Wisdom: A Software Engineering Method for Small Software Development Companies

Wisdom, a lightweight software engineering method, addresses the needs of small teams that develop and maintain interactive systems. Small companies leveraging on their communication, speed, and flexibility can benefit from Wisdom's process, notation, and project philosophy.

Nuno J. Nunes, *University of Madeira*
João F. Cunha, *University of Porto*

Software process assessment and improvement models such as the Capability Maturity Model, the Quality Improvement Paradigm, and ISO 9001 are not suitable for small software developing companies. Several studies[1,2] concluded that small companies want to improve their process and product quality but face organizational, cultural, financial, and technical obstacles. In terms of organizational problems, small companies have difficulties forming an internal dedicated process improvement group. New practices can affect existing-product maintenance and client demand. Management's fear of high costs, delayed time-to-market, and low return on investment manifests in cultural problems. Software engineers often resist new methods, tools, technologies, and standards. Reaching maturity levels—required to comply with quality standards—is a long-term commitment. Financial problems often arise because allocating resources for quality groups and hiring consultants are costly. Modern methods and tools are expensive and require training (which temporarily reduces productivity). Finally, small companies can't avoid technical problems: tailoring complex software engineering methods and techniques that are designed for large companies is costly and time consuming. Low-end tools (for example, fourth-generation languages) lack integration and support for management tasks (for example, documentation, configuration management, and so on).

Besides raising these problems in small software companies, existing process assessment and improvement models fail to benefit from the strengths of such environments. For instance, small companies are usually more flexible and controllable and they react faster than large companies. Also, communication is usually enhanced in such companies both internally and toward external partners (clients, technology providers, and consultants).

We developed Wisdom (Whitewater Interactive System Development with Object Models) to address the specific needs of small development teams who are required to build and maintain interactive systems with the highest process and product quality standards. The Wisdom software engineering method has three important components:

- a software process based on a user-centered, evolutionary, and rapid-prototyping model that is ideal for constructing and maintaining interactive systems, such as those with a strong Web-user-access component;

- a set of conceptual modeling notations (based on a simple Unified Modeling Language subset) that support the modeling of functional and nonfunctional requirements (regarding user tasks and interactions in particular); and
- a pragmatic project management philosophy based on open documentation and tool-usage standards and requiring a flexible team with efficient, open communication channels between contractors, users, and developers.

Wisdom emerged from our experience with small companies that were developing software in ad hoc and chaotic ways, and it has evolved over the years.

Wisdom emerged from our experience with small companies that were developing software in ad hoc and chaotic ways, and it has evolved over the years. This article briefly presents our approach and outlines how companies can apply it in context.

The Wisdom Process

Small companies have limited resources and can't afford disasters or failure. But having "chaotic ways of working" doesn't mean they're working disastrously or failing. In fact, before software engineering researchers invented the spiral model and the iterative and incremental development process, small companies were working chaotically and making money. Having chaotic ways of working means taking a "just do it" approach (also known as the "Nike approach") to software development—evolving prototypes toward a finished product and always urging toward implementation. It also means having runaway projects, unrepeatable success, and low-quality products.

To overcome managers' and practitioners' cultural barriers, we need a model that adapts to their idiosyncrasies. Wisdom rationalizes "just do it" and builds on what characterizes small companies: speed, flexibility, and good communication. Our approach leverages those characteristics and is a natural evolution of the chaotic atmosphere into a user-centered evolutionary prototyping model.

To implement Wisdom, we start by introducing the iteration concept; that is, the development team states prototype objectives and evaluates them at the end of the iteration. This activity introduces a sense of completion and control, very effective for gaining developer and manager support. The key ideas are to raise the importance of progress measurement and identify the major intervention areas for Wisdom's process improvement techniques. At this stage, the iteration should reflect the development team's pace. The iteration's duration changes between companies and even between projects and development teams. Enforcing a predetermined duration can introduce undesired disturbance. The subsequent introduction of different activities during the iteration cycle will eventually increase the duration but will also reduce the total number of iterations, keeping (hopefully improving) the pace of the life cycle overall. That way, we can maintain time-to-market and support parallel product-development and maintenance life cycles, therefore minimizing the overall impact on the company.

