organizational Structure			Description of the organizational structure of the supplier company	
	Positioning and Strength			Description of the position and orientation of the supplier company in the market	
	Reputation			Recognition of the capability of the supplier to perform similar projects based on past experiences and certifications	
		Supplier Company Existence		Years of the supplier company in the market from its foundation	Number of years: Integer
		Quality Process Certification		Certifications of the quality of the process followed by the supplier company given by recognized certification authorities	Qualification: (Good, Correct, Suitable) Formula for calculating the value from the values of the subattributes
			CMM Level	Capability Maturity Model Level granted to the supplier company	CMM Level: Integer (1 … 5)
			ISO 9000	ISO 9000 Certificate granted to the supplier company	ISO 9000: Boolean
			Other Certificates	Other quality process certificates	List of (Certificate, Level) Certificate: (Spice, SixSigma, …) Level: String
		Client Recommendations		References and recommendations of the supplier company that other clients have given	List of (Client, Comments) Client: String, Comments: List of String
	Services Offered			Description of the services offered by the supplier	
	Support			Description of the support mechanisms offered by the supplier company	
Business				Characteristics of the contract among the supplier and the client that can influence the quality of the software product	
	Licensing Schema			Description of the product-licensing options	
	Ownership			Description of the aspects in relation to the intellectual property rights	
	Guarantees			Detail of the guarantees provided over the product	
	Licensing Costs			Description of the costs components and total cost of ownership for the different licensing options available	
	Platform Cost			Estimation of the cost for the required production platform	
	Implementation Cost			Estimation of implementation costs based on similar past experiences	
	Network Cost			Estimation of additional costs for network operation	
Product				Characteristics of the commercial aspects of the software product that can influence its quality	
	History			Evolution of the product since it has been offered to the clients	
	Deliverables			Detail of the out-of-the-box and expected postimplementation deliverables	
	Parameterization & Customization			Description of the initial effort required for the product to operate	

Figure 2. An excerpt of the nontechnical part of the extended ISO catalog.

Problem

Even when you use a widespread quality standard such as ISO/IEC 9126-1, you might find some aspects that aren't precise enough. Can characteristics and subcharacteristics be measured? Are hierarchies of subcharacteristics and attributes allowed? Is overlapping allowed? Not having precise answers to these questions makes it difficult to be consistent over time.

Solution

Starting from the ISO/IEC 9126-1 standard document, we've clarified and defined precisely its quality framework with a UML class diagram (see figure 3) and an associated glossary of terms, detecting and correcting some problems. This approach is similar to the one that Barbara Kitchenham, Robert Hughes, and Stephen Linkman proposed.[4] The answers to our previous questions become clear when we analyze the class diagram: only attributes can be measured, subcharacteristics and attributes can be arranged into hierarchies, and overlapping is possible.

