Requirements & Specification Exemplars[1]

Martin S. Feather*, Stephen Fickas+, Anthony Finkelstein~, and Axel van Lamsweerde#

*Jet Propulsion Laboratory, Pasadena
+Computer Science Department, University of Oregon
~Department of Computer Science, City University
#Département d'Ingénierie Informatique, Université Catholique de Louvain

Abstract. *Specification exemplars are familiar to most software engineering researchers. For instance, many will have encountered the well known library and lift problem statements, and will have seen one or more published specifications. Exemplars may serve several purposes: to drive and communicate individual research advances; to establish research agendas and to compare and contrast alternative approaches; and, ultimately, to lead to advances in software development practices.*

Because of their prevalence in the literature, exemplars are worth critical study. In this paper we consider the purposes that exemplars may serve, and explore the incompatibilities inherent in trying to serve several of them at once. Researchers should therefore be clear about what successfully handling an exemplar demonstrates. We go on to examine the use of exemplars not only for writing specifications (an end product of requirements engineering), but also for the requirements engineering process itself. In particular, requirements for good requirements exemplars are suggested and ways of obtaining such exemplars are discussed.

1: What are Specification Exemplars?

The use of standard exemplars has become a widely accepted tool in specification research. For orientation the reader should have in mind the relatively small number of exemplars appearing frequently in the literature, such as: the lift and the library problems [Marca & Harandi 1987]; the production cell [Lewerentz & Lindner 1995]; the generalized railroad crossing problem [Heitmeyer et al 1993]; the steam boiler control system [Abrial et al 1995]; the patient monitoring system [Stevens et al 1974]; the conference organization system [Olle 1982]; the package router [London & Feather 1986]; the heating system [Marca & Harandi 1987]; the Swiss tournament system [van Diepen & Partsch 1991]; etc. A representative sample can be found in [Icarus 1989].

Such exemplars generally amount to a self-contained, informal description of a problem in some application domain; they are proposed as unique input for the specification process. Exemplars thus define, in the broadest sense, model specification tasks. They are to be considered immutable; the specifier must do the best she can to produce a specification from the problem statement. In this sense, they capture the harshness of reality - we cannot expect to change the world to make it more easily specified.

Note that exemplars are not quite comparable to the paradigmatic problems which have received attention in computer science, for example Dining Philosophers [Dijkstra 1971]. In such cases the presentation is entertaining but incidental to the problem; the goal of the person formulating the description is to characterize a problematic aspect of computation in as concise and transparent a manner as possible. By contrast, specification exemplars are intended, at some level, to represent the real-world specification task - that is, to be case studies. The presentation therefore is at least as important as the underlying set of concerns it embodies.

1. To appear in *Automated Software Engineering,* Kluwer Pubs., Vol. 4 No. 4, 1997.

Exemplars have been used variously for: creating and refining a specification technique; illustrating and explaining a specification technique; testing or validating a specification technique; and, on occasion, for comparing specification techniques.

In what follows, we first consider the major purposes which specification exemplars serve (Section 2), and identify their ramifications, particularly when suitability for one purpose may be at odds with suitability for another (Section 3). As illustration of these claims, we summarize our first-hand experience with two such exemplars (Section 4). We then address the emerging shift in attention from specification expression to requirements engineering, and the ramifications of employing exemplars in the latter realm (Section 5). An exemplar constructed expressly for addressing some of the concerns of requirements engineering is discussed (Section 5.2, with its full details in the appendix). Further steps towards improved exemplars for requirements engineering research are then proposed.

2: The Purposes of Specification Exemplars

There has been much debate in software engineering about research methodology. This debate has been driven by, on the one hand, some concern for more experimental work in software engineering research and, on the other hand, by a perception that software engineering research is proving particularly difficult to transfer to industrial application. It is particularly relevant in this context that we clarify the status, purposes and contribution of specification exemplars.

The following are the primary purposes of using specification exemplars:

• advancing a single research effort (that of an individual, or single research group);

• promoting research and understanding among multiple researchers or research groups;

• contributing to the advancement of software development practices.

In the following subsections we explore the ramifications of these issues.

(a) Advancing a single research effort

The exemplar must set some challenge by incorporating interesting aspects that motivate the introduction of a new specification technique. Accomplishing the specification of the exemplar must serve to explain and illustrate the points of the research in a clear and convincing manner, and suggest some reality check [Lewerentz & Lindner 1995].

The exemplars listed in Section 1 have been constructed to be small but nevertheless challenging. Their specifications typically (i) have sufficient size to demonstrate structuring schemes, (ii) have properties not immediately obvious by inspection, hence allowing experimentation with associated analysis techniques, (iii) lie on the boundary of what can be accomplished independent of tool support. Furthermore, many of them contain snares for the unwary [Wing 1988], [Lewerentz & Lindner 1995].

The resulting specifications should be arguably superior to specifications written using known techniques; it should be possible, but not trivial, to emerge with a good specification [Meyer 85] - one that is: provably consistent; satisfactorily complete; minimal (no uncontrolled redundancy, no overspecification); and well-structured to support incremental construction and compositional reasoning.

For the purpose of research explanation an exemplar must not require an overly large investment of time relative to the results yielded. The exemplars generally used in the literature therefore combine the following three characteristics:

• a domain setting already familiar to the computer science community - e.g., text formatting

[Meyer 85];

- a self-contained problem statement, formulated at a sufficiently high level of abstraction that intuitive notions suffice - e.g., lift control [Marca & Harandi 1987];
- a miniaturization of the problem that eschews sheer size of description at both start and end points (initial problem description, final specification) - e.g., library lending [Kemmerer 1985].

In spite of miniaturization, the problem descriptions are aimed to suggest some reality check through the choice of real domains in which the problem occurs. Researchers use various aspects of the exemplars to demonstrate the strengths and applicability of their own approaches in such real settings [Jones 1990], [Hayes 1993], [Lano & Haughton 1994], [Chung and Nixon 1995].

From the base exemplars, it is easy to construct plausible variants and extensions that further the goals of a researcher using that exemplar. The library problem is a typical instance of an exemplar that has undergone multiple variants and extensions in the literature. Such lack of closure may raise problems when exemplars are used to compare different approaches, as we discuss next.