Clearly identifying iterations raises the need to introduce techniques for stating objectives and evaluating prototypes. It is imperative that the developers see the advantage of using these techniques over the random hacking they usually perform. Introducing complex and time-consuming techniques will cause resistance, increase training costs, and temporarily reduce productivity, which will reflect on the company and endanger the improvement effort. With this in mind, we introduce participatory techniques[3,4] to aid requirements discovery and jump-start process improvement. Requirements discovery is a major problem software developers face and drives all subsequent activities. Participatory techniques are also a good excuse for introducing modeling notations such as the object-oriented UML. At this stage, we hold short requirements-discovery sessions with small groups of developers and end users. During these sessions and subsequent internal brainstorming meetings, we introduce the Wisdom notation for creating the requirements model.

We use the requirements model for stating prototype goals. Depending on the model's complexity and the team's maturity, we can also use it for strategic decisions; that is, for prioritizing development, identifying risks, managing resources, scheduling, and so on. This way, small companies can capitalize on enhanced communication with end users to capture critical process management information. For example, the companies identify and negotiate risks and priorities with end users during the participatory sessions, and the development team evaluates these risks and priorities for scheduling development and drive deployments (prototype evolutions). We also use the requirements model to evaluate the evolving prototypes, enabling the team to manage requirements and measure progress. We also conduct prototype evaluation in short participatory sessions with end users. During the sessions, the development team can reevaluate the risks and priorities identified when the prototype objectives were stated. If problems arise, developers can react fast, renegotiating the new development strategy with end users as needed. Throughout the process, we introduce different prototyping techniques, from mockups to low- and high-fidelity prototypes. We take special care to prevent the development team from using hi-fi prototypes in the early iterations. End users have problems criticizing hi-fi prototypes, and preventing the use of them can be an effective way to postpone the implementation rush.

This goal-setting and evaluation cycle, anchored on the requirements model, forms Wisdom's foundation. (Depending on the company, the complexity of its projects and models, and the development team's size, we can also introduce the analysis and design models.) For example, a small company that works on static Web design can work seamlessly under this simple model. This doesn't mean that analysis or design isn't performed in such environments, but that those activities are embedded in the prototype implementation, as they are when you write a small spreadsheet program. Therefore, development teams can expand the iterative cycle according to their needs. Once we've implemented the Wisdom process's foundation, the other activities emerge naturally and the improvement process proceeds. As the process matures, the models increase in complexity, and support activities for project management come to the forefront.

A diagrammatic, model-based approach is important in small companies because the development team can use working models to specify and document the software systems. Creating, managing, and maintaining documentation becomes a consequence of the development process and not an additional support activity. In Wisdom, we recommend an almost exclusive use of models to document the project. The models emerge from the participatory sessions as sticky notes that development team translates to a more formal specification in the subsequent wrap-up sessions. Sometimes, models require translation to textual descriptions for specific purposes (for example, a contract for the client), but they are not maintained as actual development-process descriptions. Some companies that were implementing ISO 9001 successfully used this approach to produce documentation required for quality practices.

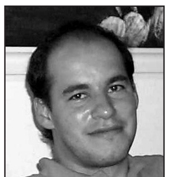
Wisdom doesn't require direct tool support (such as CASE tool support). Teams using our approach would benefit from a transparent, flexible integration of the UML's static diagrams with the relational schema or of the user-interface-specific models with a GUI builder, but modeling tools lack such integration with rapid-development tools. Nor do they support several development activities, such as process management activities (including development-task prioritization, progress measurement, and traceability). Due to the weak tool support, process management activities are usually performed manually or with word processing or spreadsheet software. Modeling tools also don't focus on the cognitive aspects involved in software development—such as opportunistic design, comprehension, and problem solving. With a lightweight approach, we could join process management features with the modeling tools, thereby reducing the need for additional tools (and avoiding the acquisition and training costs that new tools bring).

The Wisdom Notation

The Wisdom notation is a subset of the UML, a standard object-oriented language for visualizing, specifying, constructing, and documenting a software-intensive system's artifacts.[5] UML 1.1 has an overwhelming 233 concepts (84 basic and 149 diagram concepts).[6] UML predecessors (Object Modeling Technique, Object Oriented Software Engineering, and Booch) were methods, but the UML is process-independent, which explains its number of concepts and the need to put it into a method's explicit context.