Observations

Because the domain experts in our projects

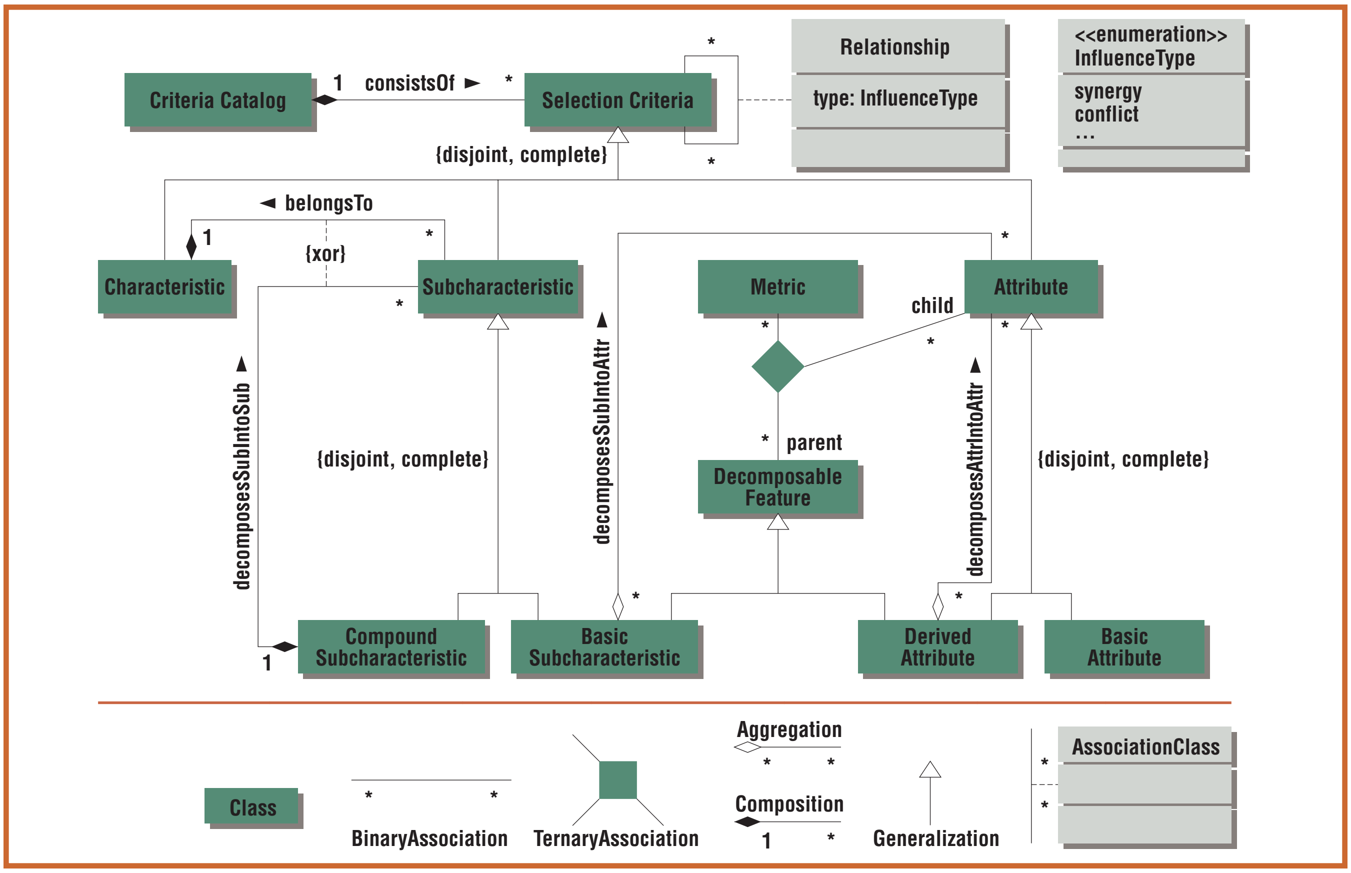

Figure 3. A UML class diagram for the ISO/IEC-9126-1-based quality framework.

understood the class diagram well, we could share a common reference model. The glossary of terms was also a success factor for communication.

The UML model has other implications. For instance, you can't decompose a subcharacteristic into other subcharacteristics and attributes at the same time. Also, because we don't include class multiplicities, meaning that the number of instances of classes aren't restricted, we could eventually add new characteristics if some particular project requires them.

We can enrich the framework with new concepts and integrate them into the original proposal in a clearly stated form. For instance, we've incorporated the concept of relationships among selection criteria such as synergy or conflict (see the classes in gray in figure 3). These relationships help us understand the addressed scope. Also, we've defined precisely the concepts involved in metrics definition (types of metrics, scale, measurement units, measure instruments, and so on), although the figure doesn't show this.

An added benefit of the class diagram is that it served as the basis for developing DesCOTS (description, evaluation, and selection of COTS components), a tool for constructing CCs.[5] (You can download the tool at www.lsi.upc.edu/~gessi/DesCOTS.)

Consequences

In a new project, use the conceptual model and glossary to know which concepts are relevant for communication among domain experts and technicians and to know what you can and can't do. Define your CCs consistently with respect to this conceptual model. Use the associated tool support to populate the database that stores the CC.