(b) Promoting research comparison and understanding

The exemplar must act as an agent of research direction for the community (e.g., by focussing attention on a problematic specification issue), and facilitate the comparison of different specification techniques.

There have been a number of reviews, meetings and papers themed round common exemplars. Such collections have led to continuing attention and interchanges; the resulting proceedings have been widely cited. Handling a common exemplar through different techniques allows the respective strengths and weaknesses of those techniques to be highlighted [Olle et al 1983], [Wing 1988], [Lewerentz & Lindner 1995], [Abrial et al 1995]. For example, a specific constraint may be naturally captured in one language but need to be hard coded in the other; a property of interest may be easily proved in one formalism but with great difficulty (if at all) in the other; a problem in the original statement may be easily detected with one technique but not be revealed by the other. Sometimes implicit assumptions underlying the design of the language emerge as a consequence of this.

The most widely used specification exemplars are purpose-built (for scientific papers, courses, technical reports and so on). They may be defined so as to focus attention on some particular specification concern, for example:

- temporal constraints (heating system [Marca & Harandi 1987], railroad crossing [Heitmeyer et al 1993]);
- reactiveness (production cell [Lewerentz & Lindner 1995], steam boiler control [Abrial et al 1995]);
- explicit concurrency (doctors surgery [Roman & Babb 1989]);
- physical distribution (mine pump [Kramer et al 1983]);
- system structure (patient monitoring [Stevens et al 1974]);
- complex data (conference organization [Olle 1982]);
- interacting constraints (text formatting [Meyer 1985]).

Within their area of concern exemplars have proved to be excellent vehicles for the investigation of approaches. Since different approaches will typically have different sets of strengths and weaknesses, exemplars can be used to suggest how different approaches can perhaps be combined to yield a hybrid of their respective strengths. When an aspect appears problematic to all approaches, this suggests an area worthy of attention by the field as a whole.

Ideally exemplars should be designed to avoid modelling and implementation bias, so as to make them amenable to a wide range of research approaches without favoring any one in particular. Neutrality may be difficult to achieve; it turns out that many exemplars introduce one modelling bias or another. For example, the inability for state-based techniques to capture implicit historical referencing yields formulations such as "if a request cannot be satisfied immediately, it will be stored by the network" [Hayes 1993]; a statement such as: "to describe the system formally, we define a gate function $g(t) \in [0,90]$, where $g(t) = 0$ means the gate is down and $g(t) = 90$ means the gate is up" [Heitmeyer et al 1993] is another typical example of representation bias; see [Meyer 1985] for an analysis of implementation biases in the text formatting exemplar.

(c) Contributing to the advancement of software development practices

To fulfill this purpose an exemplar must represent the real-world specification task. This means that (i) the specification to be produced and analyzed should correspond to some fragment of a system that is likely to exist in the real world, and (ii) the specification process should be combined with the upstream activities of requirements acquisition and negotiation in some organizational setting, and with the downstream activities of architectural design, implementation, fielding and maintenance of the system specified.

There is no good example in the literature of a specification exemplar that achieves both conditions (i) and (ii) above. There has been some attempts to put forward specification techniques from exemplars that meet condition (i), e.g., [Heninger et al 1978], [Parnas et al 1991], [Leveson et al 1994], [Hinchey & Bowen 1995], [Modugno et al 1997]. None of the exemplars we are aware of integrates condition (ii) - and, in particular, the requirements elicitation dimension. This will be further discussed in Section 5.

3: Interferences Between Purposes of Exemplars

We now argue that an exemplar suited to one purpose is not necessarily suited to the other purposes. Indeed, many of the ramifications of seeking to fulfill one purpose are at odds with those others. *Thus we should be clear about what successfully completing specification of an exemplar demonstrates.* It would be a mistake to make overly-strong claims (e.g., that success for one purpose automatically means success for the others), and it would be a mistake to be overly critical (e.g., that failure to achieve success for all three purposes at once means that there has been no value in demonstrating success for an individual purpose).

Why an exemplar that satisfies purpose (a) may not satisfy purpose (b)

Specification exemplars are open to the accusation that they have been cooked up to show a specification technique, or class of specification techniques, to particular advantage. Moreover their formulation itself may be biased towards the mechanisms/paradigms supported by the technique used by their creators, as noted above.

While shared exemplars are a first step towards comparison between techniques, the extent to which they have actually led to proper critical comparison is open to question. The scope of exemplar-driven comparisons is often limited to symptoms of strengths and weaknesses, rather than their causes, and to the specification product rather than the process. Notable exceptions to this include: [Olle et al 1983], in which features of the languages and associated methods are analyzed and compared, not simply their solutions to the given exemplar; [Wing 1988], in which selected specification schemes are analyzed with respect to a set of ambiguities and incompletenesses.

Specification exemplars such as the library problem are easy to state and work on because the domain is familiar. It is easy to fill in domain details that are not written down in the statement of the exemplar. However, different teams may make different choices about the implicit domain. As noted before, such tendency to work on different variants and extensions of a same exemplar may make comparisons sometimes difficult.

The use of specification exemplars to support comparison of techniques is tied to a more general problem - the lack of appropriate evaluation criteria. It is all very well providing a specification of a standard exemplar but what, beyond the existence of a specification technique itself, does this demonstrate? How can we determine whether the specification language used on one exemplar has appropriate expressiveness, scaleability, evolveability, deductive power, development process efficiency, and so on? Without appropriate criteria the relevance of exemplars, regardless of their size or complexity, is limited. Furthermore, there is a tendency to promote only those exemplars best suited to our own approaches. Successes are emphasized, while failures are downplayed. Within the research literature, the few exceptions to this tend to be retrospective papers, often produced much later than the work upon which they reflect (e.g., [Balzer1985], [Zave 1991]).

Why an exemplar that satisfies purpose (a) or (b) may not satisfy purpose (c)

While the exemplars can easily be shared by researchers in the field, they lack credibility as representatives of the scale and multi-disciplinary nature of much of software development.