Figure 1 presents Wisdom's four major workflows and the corresponding activities, models, and diagrams. As the illustration shows, Wisdom is based on seven models and uses four types of diagrams (two types each of structural and behavioral diagrams).

About the Authors


Nuno J. Nunes is a teaching assistant at the Computer Science Unit of the University of Madeira. His research interests include interaction design, object-oriented methods and languages, and lightweight software engineering techniques. He holds a five-year degree in informatics from the Instituto Superior Técnico of the University of Lisbon and an MPhil in software engineering from the University of Madeira. He is a member of the ACM, SIGCHI, SigSoft, and the IEEE Computer Society. Contact him at Universidade da Madeira, Dep. de Matemática, Campus Universitário da Penteada, 9000-011 Funchal, Portugal; njn@math.uma.pt.

João F. Cunha lectures at the University of Porto on information systems and databases. His research interests include decision support systems for operational transport planning, graphical user interfaces, and electronic commerce. He holds a first degree in electrical engineering from the University of Porto, an MSc in operational research from Cranfield University, and a PhD in computing science from Imperial College. He is chair of the IEEE Portugal section and a member of the ACM. Contact him at Faculdade de Engenharia da Universidade do Porto, Rua dos Bragas, 4050-123 Porto, Portugal; jfcunha@fe.up.pt.

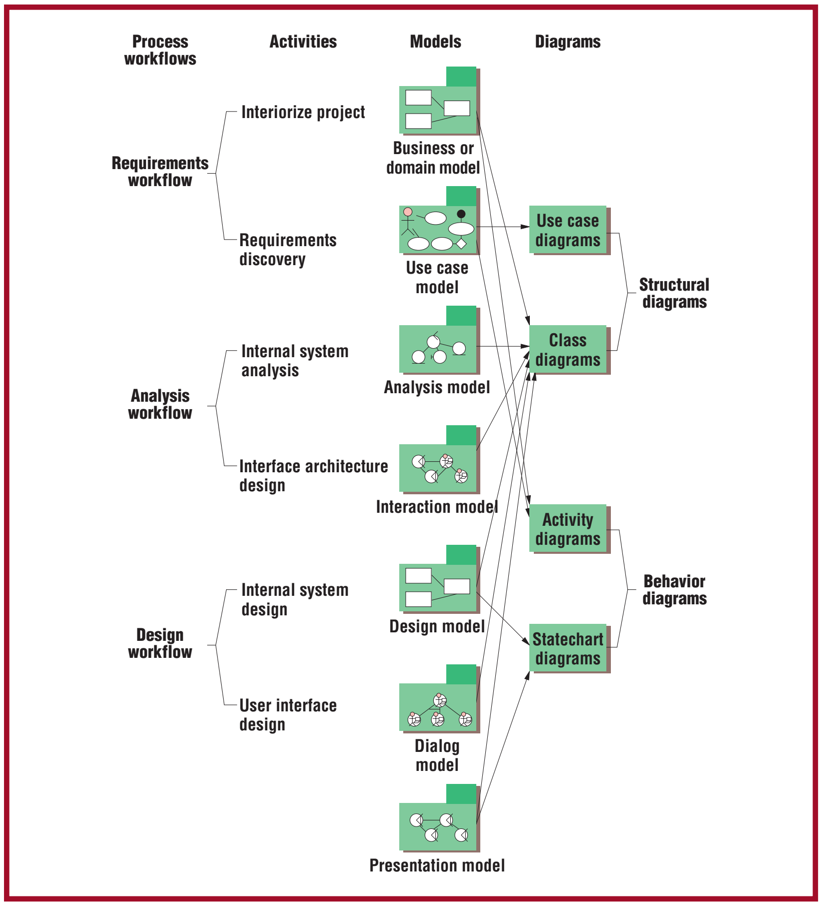


Figure 1. Workflows, activities, models, and diagrams in Wisdom.