Lesson 4: Consider the CC's final purpose

Depending on a CC's life span, we can classify it as *nonreusable* or *reusable* (see figure 4). Nonreusable CCs appear only in specific projects, and their existence is bound to them. Reusable CCs are persistent for a certain scope; you can reuse them in many projects.

Problem

Usually, approaches for building CCs don't

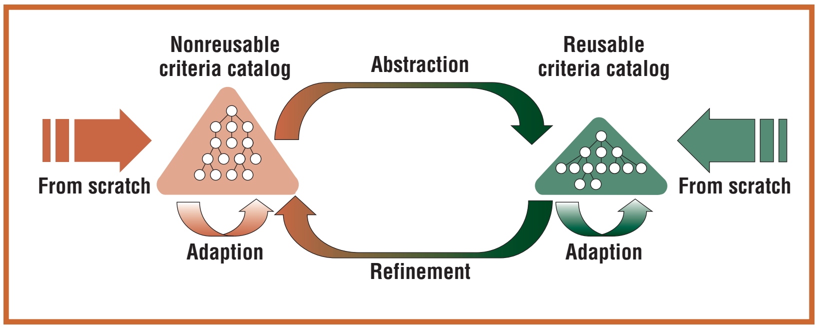

Figure 4. Constructing nonreusable and reusable CCs.

consider this classification as relevant. So, nonreusable CCs might include aspects that aren't of interest for the scope or might not refine enough aspects that are of interest. Also, reusable CCs might be too detailed or might not address aspects that are of general interest for the scope.

Solution

When building nonreusable CCs from scratch, we intertwine requirements engineering and CC construction.[6] The initial requirements identify selection criteria of interest; refining these criteria might generate more concrete requirements. If a reusable CC for the scope exists, CC construction consists basically of bridging the gap between that CC and the requirements (*refinement*).

We build reusable CCs in two different ways. We can manipulate a deployed nonreusable CC to create an associated reusable model, removing the project-specific parts (*abstraction*). Otherwise, we can build a reusable CC from scratch. In this case, we base its construction mainly on the services offered by the software components on the market. The criteria in the catalog should be present in most, if not all, of these components; however, we can include concepts that aren't yet available.

Observations

For coarse-grained scopes, such as some software domain in the category of business applications, you'll need to identify, organize, and maintain dozens or even hundreds of selection criteria. In these cases, it's crucial to determine in advance the CC's objective and which parts of it are of interest.

The construction of nonreusable CCs finishes once all the requirements have become measurable.

If you build nonreusable CCs by refinement, it might be worth considering all the criteria, not just the selection criteria bound to the requirements of interest. Analyzing all the criteria might induce the discovery of new requirements that you hadn't considered before.

You can build both kinds of CCs starting from an existing model of the same kind (*adaptation*). For nonreusable CCs, this process will likely compromise reusability. However, by analyzing the adapted CC, you might be able to identify very particular requirements (for example, the kind of encryption algorithm to use and the required encryption key's number of bits).

Consequences

In a new project, decide first which kind of CC you need. In any case, check whether a CC for the same domain is available. Depending on this search's result, determine the most appropriate construction process, according to figure 4. Once you've finished the construction of a nonreusable CC, decide whether abstracting it into a reusable CC is appropriate.

Lesson 5: Organize software scopes hierarchically

You can construct a CC for hundreds of different scopes. The frontiers among these scopes aren't always clear and change continually as new organizational needs arise or technology evolves.

Problem

To construct a CC, you must identify the target scope, which isn't easy for the reasons we just mentioned.

Solution

We propose organizing the scopes hierarchically. We use *classification attributes*[7] as classification criteria; for each value that a classification attribute may take, we create a child.

The root in our taxonomy (see an excerpt of it in figure 5) stands for the universal context `Software Applications`. It has a classification attribute bound, `Mission of Application`.

