ADOPTING USER-CENTERED DESIGN WITHIN AN AGILE PROCESS: A CONVERSATION

(Cutter IT Journal, October 2003)

William Hudson
Syntagm Ltd
Design for Usability
Abingdon, UK
william.hudson@syntagm.co.uk

eXtreme Programming and other Agile processes provide a middle ground between chaos and over-elaborate processes sometimes referred to as “death by documentation”. A particularly attractive aspect of the Agile approach for many teams is its willingness to accommodate change no matter how advanced development might be. However, this very flexibility can cause user interface design issues and ensuing usability problems.

Adopting a user-centered approach to user interface design can address these issues, as I hope the following simulated conversation between a user-centered design consultant and an XP team leader will explain.

XP: The team is pretty excited about the improvements we’ve seen in our development process since moving to XP, but our users are less enthusiastic. “Easy to use” is not the first phrase to pass their lips and our support desk is pretty overworked.

UCD: Agile processes focus a lot on making things better for the team, but they don’t really have much to offer users. Have a look at Figure 1.

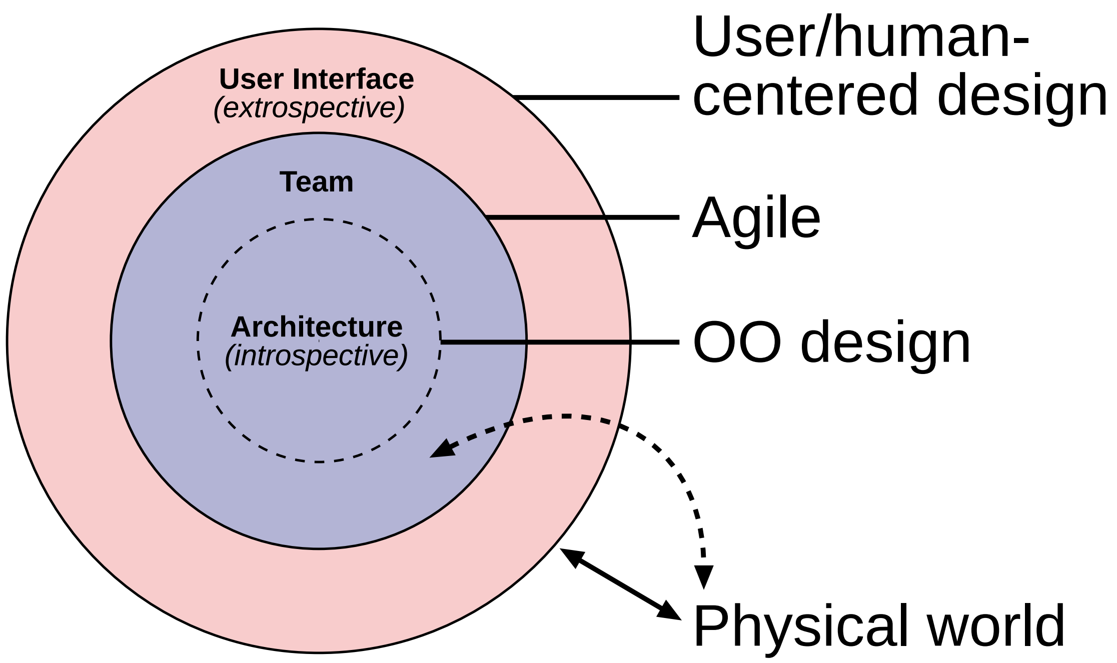

Figure 1, the relationship between OO, Agile and UI design

OO design concerns itself with the architecture of the underlying system – primarily in terms of objects and how they communicate with each other to realize the required functionality. This is a very introspective approach which *use cases*[1] were meant to remedy, but unfortunately they mostly caused confusion. OO design itself has nothing to say about how a project should be run or how to maximize return on effort.

Agile processes embrace some of the best practices from OO design, but replace use cases with user stories and address many of the team and project management issues that have plagued developers for decades. However, there is still no effort in trying to understand real users' needs and no mention of evaluating software through usability testing.