We estimate that Wisdom uses approximately 39 basic and 29 diagram concepts—68 total (29% of the UML 1.1's total concepts).[6] Our proposal differs from existing UML-based methods and processes, not only because of the reduced number of concepts, but also because of its focus on the interactive aspects of the software systems. Usability is recognized as significantly impacting the quality, efficiency, and acceptability of the end products. Because small companies tend to focus their development contexts on custom or in-house development, we leverage the usability aspects in Wisdom. The interface architecture design and user interface design focus the usability aspects, prioritizing the development and scheduling of prototype evolutions.

Requirements Workflow

The requirements workflow aims for the development of a system that satisfies the customer, including the end user. This workflow includes creating the product's high-level concept, profiling the users, analyzing the users' tasks, creating the domain (or business model), and producing the requirements model. The accomplishment of these activities depends on the project's complexity and the development team's maturity and size. Regardless, all of the requirements workflow's outputs are a consequence of participatory sessions and the subsequent internal wrap-up. We only introduce the notation on a need-to-know basis.

The Wisdom notation expresses functional requirements, task flows, and nonfunctional requirements with use cases and activity diagrams. Because of the focus on interactive applications, we use the essential interpretation of use cases.[7] An essential use case is a complete, meaningful, and well-defined task of interest to the user. Essential use cases play a major role in our approach; they drive the entire development process, binding the evolutionary cycle together. In Wisdom, they also serve as the major source for finding and specifying analysis classes, task flows, interaction contexts, and tasks (user-interface-specific constructs). Additionally, they work as containers for nonfunctional requirements—a major problem in evolutionary prototyping approaches. In the requirements model, we use notes to attach nonfunctional requirements to use cases either at the model level (general requirements), at the use cases level, or at the activity level (annotating actions in activity diagrams). This approach reduces the number of artifacts to manage and keeps nonfunctional requirements at the hierarchical level at which the developers should consider them. Our approach enables the development team to manage several types of constraints, including technical, usability, and organizational ones. The Wisdom notation details use cases with task flows—a diagrammatic adaptation of an essential use case narrative—expressed in UML activity diagrams.

Task flows play a major role in Wisdom. A task flow corresponds to a technology-free and implementation-independent description of user intentions and system responsibilities in the course of accomplishing a specific task (the essential use case). They replace textual descriptions of essential use case narratives that prevent the development team from managing multiple success and failure scenarios. Developers use activity diagrams throughout the process to prototype and design the user interface. They foster reuse of user interface components and ensure the task and presentation models focus on the actual user tasks. They also guarantee that the user interface reflects not the application's internal structure, but the actual structure of use—a well-known