This root's children are in turn roots of large categories of software applications. Two of these categories correspond to the `Business Applications` taxonomy (for example, enterprise-resource-planning systems and supply-chain-management systems) and `Software Development Tools` taxonomy (integrated de-

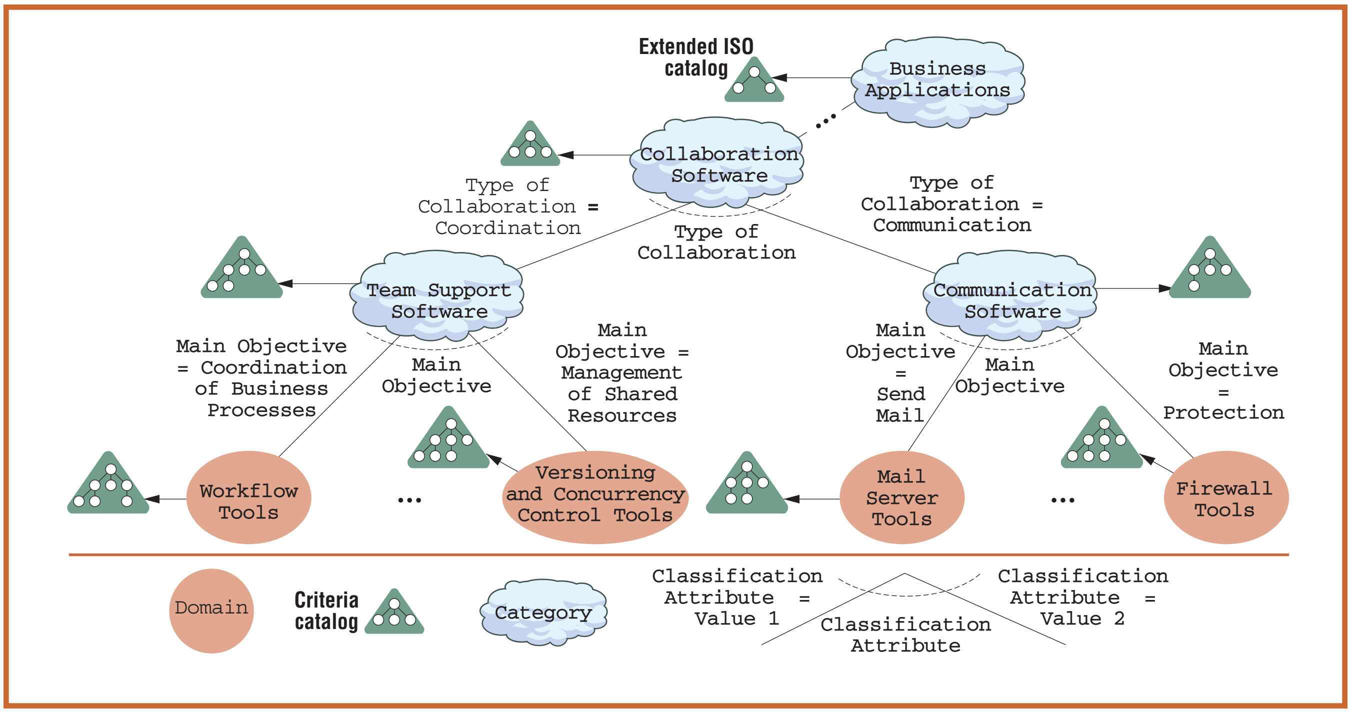
Extended ISO catalog
Business Applications
Collaboration Software
Type of Collaboration = Coordination
Type of Collaboration
Type of Collaboration = Communication
Team Support Software
Communication Software
Main Objective = Coordination of Business Processes
Main Objective
Main Objective = Management of Shared Resources
Main Objective = Send Mail
Main Objective
Main Objective = Protection
Workflow Tools
...
Versioning and Concurrency Control Tools
Mail Server Tools
...
Firewall Tools
Domain
Criteria catalog
Category
Classification Attribute = Value 1
Classification Attribute
Classification Attribute = Value 2

Figure 5. A taxonomy corresponding to the `Collaboration Software` **category, which is a subcategory of the** `Business Applications` **taxonomy.**

velopment environments, requirements management tools, and so on). For these taxonomies, the `Mission of Application` classification attribute has the values `Support to Organization Business Processes` and `Support to Software Development`, respectively.