Many specification exemplars have been cooked up to show in a good light our limited ability to specify complex systems. They enable searching under the lamp-post - they push us to find and compare elegant formalisms in areas where we have existing expertise. The library exemplar [Kemmerer 1985] and the railroad crossing exemplar [Heitmeyer et al 1993] could be considered as typical examples of this.

The use of exemplars drawn almost incestuously from computer science domains begs the question of validity to the broader arena of software systems. Together, small scale, and problems all too easily mastered by the computer scientist give the misleading impression that specification tasks can be accomplished by a single analyst, and without the need to take advice from other parties (clients, users, domain experts, etc.). We have further observed that some specification exemplars embody the common, but often mistaken, assumption that the client is a domain expert and the analyst a neophyte. In our experience the reverse is often the case; the starting point of specification is not necessarily the clients saying what they want but the analyst saying what is possible.

In order to be self-contained, exemplars generally provide stripped down functional descriptions and a set of constraints on those functional descriptions. They tend to focus upon on a terminal stage where operational descriptions are available; the important decisions have already been made to resolve conflicting goals, assign responsibilities and delimit the boundary between the automated system and its environment. In addition, exemplars tend to present idealized systems, where unrealistic assumptions are made and tricky exceptions are ignored.

The starting problem description has most often been significantly tidied up. Unlike real-world situations, much (though by no means all) unnecessary detail, inconsistency, ambiguity and overspecification has been removed. This tidying up has a tendency to reduce the specification task to an issue of accurate representation (or perhaps translation). This diminishes the role of many practices. For example, no elicitation is required; conflicts and negotiation are absent; validation techniques are reduced to being mere confirmations of the already crafted correctness, rather than discoverers of errors. As a consequence, the tidying up of the exemplars focuses attention on the end-product (the specification) at the cost of the complex engineering process which gave rise to it. The result is a specification presented without its accompanying rationale and without any account of method.

Exemplars are often abstractions of embedded or composite systems; they would require a rich context to implement. The actual problems that exemplars are supposed to represent are rarely, if ever, solved in a real-world setting. Since current exemplars lack this detail, very few people have followed one down to the implementation of the associated embedded system. Given the disconnectedness of specification exemplars from any downstream process (design, implementation, testing and maintenance of an actual fielded system), it is clear that we should avoid generalizing from the success of specifying non-embedded exemplars to claim success on specifying embedded problems, let alone benefit to the entire software lifecycle.

The real-world software development process involves far more iteration and evolution than is represented by a static exemplar. In practice, specifications are frequently modified as understanding is gained of how easy or hard it will be to implement them. This could be approximated by having an exemplar comprise not just a single problem statement, but a series of problems.

The research community should perhaps do less invention of the problems upon which to focus. Instead, we could pay more attention to the problems that arise in practice. This point has been cogently argued in [Potts 1993], who suggests researchers use industry as their laboratory. The research world has produced a proliferation of languages with little reference to what are the specific real-world problems solved by language X that could not be solved by language Y. More generally, there is a lack of experimentation and evaluation in our field [Tichy et al. 1995]. Experimentations in real settings are promising steps towards the advancement of software development practices [Parnas et al 1991], [Hinchey & Bowen 1995], [Modugno et al 1997].

4: Examples of Exemplars

As illustration of our general points, we present two commonly used exemplars, drawn from different domains. We have chosen these because we have first-hand experience of working with them.

4.1: Library

Figure 1 holds the description of the library exemplar [Kemmerer 1985].

Consider a small library system with the following transactions:

1. *Check out a copy of a book/ Return a copy of a book.*
2. *Add a copy of a book to/ Remove a copy of a book from the library.*
3. *Get the list of books by a particular author or in a particular subject area.*
4. *Find out the list of books currently checked out by a particular borrower.*
5. *Find out what borrower last checked out a particular copy of a book.*

There are two types of users: staff users and ordinary borrowers. Transactions 1,2,4 and 5 are restricted to staff users, except that ordinary borrowers can perform transaction 4 to find out the list of books currently borrowed by themselves. The system must also satisfy the following constraints:

1. *All copies in the library must be available for check- out or checked out.*
2. *No copy of a book may be both available and checked out at the same time.*
3. *A borrower may not have more than a pre-defined number of books checked out at one time.*

Figure 1 - The library exemplar

Observations

(a) Advancing a single research effort:

This statement of the library problem exemplifies the abstracted and miniaturized nature of many exemplars. It is a perennial favorite for quickly conveying aspects of a language / approach / tool set, etc. If a researcher is concerned primarily with the specification of database constraints, then the library exemplar represents a readily understood illustration of an abstract borrowing system, or even more abstractly, of a system that combines transactions processing and history tracking. These chosen aspects can be carried through to simulation, testing and prototyping.

The library exemplar sets several traps/challenges, e.g., confusion between the concepts of book and book copy; confusion between readers in the physical library and users of the automated system; privacy concerns; etc.

(b) Promoting research and understanding over the community of specification researchers:

This problem was used as one of four focus problems distributed in advance by the 4th International Workshop on Specification and Design [Marca & Harandi 1987]. Would-be participants were strongly encouraged to address one or more of these problems in their submissions; the published proceedings reflects this emphasis. Afterwards, twelve specifications of the exemplar were studied to suggest the strengths and weaknesses of the various approaches [Wing 1988].

(c) Contributing to the advancement of software development practices:

The library exemplar is very far from representative of real-world specification; it seems to play no part in the real-world lifecycle of library software. The check-in and check-out activities of libraries are regarded as a solved problem by real-world librarians [Potts & Fickas 1994], for which off-the-shelf software packages are available to do the bookkeeping. Thus the tackling of this exemplar, while beneficial for easily conveying some proposed approach, in itself provides no indication of the utility of that approach.

Another way to use this exemplar, however, is as inspiration to elicit and investigate the real problems in this domain. Our work with automated library systems [Fickas & Nagarajan 1988] gives a very different picture of specification in practice, as contrasted to the exemplar itself. The real-world specification task of designing an entire library system often involves: rich non-functional requirements, particularly those relating to cost, performance, accuracy and evolveability; the importance accorded to system-level and organizational objectives; the important role of references to projected system architectures and envisaged scenarios to establish feasibility; the mutability of the services required depending on the balance of costs and benefits; the prevalence of system extension and evolution.