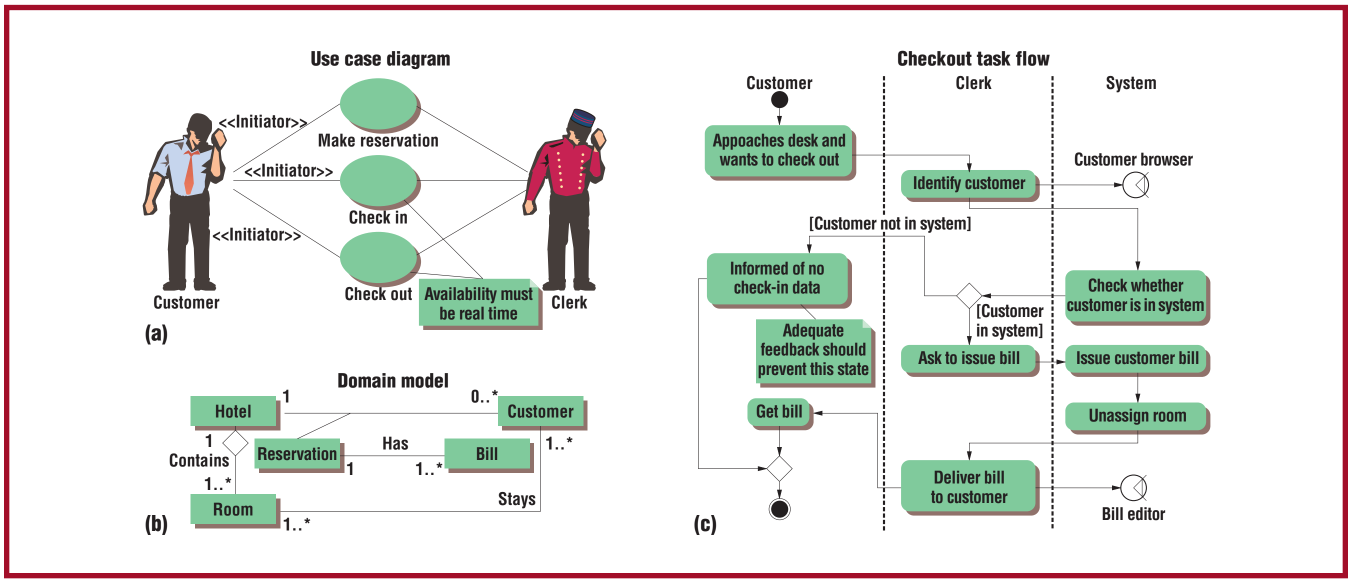


Figure 2. Example of Wisdom requirements artifacts for a hotel reservation system, including (a) a use case model with a requirement set and two users, (b) a domain model with the significant objects labeled, and (c) a checkout task flow demonstrating the connections of relevant actions.

problem that influences product usability.

Figure 2 illustrates some typical requirements-workflow artifacts for a simple, well-known example[3,8] of a hotel reservation system. Figure 2a shows the use case diagram with one example of a nonfunctional requirement attached to two use cases. Figure 2b shows a domain model representing the most important objects in the context of the system. Figure 2c shows an activity diagram detailing the checkout use case. Round rectangles denote actions, which correspond to user intentions in the customer and clerk swimlanes and to system responsibilities in the system swimlane. Another nonfunctional requirement is attached to the *informed of no check-in data* action (a low-level constraint). Also, under the system swimlane, we show how activities can be used to identify and detail interaction contexts. Interaction contexts are a Wisdom-specific extension to the UML; they play a central role, supporting interface-architecture and user interface design in our approach.

Analysis Workflow

The analysis workflow refines and structures the requirements described in the requirements model. The purpose is to build a requirements description that shapes the whole system's internal and user interface structure in the developer's language. In this workflow, the major activities are identifying and structuring both analysis classes and the Wisdom-specific interaction classes (interaction contexts and tasks). Analysis classes represent abstractions of domain concepts captured in the requirements workflow. They are focused on functional requirements and postpone the handling of nonfunctional requirements until the design phase. Analysis classes are divided according to the UML standard profile for software development processes, into entity (passive information), control (complex business logic), and boundary (external communication). This partitioning into three different types of objects distributes responsibilities, improving robustness, reuse, and change location.

In Wisdom, we expand the UML analysis framework, introducing two new user-interface-specific constructs organized in the interaction model.[9] Interaction spaces are a special kind of interaction class for modeling the interaction between the system and its actors; that is, they are responsible for receiving and presenting information and requests from and to the users. Interaction spaces have actions instead of operations, as well as stereotypical input and output elements that model, respectively, events the user produces and information presented to the user. Interaction spaces also have relationships that represent navigational or containment relations among them. The other new construct proposed in Wisdom is the task class stereotype. Tasks are a special kind of interaction class for modeling the structure of the dialogue between the user and the system. Task classes are responsible for task-level sequencing, maintaining consistency for multiple interrelated interaction spaces, and mapping back and forth between entity and control classes and interaction spaces.

In the analysis workflow, Wisdom leverages two major models. The analysis model reflects the system's internal structure and the organization of analysis classes (entity, control, and boundary) to realize the system responsibilities. The interaction model structures the user interface architecture in terms of