Leaves of the taxonomy represent software domains, such as `Mail Server Tools` or `Workflow Tools`.

We group similar domains into categories. For instance, the `Team Support Software` category has a classification attribute `Main Objective` that might take, among others, the values `Coordination of Business Processes` and `Management of Shared Resources` that yield nodes for the `Workflow Tools` and `Versioning and Concurrency Control Tools` domains, respectively. We can also group categories to form a multilevel taxonomy.

Observations

Software domain taxonomies are usually proposed and maintained by IT consultant companies, commercial Web sites, academic organizations, and so on. We've used one such taxonomy as a source.[7] We took a taxonomy (`Business Applications`) made by one major IT consultant company with 60 domains and 18 categories, with an average width of 4.33. After goal-oriented analysis, we arranged it into a new taxonomy of 72 domains and 48 categories, with an average width of 2.78 (so, domain identification was more directed than before).

We don't recommend building a whole taxonomy from the beginning for each project unless you have good reasons (for example, your organization already has a thorough knowledge of the addressed market segment). Instead, make it grow incrementally with each new project.

We've used the taxonomy to enhance selection criteria reuse by binding reusable CCs to its nodes (see figure 5). Each node inherits its parent's CC and adds those selection criteria that are common to all its descendants' scopes. Taxonomies with reusable CCs are a good infrastructure to implement classic reuse frameworks such as the experience factory.[8]

Consequences

When you start a project, identify the domain involved. To do so, you can use the classification attributes to traverse the hierarchy downward until you reach the domain. In this case, you use the reusable CC bound to the node as the starting point. If the search fails, you must create a node for this domain, possibly with some path starting at the node that might be considered its closest ancestor. For all the new nodes, create reusable CCs before, during, or after the project.

More important than the concrete artifacts we've presented (the CCs, the taxonomy, and so on) are the underlying ideas.

Some scenarios of use

Here are several scenarios that might benefit from our lessons learned. They are based on our experiences but have been abstracted to obtain a setting of interest to more people.

Call-for-tenders case

The National Finance Department wants to acquire document management software. It knows exactly what type of services it needs. Owing to regulations, the bidding process must be highly transparent. The department requests its software lab to write a call-for-tenders document expressing the requirements precisely, such that evaluation of candidate solutions can be as objective as possible. Because these regulations are recent, it's the first time that the lab has had to write such a document.

One lab employee is an IEEE member who has recently read an article on lessons learned about selection criteria. She convinces her boss to follow this approach. Figure 6a shows this process. They end up with a highly structured call for tenders (following the CC's layout) listing the measurable requirements for which the candidate suppliers must provide information.

Product selection case

Our university's software engineering department wants to acquire a meeting scheduler tool. The department head has some idea of what services the tool must offer, but she would like to know which other aspects of MSTs could serve the department's purposes. She knows that we've participated in some software selection projects and therefore asks us to participate as consultants in the selection. We accept.

Figure 6b shows the selection process. From our previous experiences, we have an ongoing taxonomy with the scopes identified so far; in particular, we take advantage of projects we've undertaken in domains similar to MST.[3] Furthermore, our DesCOTS tool facilitates our analysis. We produce a comparison of several products and a report on additional services that you can expect from the MST. Finally, we make the constructed CC reusable, and we update our taxonomy, binding this reusable CC to a new node for the MST domain.

Quality consultancy case

The Acme consultant company has a department specializing in consultancy on software component quality. That department maintains its own version of a business applications taxonomy that was referenced in an issue of *IEEE Software*. The EMCA company's software lab asks Acme to participate in the requirements elicitation of a new component-based software system for supporting its supply chain management. Because the decision to implement an SCM system was political, the EMCA Software Department isn't sure about which types of components it needs or which requirements it might state over these components.