4.2: Package Router

Figure 2 holds the description of the package router exemplar [London & Feather 1986].

Observations

(a) Advancing a single research effort:

The domain, that of distribution of physical objects by controlling their routing through a simple mechanism, is easy to understand; it requires little effort on the part of the would-be specifier to acquire domain knowledge.

We used this exemplar to demonstrate the freedoms from implementation concerns that we believed a specification language should exhibit, and the ways in which transformation towards an implementation could address those freedoms. For example, the specification of switch setting should mirror the problem statement, where the desired behavior ("...to direct the package through the network and into the correct bin") is stated; in contrast, an implementation must compute when and how to set switches so as to achieve this behavior.

Our treatment of this problem mixed abstraction and case study. We used specific instances drawn from this problem to illustrate our general points; also, we addressed the problem as a whole as a single case study.

Our specification of this problem was approximately 6 pages in length, including comments [London & Feather 1986]; thus while it was large enough to require some thought on how to organize it for readability, it remained well within the size that could be constructed and manipulated by hand.

A source station at the top feeds packages one at a time into the network, which is a binary tree consisting of switches connected by pipes. The terminal nodes of the binary tree are the destination bins.

When a package arrives at the source station, its intended destination (one of the bins) is determined. The package is then released into the pipe leading from the source station. For a package to reach its designated destination bin, the switches in the network must be set to direct the package through the network and into the correct bin.

Packages move through the network by gravity (working against friction), and so steady movement of packages cannot be guaranteed: they may bunch up within the network and thus make it impossible to set a switch properly between the passage of two such bunched packages (a switch cannot be set when there is a package or packages in the switch for fear of damaging such packages). If a new package's destination differs from that of the immediately preceding package, its release from the source station is delayed a pre-calculated, fixed length of time (to reduce the chance of bunching). In spite of such precautions, packages may still bunch up and become mis-routed, ending up in the wrong bin; the package router is to signal such an event.

Only a limited amount of information is available to the package router to effect its desired behavior. At the time of arrival at the source station but not thereafter, the destination of a package may be determined. The only means of determining the locations of packages within the network are sensors placed on the entries and exits of switches, and the entries of bins; these detect the passage of packages but are unable to determine their identity. (The sensors will be able to recognize the passage of individual packages, regardless of bunching).

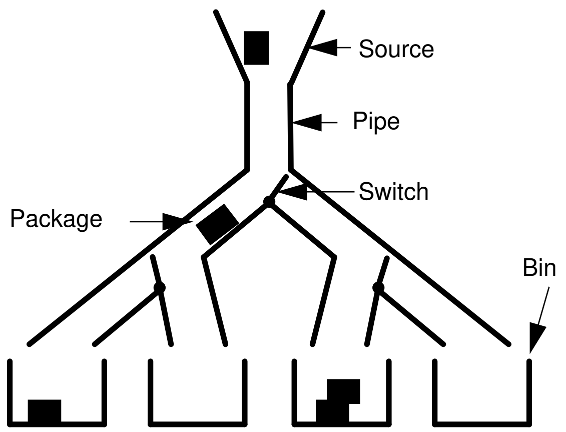

Figure 2 - The package router exemplar

The challenge set by this exemplar is the classic one of 'what' not 'how', namely, to specify what destinations packages are to be routed to, not how to program switches so as to achieve this. The biggest snare of this example lies in the potentially unavoidable misrouting of packages should they bunch up during movement. We have never been completely satisfied with any treatment (ours included) in the specifications that we have seen. In many ways, however, we found this an excellent exemplar for the purpose of driving our own research, and demonstrating our results to others. Of course, our ready acceptance of this exemplar was motivated by its suitability to the specification style we had already established.

(b) Promoting research and understanding over the community of specification researchers:

We found it illuminating to present the package router at a meeting of IFIP Working Group 2.1 and see the specifications that several of the people produced.

(c) Contributing to the advancement of software development practices:

Credibility is lent to this problem by virtue of its origin - it came to us from representatives of the process control industry, who had constructed it to serve as a (small-scale) challenge by which they could judge the potential contribution of various methodologies. Presumably it contains the essence of at least some of the aspects of the problems they face. Also, we, in trying to specify a solution, can avoid the accusation that we constructed the exemplar ourselves to serve our own purposes.

Nevertheless, this remains a simplified and small-scale problem. A real-world problem would likely be much more complex, by virtue of a more intricate topology (the binary tree structure of the router has the simplicity of a unique path from source to any given destination), by having to tolerate imperfect mechanisms (e.g., sensors that sometimes fail to notice the passage of a package - especially packages bunched up), and by having to make design trade-offs (e.g, between throughput and accuracy). In summary, there is still a vast gap between specifying the simple package router exemplar and tackling the scale of problem of, say, the automated baggage distribution system at the new airport in Denver, Colorado; this system is reported to employ 3,000 conveyor drives, 2,750 photo cells and 112 programmable logic controllers, and proved somewhat difficult to get operating reliably.

A further aspect illustrated by this exemplar is its fine-tuned construction: the available capabilities (control over switches, access to information provided by sensors) and the requirements on routing have been carefully crafted to be in harmony. Fewer capabilities would preclude implementation; more would be unnecessary. It is clearly not the first cut at stating the problem, rather, it is the result of careful forethought. Swartout and Balzer observed this, and went on to make the more general point that specifying what you want is inevitably intertwined with knowledge of how it is to be implemented [Swartout & Balzer 1982]. The implication for specification exemplars is that one should expect to be involved in an ongoing cycle of specification and re-specification, not simply the one-time specification of a perfectly crafted problem statement.

We never carried implementation through to anything other than a sanitized, purely software, simulation of routing; we did not attempt to hook this up to any mechanism that controlled movement of physical packages. Thus we never faced the task of, say, tuning our design to lead to better performance. For mechanical control problems, it is unlikely that software researchers will be granted access to run experiments on full-scale machinery; however it may be feasible to experiment with scale models, say (for example, [Kramer et al 1990] had their software control a toy train system).