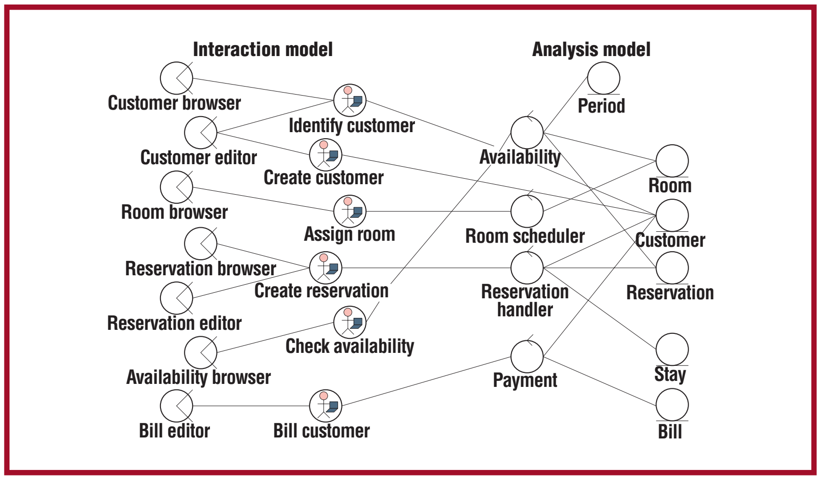


Figure 3. Example of Wisdom analysis artifacts for a hotel reservation system. Underlined circles are entity classes, circles with a top arrow are control classes, circles with a stick figure and a computer are task classes, and circles with a center arrow are interaction space classes.

interaction classes (interaction space and task) and how developers organize them to support the user intentions. The user interface accounts for approximately half the development effort of interactive systems. Moreover, in rapid-prototyping approaches, the user interface architecture usually reflects the application's internal structure, compromising coherence and consistency. The Wisdom notation extends the UML to support effective user interface design, fostering the development of an adequate architecture that reflects users' tasks and not the system's internal structure.

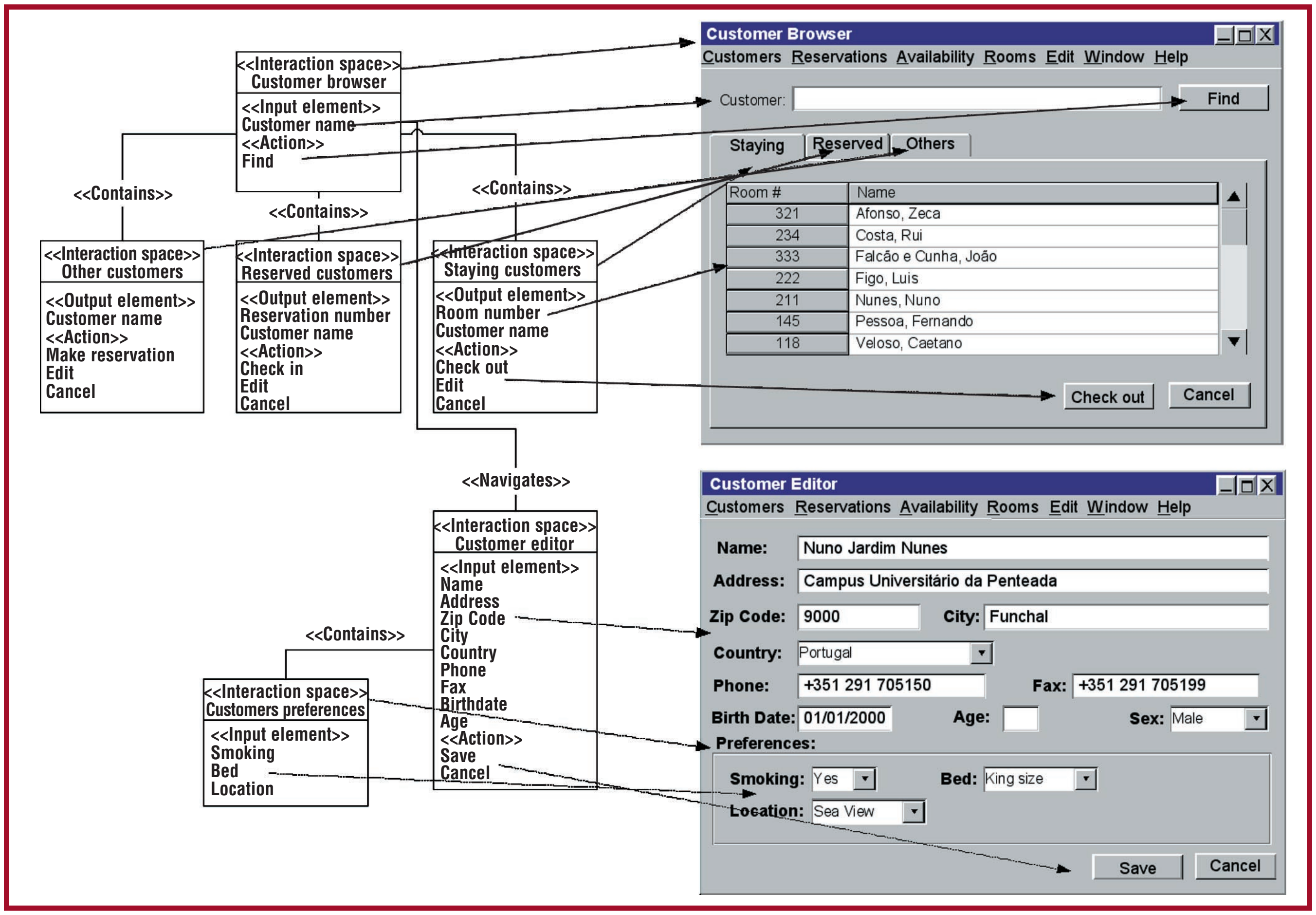
Figure 3 illustrates two analysis-workflow artifacts corresponding to the internal (analysis model) and user interface (interaction model) architectures of the hotel reservation system. On the right is the analysis model, with entity classes, control classes, and corresponding communication relationships. Entity classes correspond to passive information and usually coincide with domain types. Control classes represent coordination, sequencing transactions, and control of other classes that can't be allocated to specific entity classes. On the left is the interaction model, with task classes, interaction space classes, and corresponding communication relationships. Task classes represent specific tasks users must perform to complete the use cases. Interaction space classes correspond to abstractions of physical user-interface components—usually windows, forms, panes, and so on.