Acme accepts the consultancy and follows the process in figure 6c. Acme uses the taxonomy to determine the candidate scopes in which the client organization might be interested. The CCs bound to these scopes serve to inform the client about the type of services and other expected characteristics. Also, Acme summarizes information about budget, licenses, and so on from the evaluation of the products of that scope. The Software Department uses this information to prepare a report for company management describing the different options and their costs.

Information brokerage case

The Component Description Provider company offers characterization of domains through the Internet to people interested in knowing what aspects to consider when looking at some product in a certain domain. It offers the characterizations as CCs and classifies the domains as a taxonomy, in both cases following the guidelines presented in an *IEEE Software* article on CCs.

Recently, this company's employees have been aware of a new domain: tools for making conceptual maps (for example, VisualMind and MindGenius). They have examined the existing taxonomy, discovered the category to which this domain belongs, updated the taxonomy, and created a reusable catalog. Figure 6d summarizes this process. As a result, they have their taxonomy constantly updated, including the latest technologies, which adds value to the company's business.

The concept of selection criteria has similarities with concepts such as persistent software attributes[9] and quality characteristics.[10] While these concepts focus

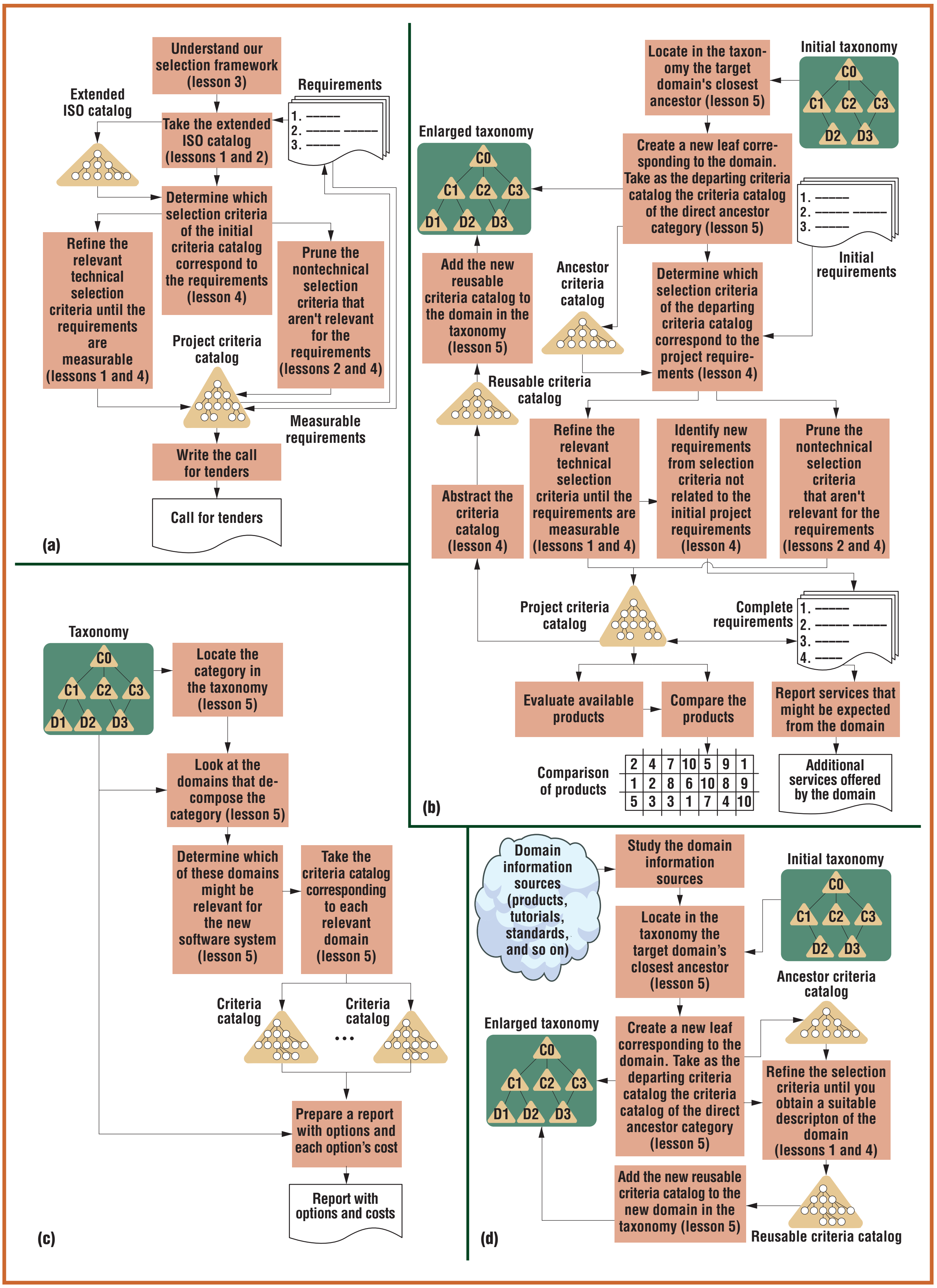
Understand our selection framework (lesson 3)
Extended ISO catalog
Requirements
1. 2. 3.
Take the extended ISO catalog (lessons 1 and 2)
Determine which selection criteria of the initial criteria catalog correspond to the requirements (lesson 4)
Refine the relevant technical selection criteria until the requirements are measurable (lessons 1 and 4)
Prune the nontechnical selection criteria that aren't relevant for the requirements (lessons 2 and 4)
Project criteria catalog
Measurable requirements
Write the call for tenders
Call for tenders
(a)
Locate in the taxonomy the target domain's closest ancestor (lesson 5)
Initial taxonomy
C0 C1 C2 C3 D2 D3
Enlarged taxonomy
C0 C1 C2 C3 D1 D2 D3
