

# Invented Requirements and Imagined Customers: Requirements Engineering for Off-the-Shelf Software

Colin Potts

College of Computing, Georgia Institute of Technology  
potts@cc.gatech.edu

## Abstract

*The requirements engineering research and consulting communities are not serving the interests of software developers who build off-the-shelf application software. Most of our models and methods evolved with the aid of funding from organizations interested in obtaining unique systems under contract and in which there is a clear interface between “customer” and “developer”. These origins have spawned many assumptions about what requirements are. Through several design scenarios I illustrate how these assumptions break down in the case of off-the-shelf software. I then suggest some alternative priorities that would address these shortcomings. My aim is to provoke and stimulate thought, not to propose a developed solution.*

## 1. Product Development is Different

Several years ago, I wrote a paper called “The Other Interface” [5] for an audience of HCI practitioners and researchers. In it, I argued that requirements analysis is a process of communication across a cultural barrier between developer and customer. There was nothing remarkably new in this claim. Governments buy software by negotiating contracts, and many large organizations today still require unique systems that they purchase by contract. In these cases, the contractual process enforces a strict division between customer and developer. But it seemed odd to me that no one thought of the barrier these practices introduced as an interface comparable to a user interface. I argued that behavioral science could be applied to eliciting information and expressing requirements just as surely as it could be applied to constructing user interfaces.

I mention this because it is a good example of misguided thinking arising from a flawed metaphor. Undoubtedly, HCI can contribute to requirements engineering, but not if we construe analysis as simple communication across an interface. During requirements analysis, one party does not always “elicit” requirements from another, nor does it “play back” requirements so that the other may accept, reject or refine them. Such a description simplifies even a contractual procurement, but

developers of off-the-shelf software produce requirements through a process of collective and incremental *design*.

## 2. Design Scenarios

To illustrate these distinctions consider some design scenarios.

### 2.1. Generic office tools

One way to design generic off-the-shelf tools is to automate a manual process or tool so that physical artifacts are replaced by more malleable and versatile virtual artifacts. Word processing packages are a prosaic example. Pre-computer offices used typewriters to print on plain and headed paper; the word processing package extends this mechanical precursor in valuable but obvious ways.

Some generic tools result from more serendipitous and creative thinking. The spreadsheet program seemed revolutionary when VisiCalc was first introduced. Although accountants manipulated balance sheets before the invention of the spreadsheet program the transition from manually manipulating physical sheets to abstractly manipulating virtual sheets required a leap of innovation that was only obvious in hindsight.

Today, however, there is little to distinguish the way in which word processing packages and spreadsheet programs are enhanced and new products created. Clear market leaders have emerged and the vocabulary of the average user reflects the models provided by the most popular tools. Word processing packages, for example, must provide “style sheets” or an analogous feature. Requirements for a new package depend more on the marketability of named features and feature comparisons with competing products than on “elicitation” of requirements from “the customer.”

Some features come from usage patterns, and therefore reflect, albeit in an indirect and filtered fashion, the needs of likely users. Spell checkers and grammar checkers, for example, are programmed to look for typical errors, such as consecutive repetitions of the same word. Spreadsheet programs are increasingly used to prepare business

presentations, and so modern spreadsheet programs include features for preparing and delivering presentations.

The gradual embellishment and streamlining of software features over time closely resembles the evolution of designs for other engineered artifacts, such as paper clips and wheelbarrows [4]. Designers of such artifacts do think about user needs, and may even undertake some task analysis, but they do not do what we call “requirements engineering”.

## 2.2. Specialized power tools

You might object that office tools do not require requirements engineering precisely because they are generic. What makes requirements determination so problematic is that developers often lack relevant domain knowledge [1,3] and unwittingly embed their misunderstandings in the specifications and code they produce. These misunderstandings occur only when the application is unfamiliar.

A good example of a special-purpose off-the-shelf application is the upper-CASE tool that supports structured or object-oriented methods for modeling systems and portraying them as collections of boxes and arrows. The essence of a development method is that someone, the method protagonist, recommends or prescribes development procedures and analyses. CASE tools differ in the strength with which they enforce a mode of working on the user, but all embed assumptions about what constitutes good analysis and design practice. These assumptions could be “wrong” in a number of senses that deserve more lengthy analysis than is possible here. Suffice it to say that the developers of the tool may not be familiar with the experience of working under such strictures. They may have excused themselves from the necessity of using a similar tool themselves on the grounds that such tools do not apply to their type of system (but only to contractual development for aerospace applications, perhaps), or they may not fully understand the design philosophy underpinning the method that their tool is to support. In any event, the resulting tool may not improve productivity or quality.

Another power tool with which I am personally familiar, and which I strongly suspect to be much more widely and avidly used by its prospective user community than the upper-CASE tool, is the chessplayer’s database. Such a tool does not play the game of chess but provides a way of accessing large databases of previously played games and analyses of opening variations. Familiarity with openings and the styles of one’s opponents is crucial to the success of the modern tournament chessplayer. Several special-purpose PC-based applications have therefore gained wide currency among professionals and serious amateurs. They provide multiple views of games (static portrayals of positions, treelike renderings of variations, textual game scores, lists of games sorted by properties, etc.) and let the user directly manipulate these views so that they can perform application-specific

operations and comparisons (i.e. make moves and analyze positions, recall positions and games in which certain pawn skeletons or dispositions of pieces occurred, etc.). Domain expertise is essential to develop a system like this.