Furthermore, we did have to modify our specification in response to a change in the problem or its context, as would likely be the case in any real development.

5: Towards Effective *Requirements* Exemplars

We have observed a shift in attention among researchers towards the upstream concerns of how to acquire, elaborate, evaluate, and negotiate alternative requirements, and from them emerge with a specification; the latter is merely an end-product of the requirements engineering process. (In the unfortunate cases where a specification is never produced as an explicit intermediary between requirements and implementation, the implementation itself represents an implicit resolution of requirements issues.)

This shift towards requirements engineering leads us naturally to the question - what sort of exemplars are suitable for supporting work in this area, and how should they be used? As a first step let us consider what objectives requirements engineering exemplars should meet.

5.1: Objectives Particular to Requirements

Requirements exemplars share with specification exemplars many of the objectives discussed above. In particular, requirements exemplars should: test the ability of a technique to capture a consistent, complete and unambiguous specification; not require an overly large investment of time; act as an agent of research direction; support comparison of different techniques; provide a prelude to implementation and fielding of actual systems; represent the corresponding real-world task.

Requirements engineering encompasses a much broader range of concerns though. A specification is an end-product of the requirements process - it represents a hopefully clear and consistent agreement on what should be implemented. To get to such an end-point, requirements engineering has to:

- deal with a wide variety of aspects - such as goals, assumptions, constraints, services, costs, resources, responsibilities, priorities, traceability, evolution, etc.
- work with multiple sources and media such as interviews, observations of the place in action, documents about the existing system, knowledge about the domain and about similar systems, etc.
- gather information from multiple viewpoints and stakeholders;
- handle interfering goals giving scope for conflict detection and resolution;
- reason with intermediate incoherent descriptions;
- identify, assess, and select among different automation alternatives;
- reason about the interaction/cooperation between the system and its environment;
- organize a vast amount of concerns, details and exceptions into a coherent, understandable and manageable structure.

The specification exemplars mentioned in the previous sections fail to encompass most of these concerns. Primarily, specification exemplars have already pre-solved the bulk of the requirements issues, and are too static in nature. We consider possible remedies - a focus on case studies obtained from industry, the design of exemplars particularly suited to incorporate requirements concerns, and, finally, speculations on how requirements exemplars may be made more dynamic.

5.2: Industrial Case Studies of Requirements

An approach adopted by some research groups is to obtain interesting case studies from industry [Heninger et al 1978], [Parnas et al 1991], [Hinchey & Bowen 1995]. This may guarantee the realism of the exercise. It also generally imposes a task size such that the use of tools is required, thus providing further opportunities for tool evaluation [Modugno et al 1997].

However, for industrial case studies a careful balance needs to be struck between the pay-back provided by the exemplars and the need to do research in reasonable time.

There are some major obstacles that stem from organizational barriers and problems of working on a live project. Confidentiality concerns - whether motivated by security (as in defence applications) or commercial advantage, often impede access to the requisite information, and impede sharing and dissemination of results. For systems that are evolutionary in the sense that products result from the extension and enhancement of preexisting artifacts rather than specified and produced de novo, the requirements process is spread over long periods of time. For software produced by organizations for use within the organization itself, the process is characterized by the large amount of shared knowledge of the domain and of organizational practices. This makes access by external researchers difficult.

The result of these difficulties is that researchers tend to work on projects which are already completed (or have recently deceased). They must rely on existing documentation and, as most organizations do not preserve rationale, track the elicitation process or make domain knowledge explicit; this limits the visibility of the specification process. The obvious danger is that researchers are forced onto the familiar narrow track of “translating” a pre-existing specification already filtered by previous passes through the specification process. In short, case studies from completed projects often do not address the objectives stated in Section 5.1.

Industrial case studies also tend to exhibit a “forest hiding the tree” syndrome - a forest of repetitive details hiding the trees of interest. This could be taken to mean that industrial case studies are misleading, or that they reveal that what researchers think of as characteristics of problems interesting to work on are not the norm, instead in reality people are mired in a sea of shallow problems.

Another possibility is to ask industry to provide exemplars; an instance of this approach is the Package Router exemplar (section 3.2). This approach still has many of the problems identified above but at least the people providing the exemplars and the people using them for demonstration are disjoint. This may also lead to fielding of exemplars where actual use brings in other concerns.

Taken together, the various impediments above render it infeasible to rely upon industrial case studies as the *only* source, or even as the primary source, of requirements exemplars. The field is thus likely to continue to employ self-generated exemplars as common vehicles for research exchange.

5.3: Designed Requirements Exemplars - the Meeting Scheduler Example

Requirements exemplars must exhibit the messy nature of real-world requirements, in which no unique clean and neat solution is evident.

The meeting scheduler exemplar elaborated by van Lamsweerde and colleagues is one attempt to move into this messier world [van Lamsweerde et al 1995]. It builds on early attempts to provide improved exemplars, for example the Swiss tournament system [van Diepen & Partsch 1991]. The starting description is given in the Appendix. The supposition is that this first cut is wholly inadequate to produce a useful implementation. What separates this exemplar out is:

- a description of both the software and its environment, with requirements and assumptions on both parts;
- shadowy areas requiring further elicitation, for example, typically vague requirements taking the form “as *X* as possible”, where *X* may be small, fast, reliable, etc.;

- conflicts between interfering objectives that need to be resolved through reasonable compromises;
- a large space of alternatives to be negotiated during the specification process;
- a statement of likely changes, variants and extensions;
- the ability and expectation that an implementation will be produced, validated, and *used*.

This exemplar, then, exhibits many of the objectives we desire of requirements exemplars:

- it arises out of a real problem;
- the domain of expertise is accessible;
- it covers many interesting specification issues, for example, complex concurrency and distribution aspects, real-time performance constraints, non-functional requirements such as privacy, usability and flexibility, optimization requirements, etc.