Create a new leaf corresponding to the domain. Take as the departing criteria catalog the criteria catalog of the direct ancestor category (lesson 5)
Initial requirements
Ancestor criteria catalog
Add the new reusable criteria catalog to the domain in the taxonomy (lesson 5)
Determine which selection criteria of the departing criteria catalog correspond to the project requirements (lesson 4)
Reusable criteria catalog
Abstract the criteria catalog (lesson 4)
Refine the relevant technical selection criteria until the requirements are measurable (lessons 1 and 4)
Identify new requirements from selection criteria not related to the initial project requirements (lesson 4)
Prune the nontechnical selection criteria that aren't relevant for the requirements (lessons 2 and 4)
Project criteria catalog
Complete requirements
1. 2. 3. 4.
Evaluate available products
Compare the products
Report services that might be expected from the domain
Comparison of products
2 4 7 10 5 9 1
1 2 8 6 10 8 9
5 3 3 1 7 4 10
Additional services offered by the domain
(b)
Taxonomy
C0 C1 C2 C3 D1 D2 D3
Locate the category in the taxonomy (lesson 5)
Look at the domains that decompose the category (lesson 5)
Determine which of these domains might be relevant for the new software system (lesson 5)
Take the criteria catalog corresponding to each relevant domain (lesson 5)
Criteria catalog
Criteria catalog
...
Prepare a report with options and each option's cost
Report with options and costs
(c)
Domain information sources (products, tutorials, standards, and so on)
Study the domain information sources
Initial taxonomy
C0 C1 C2 C3 D2 D3
Locate in the taxonomy the target domain's closest ancestor (lesson 5)
Ancestor criteria catalog
Enlarged taxonomy
C0 C1 C2 C3 D1 D2 D3
Create a new leaf corresponding to the domain. Take as the departing criteria catalog the criteria catalog of the direct ancestor category (lesson 5)
Refine the selection criteria until you obtain a suitable descripton of the domain (lessons 1 and 4)
Add the new reusable criteria catalog to the new domain in the taxonomy (lesson 5)
Reusable criteria catalog
(d)

Figure 6. Four scenarios that benefit from the lessons learned about CCs: (a) call-for-tenders case, (b) product selection case, (c) quality consultancy case, and (d) information brokerage case.