Design Workflow

The design workflow drives the system to implementation, refining its shape and architecture. While analysis focuses only on functional requirements, design also focuses on nonfunctional requirements. At this level, developers manage constraints related to the development environment (languages, databases, GUIs, operating systems, and so on). The major activities in this workflow reflect the incremental refinement of the analysis and interaction models. The Wisdom notation leverages the separation of the internal functional core from the specifics of the user interface, driving the design workflow, fostering reuse, and leading to robust implementations.

Technologies and tools available to small software companies are a major design influence. Small companies usually adopt fourth-generation languages, which influence the development environment in several ways. They include a relational database management system, a graphical interface builder, and a report-generation tool. They seldom support object-oriented programming and integration with high-end CASE tools (for example, modeling tools). Wisdom is independent of development and CASE tools. However, we recognize the importance of producing models that can be used, or smoothly transformed (even if not automatically), to feed available tools. Therefore, modeling in Wisdom represents a good return on investment, because tools available to small companies can sustain the modeling. Another major factor in design is supporting the evolutionary cycle. Once passed on to the development tools (and admitting lack of integration), design models will eventually become outdated. The process of reverse-engineering design models, feeding back the conceptual models, and maintaining traceability is then the main source of problems. In light of available technology and tools, lightweight notations that are easy to use and learn, such as the ones we propose in Wisdom, must support this process. The translation of Wisdom models into implementation technologies—such as the relational schema or a GUI—can be a simple, straightforward task. For example, the entity classes in Figure 3 outline the application data model and can be translated to a relational database structure following well-known transformation rules.[10] Because of the relative simplicity of such transformation, this is clearly one area where rapid-development tools could benefit from a direct integration with modeling tools, or even the use of a direct UML representation for the data model.

Figure 4 illustrates several Wisdom design artifacts for the hotel reservation system. On the left are five detailed interaction spaces. The resulting prototyped user interface is on the right. Dependencies from left to right show design decisions the user interface technology