At the same time, it answers several of our criticisms:

- it is implementable and validatable without an enormous outpouring of implementation resources;
- it forces one to address many of the requirements engineering concerns mentioned above;
- it is representative of an interesting set of distributed groupware systems.

On the downside, it is in reality a simplification of a real-world task formed from past experiences with organizing a variety of meetings, so has already gone beyond the earliest stages of requirements engineering. Also, the roles of analyst, domain expert and client are being served by the same person, thus omitting all inter-personal communication problems.

5.4: Towards Dynamic Requirements Exemplars

Specification exemplars have a highly static nature in which a frozen description is provided once to the specifier, who then proceeds to develop a specification without the need for any further interaction with the problem provider to elicit, refine, or question further aspects. Much more dynamic exemplars are needed to address many of the concerns particular to requirements. We suggest the following as steps that might be taken in this direction:

1. Provide multiple aspects of the problem statement in an incremental, non-monotonic fashion to the specifier. It would be relatively straightforward to provide textual changes which could then be fed in at an appropriate point. More audaciously we might envisage a future generation of exemplars which are less like the static text descriptions and more like games that unfold based on the actions of the participants, using a variety of interactive presentation media (interview video, originals of documentation and so on).
2. Insist that the results include not only the end-product (e.g., a specification with traceability back to the requirements), but also a record of the *process* followed to attain that end-product.
3. Encourage complementary efforts in which different participants and multiple perspectives address different aspects of the problem, in a way that such efforts can be combined. One of the characteristics of specification exemplars has been that researchers tend to work independently of one another, in direct competition. It is clear from the breadth of concerns encompassed by requirements that no one researcher, or single research team, can be expected to solve all of the problems. Hence the need to have researchers make their tools and techniques available for use by one another.

We are currently following such an approach in our studies of the meeting scheduler. In loose collaboration, we are pursuing in parallel the themes of formalizing the requirements product and process, accommodating multiple viewpoints and divergences, monitoring the run-time satisfaction of requirements, and exploring the issues of negotiation and compromise.

6: Conclusion

The analysis set out above has been primarily critical. However, it is not our objective to be negative or to question the many advances that have been made in the development of specification techniques. Rather, we are arguing for a greater degree of methodological self-consciousness on the part of researchers. Such self-consciousness is particularly important as we attempt to migrate to a related but nevertheless different field such as that of requirements engineering.

Shared exemplars are indeed conducive to promoting progress, common understanding, and comparison of one another's languages, methods, and tools. However, we need to concentrate more on exemplars that are more representative of the real-world requirements engineering task - dirty exemplars with a large decision space, multiple sources to elicit from, high-level concerns to operationalize, conflicts to detect and resolve, vague statements to make precise, premature design decisions to reengineer, and lots of holes to fill in.

Moreover, we need to concentrate more on sharing use, not just understanding. It is crucial that, within software engineering as a whole, we assess and apply the products of each others research and make our own research available in a form that others can assess and apply.

Acknowledgement. Thanks to Dan Berry, Robert Darimont, Philippe Massonet, Bashar Nuseibeh, David Till and the anonymous reviewers for their help, guidance and suggestions.

7: References

[Abrial et al 1995] J.R. Abrial, E. Borger, and H. Langmaack (Eds.), *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control System*. LNCS 1165, Springer-Verlag, 1996.

[Balzer 1985] B. Balzer, "A 15 year perspective on automatic programming", *IEEE Trans. on Software engineering* Vol. 11 No. 11, 1985, pp. 1257-1267.

[Chung and Nixon 1995] L. Chung and B. Nixon, "Dealing with Non-Functional Requirements: Three Experimental Studies of a Process-Oriented Approach", *Proc. ICSE17 - Seventeenth International Conference on Software Engineering*, IEEE-ACM, April 1995, pp. 25-37.

[Dijkstra 1971] E.W. Dijkstra, "Hierarchical Ordering of Sequential Processes", *Acta Informatica* 1, 1971, pp. 115-138.

[Fickas & Nagarajan 1988] S. Fickas and P. Nagarajan, "Critiquing Software Specifications", *IEEE Software*, Nov. 1988, pp. 37-47.

[Hayes 1993] I. Hayes, *Specification Case Studies*. 2nd Edition. Prentice Hall, 1993.

[Heninger et al 1978] K. Heninger, J. Kallender, D.L. Parnas and J. Shore, "Software Requirements for the A-7 Aircraft", NRL Report 3876, US Naval Res. Lab., Washington D.C., 1978.

[Heitmeyer et al 1993] C. Heitmeyer, R. Jeffords, and B. Labaw, "A Benchmark for Comparing Different Approaches for Specifying and Verifying Real-Time Systems", *Proc. 10th Intl Workshop on Real-Time Operating Systems and Software*, May 1993. *Reprinted in* C. Heitmeyer and D. Mandrioli (Eds.), *Formal Methods for Real-Time Computing*, Wiley, 1996, p. xii.

[Hinchey & Bowen 1995] M. Hinchey and J. Bowen (Eds.), *Applications of Formal Methods*. Prentice Hall Intl Series in Computer Science, 1995.

[Icarus 1989] Icarus, "A Tasteful Variety of Specification Case Studies", ESPRIT Project 2537, 1989. Available at ftp://ftp.info.ucl.ac.be/pub/publi/89/CaseStudies.ps.

[Jones 1990] C.B. Jones and R.C. Shaw, *Case Studies in Systematic Software Development*. Prentice Hall, 1990.

[Kemmerer 1985] R.A. Kemmerer, "Testing Formal Specifications to Detect Design Errors", *IEEE Transactions on Software Engineering*, Vol. 11 No. 1, January 1985, pp. 32-43.

[Kramer et al. 1983] J. Kramer, J. Magee, M. Sloman and A. Lister, "CONIC: an integrated approach to distributed computer control systems", *IEE Proceedings* Vol. 130, Pt.E, No. 1, 1983, pp. 1-10.

[Kramer et al 1990] J. Kramer, J. Magee, A. Finkelstein, "A Constructive Approach to the Design of Distributed Systems", *Proc. 10th International Conference on Distributed Computing Systems*, IEEE, 1990, pp. 580-587.