About the Authors

Juan Pablo Carvallo is the manager of the Informatics department of Etapatelecom, a telecommunications company based in Cuenca, Ecuador, and a part-time teacher at Azuay University's Systems Engineering Department. His research interests include requirements engineering, COTS-based system development, and COTS selection, evaluation, and certification. He received his PhD in informatics from the Universitat Politècnica de Catalunya. Contact him at Calle Larga 113 y Av. Huayna Cápac, Edif. Banco Central Cuenca, Ecuador; jpcarvallo@etapatelecom.net.

Xavier Franch is an associate professor at the Universitat Politècnica de Catalunya's Department of Software. His main interests include requirements engineering, COTS-based development, and software quality. He's a member of the IEEE. He received his PhD in informatics from the Universitat Politècnica de Catalunya. Contact him at UPC-Campus Nord, Edif. Omega, despatx 122, Jordi Girona 1-3, 08034 Barcelona, España; franch@lsi.upc.edu.

Carme Quer is an associate professor at the Universitat Politècnica de Catalunya's Department of Software. Her research interests are selection of COTS components, software quality, and component-based software development. She received her PhD in informatics from the Universitat Politècnica de Catalunya. Contact her at UPC-Campus Nord, Edif. Omega, despatx 119, Jordi Girona 1-3, 08034 Barcelona, España; cquer@lsi.upc.edu.

on controlling and monitoring system development and operation for quality assessment purposes, selection criteria are used to describe beforehand the elements desired for the system.

More important than the concrete artifacts we've presented (the CCs, the taxonomy, and so on) are the underlying ideas. In other words, an organization may use its own base catalog or arrange different kinds of taxonomies that better fit its particular needs.

We've been using the presented approach in our own industrial experiences with satisfactory results. Specifically, we've found its highly incremental nature to be very valuable. This feature will let you construct the artifacts we've presented as you need them for a particular project, while supporting easy knowledge transfer from one project to the next.

Acknowledgments
This work has been done in the framework of the research project UPIC TIN2004-07461-C02-01.

References

1. A. Finkelstein, G. Spanoudakis, and M. Ryan, "Software Package Requirements & Procurement," *Proc. Int'l Workshop Software Specification and Design* (IWSSD), IEEE CS Press, 1996, pp. 141–145.
2. X. Franch and J.P. Carvallo, "Using Quality Models in Software Package Selection," *IEEE Software*, vol. 20, no. 1, 2003, pp. 34–41.
3. J.P. Carvallo, X. Franch, and C. Quer, "Managing Non-Technical Requirements in COTS Components Selection," *Proc. 14th IEEE Int'l Conf. Requirements Eng.* (RE 06), IEEE CS Press, 2006, pp. 323–328.
4. B.A. Kitchenham, R.T. Hughes, and S.G. Linkman, "Modeling Software Measurement Data," *IEEE Trans. Software Eng.*, vol. 27, no. 9, 2001, pp. 788–804.
5. G. Grau et al., "DesCOTS: A Software System for Selecting COTS Components," *Proc. 30th Euromicro Int'l Conf.* (Euromicro 04), IEEE CS Press, 2004, pp. 118–126.
6. C. Alves et al., "Using Goals and Quality Models to Support the Matching Analysis During COTS Selection," *Proc. Int'l Conf. COTS-Based Software Systems* (ICCBSS), Springer, 2005, pp. 146–156.
7. J.P. Carvallo et al., "Characterization of a Taxonomy for Business Applications and the Relationships between Them," *Proc. Int'l Conf. COTS-Based Software Systems* (ICCBSS), Springer, 2004, pp. 221–231.
8. V.R. Basili, G. Caldiera, and H.D. Rombach, "The Experience Factory," *Encyclopedia of Software Eng.*, J. Marciniak, ed., John Wiley & Sons, 1994, pp. 469–476.
9. T. Bollinger, J. Voas, and M. Boasson, "Persistent Software Attributes," *IEEE Software*, vol. 21, no. 6, 2004, pp. 16–18.
10. J. Bøegh et al., "A Method for Software Quality Planning, Control, and Evaluation." *IEEE Software*, vol. 16, no. 2, 1999, pp. 69–77.