<<Interaction space>> Customer browser
<<Input element>> Customer name
<<Action>> Find
<<Contains>>
<<Contains>>
<<Contains>>
<<Interaction space>> Other customers
<<Output element>> Customer name
<<Action>> Make reservation Edit Cancel
<<Interaction space>> Reserved customers
<<Output element>> Reservation number Customer name
<<Action>> Check in Edit Cancel
<<Interaction space>> Staying customers
<<Output element>> Room number Customer name
<<Action>> Check out Edit Cancel
<<Navigates>>
<<Interaction space>> Customer editor
<<Input element>> Name Address Zip Code City Country Phone Fax Birthdate Age
<<Action>> Save Cancel
<<Contains>>
<<Interaction space>> Customers preferences
<<Input element>> Smoking Bed Location
Customer Browser
Customers Reservations Availability Rooms Edit Window Help
Customer:
Find
Staying
Reserved
Others
Check out
Cancel
Customer Editor
Customers Reservations Availability Rooms Edit Window Help
Preferences:
Save
Cancel

Room #	Name
321	Afonso, Zeca
234	Costa, Rui
333	Falcão e Cunha, João
222	Figo, Luis
211	Nunes, Nuno
145	Pessoa, Fernando
118	Veloso, Caetano

Name:	Nuno Jardim Nunes
Address:	Campus Universitário da Penteada
Zip Code:	9000
City:	Funchal
Country:	Portugal
Phone:	+351 291 705150
Fax:	+351 291 705199
Birth Date:	01/01/2000
Age:	
Sex:	Male
Smoking:	Yes
Bed:	King size
Location:	Sea View

Figure 4. Transformation of the hotel-reservation-system design view model into a GUI.

constrains. Several different mappings of Wisdom interaction spaces exist. For example, the containment relationship between the customer browser interaction space and the three types of customer interaction spaces maps as a tabbed GUI element. Also, as illustrated in the figure, input elements map as GUI elements that the user is able to manipulate (editable text fields, combo boxes, and so on) and output elements as GUI elements that the user cannot manipulate (noneditable fields, icons, and so on). Actions (and navigational relationships, sometimes) map as buttons.

We've implemented Wisdom in several small software companies in Portugal. Our experience shows that small companies want to improve their practices and can take advantage of new developments in software engineering. Although we tailored our approach for small companies, small development teams within large development companies or user organizations can use it.

Wisdom has been efficient in a wide range of projects, such as Web site designs, interactive web applications, decision support systems, and distributed embedded systems.[11] Wisdom's user-centered perspective and the new user-interface-specific notational aspects have improved product usability with gains in terms of efficiency, user satisfaction, reduced complexity, and cost.

References

1. F. Cattaneo, A. Fuggetta, and L. Lavazza, "An Experience in Process Assessment," *Proc. ICSE '95*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1995, pp. 115–121.
2. E. Demirörs et al., "Process Improvement towards ISO 9001 Certification in a Small Software Organization," *Proc. ICSE '98*, IEEE CS Press, 1998, pp. 435–438.
3. T. Dayton, A. McFarland, and J. Kramer, "Bridging User Needs to Object Oriented GUI Prototype via Task Object Design," *User Interface Design*, CRC Press, Boca Raton, Fla., 1998, pp. 15–56.
4. M. Muller and S. Kuhn, "Participatory Design," *Comm. ACM*, Vol. 36, No. 3, June 1993, pp. 24–28.
5. G. Booch, J. Rumbaugh, and I. Jacobson, *The UML User Guide*, Addison-Wesley, Reading, Mass., 1999.
6. X. Castellani, "Overviews of Models Defined with Charts of Concepts, Information System Concepts," *Proc. IFIP WG 8.1 Int'l Working Conf. ISCO4*, Kluwer, Dordrecht, The Netherlands, 1999, pp. 235–256.
7. L. Constantine and L. Lockwood, *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley, Reading, Mass, 1999.
8. D. Roberts et al., *Designing for the User with OVID*, MacMillan, Indianapolis, Ind., 1998.
9. N. Nunes and J.F. Cunha, "Wisdom: A UML-Based Architecture for Interactive Systems," to be published in *Proc. DSV-IS 2000 Workshop*, Springer-Verlag, New York, 2000.
10. M. Blaha and W. Permelani, *Object-Oriented Modeling for Database Applications*, Prentice Hall, Upper Saddle River, N.J., 1998.
11. N. Nunes and J.F. Cunha, "Whitewater Interactive System Design with Object Models," to be published in *Object Modeling and User Interface Design*, M. van Harmelen, ed., Addison-Wesley, 2000.