[Lano & Haughton 1994] K. Lano and H. Haughton (Eds.), *Object-Oriented Specification Case Studies*. Prentice Hall Object-Oriented Series, 1994.

[Leveson et al 1994] N. Leveson, M. Heimdahl, H. Hildreth, and J. Reese, "Requirements Specification for Process-Control Systems", IEEE Transactions on Software Engineering, Vol. 20 Nr. 9, September 1994 684-706.

[Lewerentz & Lindner 1995] C. Lewerentz and T. Lindner (Eds.), *Formal Development of Reactive Systems - Case Study Production Cell*. LNCS 891, Springer-Verlag, 1995.

[London & Feather 1986] P.E. London and M.S. Feather, "Implementing Specification Freedoms", in C. Rich & R.C. Waters (eds.); *Readings in Artificial Intelligence and Software Engineering*, Morgan Kaufmann, 1986, pp. 285-305

[Marca & Harandi 1987] D. Marca and M. Harandi, "Problem Set for the Fourth International Workshop on Software Specification and Design", in *Proc. 4th International Workshop on Software Specification and Design*, IEEE CS Press, 1987, pp. ix-x.

[Meyer 1985] B. Meyer, "On Formalism in Specifications", *IEEE Software* VoL. 2 No. 1, 1985, pp. 6-26.

[Modugno et al 1997] F. Modugno, N. Leveson, J. Reese, K. Partridge, and S. Sandys, "Integrated Safety Analysis of Requirements Specifications", *Proc. RE'97 - 3rd IEEE Symposium on Requirements Engineering*, Annapolis (MD), 1997, pp. 148-159.

[Olle 1982] T. Olle, "Comparative Review of Information Systems Design Methodologies - Stage 1: Taking Stock," in T. Olle, H. Sol & A. Verrijn-Stuart (eds); *Information Systems Design Methodologies: a comparative review,* Proc. IFIP WG8.1 CRIS 1, North-Holland, pp. 1-14.

[Olle et al 1983] T. Olle, H. Sol and C. Tully, "Information Systems Design Methodologies: a feature analysis", in *Proc IFIP WG8.1 CRIS 2*, North-Holland, 1983.

[Parnas et al 1991] D.L. Parnas, J.K. Asmis, and J. Madey, "Assessment of Safety-Critical Software", *Nuclear Safety*, Vol. 32 No. 2, 1991, pp. 189-198.

[Potts 1993] C. Potts, "Software Engineering Research Revisited", *IEEE Software*, Sept. 1993, pp. 19-28.

[Potts & Fickas 1994] C. Potts and S. Fickas, "Requirements Engineering," section of "Succeedings of the 7th International Workshop on Software Specification and Design", in *ACM SIGSOFT Software Engineering Notes*, Vol. 19 No. 3, 1994, pp 18-22.

[Roman & Babb 1989] G-C. Roman and R. Babb, "Concurrency, Coordination and Distribution", *ACM SIGSOFT Software Engineering Notes*, VoL. 14. No. 5, 1989, pp 37-38.

[Stevens, Myers & Constantine 1974] W. Stevens, G. Myers and Constantine, "Structured Design", *IBM Systems Journal* Vol. 13 No. 2, pp. 115- 139.

[Swartout & Balzer 1982] W. Swartout and R. Balzer, "On the Inevitable Intertwining of Specification and Implementation", *Communications of the ACM*, Vol. 25 No. 7, pp. 483-440.

[Tichy et al. 1995] W.F. Tichy, P. Lukowicz, L. Prechelt and E.A. Heinz, "Experimental Evaluation in Computer Science: A Quantitative Study", *Journal of Systems and Software,* Vol. 28 No. 1, Jan. 1995, pp. 9-18.

[van Diepen & Partsch 1991] N. van Diepen and H.A. Partsch, "Formalizing Informal Requirements: Some Aspects", in J.A. Bergstra. & L.M.G. Feijs (Eds); *Algebraic Methods II: Theory, Tools and Applications,* LNCS 490, Springer-Verlag 1991, pp. 7-27.

[van Lamsweerde et al 1995] A. van Lamsweerde, R. Darimont and P. Massonet, "Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learned", *Proc. RE'95 - 2nd Int. Symp. on Requirements Engineering*, York, IEEE, 1995, pp. 194-203.

[Wing 1988] J.M. Wing, "A Study of 12 Specifications of the Library Problem", *IEEE Software*, July 1988, pp. 66-76.

[Zave 1991] P. Zave, "An Insider's Evaluation of PAISLey", *IEEE Trans. on Software engineering* Vol. 17 No. 3, 1991, pp. 212-225.

8: Appendix: The Meeting Scheduler Exemplar

The preliminary description that follows is deliberately intended to be sketchy and imprecise. Elicitation, formalization and validation processes are needed to complete it and eliminate the many shadowy areas.

A number of features of the Meeting Scheduler exemplar were inspired from various experiences in organizing meetings (faculty meetings, ESPRIT project meetings, program committee meetings, etc.).

Scheduling Meetings: Domain Description

Meetings are typically arranged in the following way. A *meeting initiator* asks all potential meeting attendees for the following information based on their personal agenda:

- a set of dates on which they cannot attend the meeting (hereafter referred as *exclusion set*);
- a set of dates on which they would prefer the meeting to take place (hereafter referred as *preference set*).

A meeting date is defined by a pair (calendar date, time period). The exclusion and preference sets are contained in some time interval prescribed by the meeting initiator (hereafter referred as date range).

The initiator also asks active participants to provide any special equipment requirements on the meeting location (e.g., overhead-projector, workstation, network connection, telephones, etc.). He/she may also ask important participants to state preferences about the meeting location.

The proposed meeting date should belong to the stated date range and to none of the exclusion sets; furthermore it should ideally belong to as many preference sets as possible. A *date conflict* occurs when no such date can be found. A conflict is strong when no date can be found within the date range and outside all exclusion sets; it is weak when dates can be found within the date range and outside all exclusion sets, but no date can be found at the intersection of all preference sets. Conflicts can be resolved in several ways:

- the initiator extends the date range;
- some participants remove some dates from their exclusion set;
- some participants withdraw from the meeting;
- some participants add some new dates to their preference set.