## 2.3. Off-the-shelf policy enforcement

Meeting schedulers are convenient systems to use as benchmarks for specification languages and requirements engineering methods, but many people who use meeting schedulers in this way do not go beyond the obvious optimization features. But meeting scheduling is not simply a optimization problem. It involves subtle issues of organizational practices. What happens when someone’s schedule is out-of-date? Who should give way when it is impossible to accommodate everyone’s preferences? Can a room reservation be bumped by another more important or urgent meeting, and if so, who judges importance and urgency? As Grudin has shown [3], these types of questions must be answered before a meeting scheduler can be successfully introduced into an organization. But different organizations will make different choices, and some may not be able to articulate a definitive set of principles, preferring to let the people involved work things out for themselves.

Despite the average designer’s domain knowledge, then, the type of thinking that is required here requires specialized analysis of potential target organizations. To be successful, an off-the-shelf product must be capable of being embedded in diverse organizations, and assumptions about *policy* pervade the specification and code of such a product. It is vital to be able to isolate these policy specifications so that they may be identified, challenged and changed during design and customized by the user organization.

## 3. Implications for Research and Practice

A number of new and developing methods and tools become more relevant for off-the-shelf systems.

*Just-in-time requirements determination.* In the case of off-the-shelf software, it becomes more important that designers record open questions whenever they occur so that they receive explicit attention. Assumptions should also be recorded for further analysis. For example, developers of the meeting scheduler must discuss assumptions about conflict resolution policies whenever resolution choices must be made during design. It is unrealistic to assume that these will have been definitively specified in a requirements document.

*Scenario analysis.* Because there is no customer to provide definitive answers to questions, it becomes important to evaluate likely system behavior in concrete situations. These range between informal and “inspirational” storyboards to systematic exploration of detailed transaction scenarios.

*Contextual exploration.* An off-the-shelf system is only as good as the accuracy of the contextual assumptions made by its designers. For example, the knowledge that spreadsheets are usually created as part of a more encompassing task and the results obtained by spreadsheet analysis are embedded in other work products leads to different features than would arise from a purely rationalistic analysis of spreadsheet properties. The HCI community refers to such contextual investigations as *task analysis*. Despite the narrower emphasis on using the results of task analysis to design effective interfaces, the requirements engineering community should explore the adaptation of task analysis methods.

*Model primacy.* Product design causes us to centralize the role of a model or small number of models of the application. Examples of these are the spreadsheet, the variation tree (in chess), the development method (in the CASE tool) or the model of collaboration and negotiation (in the meeting scheduler). Our general modeling languages (ER diagrams, state machines, etc.) are too general to help in discovering these, and do not represent them clearly. Designers thus often resort to freehand sketches that are too informal and ambiguous to express more than an impression of the model's properties. Bridging this gap between specificity and expressive power is an open problem.

*Design for context.* Designers of off-the-shelf products have to think about context of use and user needs but, because they do not have to deal with contractual issues, they do not articulate these concerns as "requirements engineering." Rather, they address these issues *throughout* design. Much of the research and process improvement work in this area has been carried out in the HCI community.

*Domain knowledge.* The "thin spread of domain knowledge" [1] is an issue for all types of system, but off-the-shelf products require that the domain knowledge reside in the project team as there is no external oracle to consult. Project staffing may not be a lively research question, but it is an important practical issue.

*Requirements for customization.* By having to deal with a large and possibly heterogeneous customer population, off-the-shelf products have to be user-customizable or extensible. The requirements for such products therefore occupy an object level and a meta-level. General modeling languages do not handle multilevel descriptions well. This is another open problem.

*Surfacing of objectives.* The requirements engineering community concentrates on requirements disambiguation and correctness. But attributes like time to market and

usability matter more for off-the-shelf products. When designers select features, they are therefore balancing multiple objectives, not merely reflecting what their customer tells them. We should work on making these properties explicit so that designers can reason about them.

*Design by modification.* Finally, designers of most off-the-shelf products modify previous versions of the system. Even when designing a new product, designers are constrained by previous examples of the same type. Our methods say very little about this type of activity.

## 4. Conclusions

Few of the research and practice priorities outlined above arise from the interface metaphor of requirements engineering. Off-the-shelf product requirements are not elicited or played back to the customer; they are proposed, invented, or designed.

None of this is to say that the differences between off-the-shelf and contract systems are absolute. Contractual development can also make use of assumption tracking, scenario-based analysis, explicit reasoning about design objectives, etc.

However, when there is no customer to ask, the style of development often changes radically. For too long the requirements engineering community has tacitly assumed that contractual development is the normal case. By considering off-the-shelf development too, we start to see requirements engineering and its relation to the rest of software and system development in a different perspective.

## References

- [1] Curtis, B., H. Krasner and N. Iscoe, "A Field Study of the Software Design Process for Large Teams," *Comm. ACM*, Vol. 31, No. 11, pp.1268-1287, 1988.
- [2] Grudin, J., "Why Groupware Applications Fail: Problems in design and evaluation," *Office: Technology and People*, 4(3): 245-264.
- [3] Lubars, M., C. Potts and C. Richter, "A review of the state of practice in requirements modeling," *Proc. IEEE Symp. Requirements Eng. (RE'93)*, San Diego, CA (January 4-6), IEEE Computer Society Press, 1993.
- [4] Petroski, H., *The Evolution of Useful Things*, Knopf, 1992.
- [5] Potts, C "The Other Interface: Specifying and visualizing computer systems", in G. Van der Veer, T. Green, J.-M. Hoc and D. Murray (eds.) *Working with Computers: Theory versus outcome*, Academic Press, 1988.