A meeting room must be available at the selected meeting date. It should meet the equipment requirements; furthermore it should ideally belong to one of the locations preferred by as many important participants as possible. A new round of negotiation may be required when no such room can be found.

The meeting initiator can be one of the participants or some representative (e.g., a secretary).

System Requirements

The purpose of the meeting scheduler system is to support the organization of meetings - that is, to determine, for each meeting request, a meeting date and location so that most of the intended participants will effectively participate. The meeting date and location should thus be as convenient as possible to all participants. Information about the meeting should also be made available as early as possible to all potential participants. The intended system should considerably reduce the amount of overhead usually incurred in organizing meetings where potential attendees are distributed over many different places. On another hand, the system should as closely as possible reflect the way meetings are typically managed (see the domain description above).

The system should assist users in the following activities.

- Plan meetings under the constraints expressed by participants (see domain description).
- Replan a meeting dynamically to support as much flexibility as possible. On one hand, participants should be allowed to modify their exclusion set, preference set and/or preferred location before a meeting date/location is proposed. On the other hand, it should be possible to take some external constraints into account after a date and location have been proposed - e.g., due to the need to accommodate a more important meeting. The original meeting date or location may then need to be changed; sometimes the meeting may even be cancelled. In all cases some bound on replanning should be set up.
- Support conflict resolution according to resolution policies stated by the client.
- Manage all the interactions among participants required during the organization of the meeting, for instance, to communicate requests, to get replies even from participants not reacting promptly, to support the negotiation and conflict resolution processes, to make participants aware of what is going on during the planning process, to keep participants informed about schedules and their changes, to make them confident about the reliability of the communications, etc.

The amount of interaction among participants (e.g., number and length of messages, amount of negotiation required) should be kept as small as possible.

The meeting scheduler system must in general handle several meeting requests in parallel. Meeting requests can be competing by overlapping in time or space. Concurrency must thus be managed.

The following aspects should also be taken into account.

- The system should accommodate decentralized requests; any authorized user should be able to request a meeting independently of his whereabouts.
- Physical constraints should not be broken - e.g., a person may not be at two different places at the same time, a meeting room may not be allocated to more than one meeting at the same time, etc.
- The system should provide an appropriate level of performance, for example:
  - the elapsed time between the submission of a meeting request and the determination of the corresponding meeting date/location should be as small as possible;
  - the elapsed time between the determination of a meeting date/location and the communication of this information to all participants concerned should be as small as possible;
  - a lower bound should be fixed between the time at which the meeting date is determined and the time at which the meeting is actually taking place.
- Privacy rules should be enforced; a non-privileged participant should not be aware of constraints stated by other participants.
- The system should be usable by non-experts.
- The system should be customizable to professional as well as private meetings. These two modes of use are characterized by different restrictions on the time periods that may be allocated (e.g., meetings during office hours, private activities during leisure time).
- The system should be flexible enough to accommodate evolving data - e.g., the sets of concerned participants may be varying, the address at which a participant can be reached may be varying, etc.

- The system should be easily extendable to accommodate the following typical variations:
  - handling of explicit priorities among dates in preference sets;
  - handling of explicit dependencies between meeting date and meeting location;
  - participation through delegation - a participant may ask another person to represent him/her at the meeting;
  - partial attendance - a participant may only attend part of the meeting;
  - variations in date formats, address formats, interface language, etc.
  - partial reuse in other contexts - e.g., to help establish course schedules.

Extending the System with Additional Knowledge for Conflict Resolution

Knowledge about participant status and about priorities among users and meetings should help in determining a “best” way to resolve a conflict. Even when there is no conflict, the participant status may be useful in determining a “best” meeting date and location.

The following notions should be incorporated in the proposed extension. They capture the hierarchical importance of participants, the importance for a participant to attend a particular meeting relatively to other participants or to other meetings, and the ease with which a participant can make a particular date interval free. These various notions should be used in the conflict resolution process.

Participant Status. The participant status captures the hierarchical importance of a participant with respect to others independently of any specific meeting he/she is expected to participate in. This attribute might be used, e.g., to determine a “best” compromise on date and location whenever several ones are possible. The participant status is typically determined by some superuser. For instance, in the context of scheduling faculty meetings the department head would have a higher status than normal professors. The latter would have a higher status than student representatives.

Participant Importance. The participant importance captures the importance for a specific person to attend a particular meeting *relatively to other participants*. Participant importances are typically determined by the meeting initiator. For instance, the meeting chair and secretary must be present; they have the highest participant importance. In a project meeting where specific tasks are discussed, task leaders would have a higher importance than normal project members and a lower importance than the meeting chair, the task speakers or the project reviewers.

Meeting Significance. The meeting significance represents the importance for a specific person to attend a particular meeting *relatively to other meetings or meeting requests*. Meeting significances are typically determined by the participants concerned. For instance, participants to a specific task in a research project would assign a greater significance to a project meeting where their task will be discussed. This information must be kept confidential.

Participant Flexibility. The participant flexibility is intended to indicate how easily a user can make a particular date interval free to allow meetings to be scheduled within that interval. Dates in exclusion sets and/or preference sets can thus be weighted accordingly. The participant flexibility is typically determined by the participants concerned. For instance, professors cannot move lecture periods easily; their participant flexibility for the corresponding date intervals should be low. A date interval which is not in the exclusion set of a participant should have a high flexibility for that participant. This information must be kept confidential.

Using Knowledge about Status and Priorities. The following tactics illustrate some typical uses of the various kinds of priorities suggested above.

- Best meeting dates and locations should be determined by considering participants with higher participant *status* first.
- If no date can be found to organize a meeting, the system could propose a person having low participant *importance* to withdraw from the meeting.
- If no date can be found to organize a meeting, the system could propose a participant to cancel (or to withdraw from) another meeting having a lower meeting *significance*.
- A meeting date within some exclusion set (or outside some preference set) could be considered if the corresponding participant has a high *flexibility* for it.