User or human-centered design[2] has virtually no overlap with OO design and Agile, but provides the missing interaction between those processes and the physical world. Notice that I am not saying that Agile processes have no interaction with the physical world, just that those interactions are not as extensive as many systems require (this is shown by the dashed arc in the diagram).

XP: But we have a user on the design team. Doesn't that help?

UCD: A user on the design team is better than no contact with users at all, but it is not the same as understanding real users' needs. We do this primarily by observation and interview, using an approach called *contextual enquiry*[3].

Another issue is that representative users (or user representatives) are frequently chosen for the wrong reasons. They may be

- domain experts;
- either popular or unpopular with their colleagues (depending on the method used to select them);
- either popular or unpopular with management (depending…);

---

[1] Use cases describe the interaction between the system and those entities outside it. In his first paper on the subject, Ivar Jacobson referred to these external actors as users. Soon after "users" became "actors" and although this change is not the only cause, use cases have been furrowing brows ever since.

[2] ISO standard 13407 refers to "human-centred" design, but really user-centered and human-centred are the same thing, ignoring also the use of British spelling in the standard's title. See [ISO/TC159 1999].

[3] A full discussion of contextual enquiry can be found in [Beyer & Holtzblatt 1998].

- the most technically competent;
- geographically well-placed to become part of the development team.

They are almost never typical users. Even if they were, they wouldn't be after just a few weeks. By that time they would

- be too familiar with the technical aspects of the project;
- have a vested interest in the solutions chosen;
- have forgotten what it is like to be sitting alone in a remote location with no one to help them figure out the latest release of this mission-critical application.

What we try to do in user-centered design is to make sure we understand the full *contexts of use*[4]: who the users are, their background, motivation, responsibilities, roles, the tasks they perform, the frequency and importance of each plus the physical and social environments in which they work.

XP: Surely that's overkill for most projects we're working on?

UCD: The list of issues is quite lengthy, but of course only a handful are relevant to most projects. One of the first things we try to do is to understand which of the contexts of use are important. For example, in some cases frequency of use is very important either for the system as a whole or for individual features. In others it may be that users are working in a stressful environment, such as a call centre, with lots of background noise and frequent interruptions.

XP: Come on, how would issues like that change what we're developing?

UCD: Take the example of frequent interruptions. There are several potential solutions:

- ensure that users have adequate feedback so they can pick up where they left off;
- allow incomplete information to be saved so they do not have to abandon an update;
- design the user interface so that multiple updates can be dealt with at one time.

[4] Described in detail in the ISO 13407 standard [ISO/TC159 1999].

XP: But aren’t you talking mostly about interface changes? Why couldn’t we just sort this out later?

UCD: The difficulty is that most development teams do not realize how much the user interface can affect the architecture of the system. It varies from almost negligible impact for simple form-filling applications to extensive impact for complex and highly-interactive systems. What typically happens is that well into the development process the team starts to realize the user interface is becoming very complex or that users do not understand what they have to do next. (This discovery would be quite early with regular usability testing but can be as late as product release when no real users are involved in the development process.) This can mean major architectural changes to the underlying system as well as the user interface. Designing the overall structure of the user interface and the way the application will present itself to users is something that needs to be done before the system architecture is decided.

Returning to the example we were just discussing, adding adequate feedback may or may not be an easy user interface change latter on, depending on the architecture of the system. Allowing incomplete data to be saved or multiple pending updates is almost certain to require major architectural changes: the back end system will have to be able to accept incomplete data; affected records will need to be brought to the user’s attention; maintaining data integrity in these scenarios can be a nightmare without a robust architecture to support it. These are not issues that you want to solve at a late stage of development.

XP: So what is involved? We’re using an Agile process to cut down on up-front design. Now you’re suggesting we need more.

UCD: Both software and user interfaces suffer from design erosion as they are modified over time. In software, especially in an Agile process with only minimal design, there comes a point when the objects making up the system need to be reorganized so that work can continue on a solid foundation. This is called refactoring. It works because no one outside the development team has to live the consequences of what in some cases are major upheavals. In any event, it is usually possible to draw a line around a group of objects and refactor just within that group so that there are no obvious external changes.

Refactoring is possible for user interfaces, certainly in the early stages before any broad release, but after that it becomes very difficult. Users need consistency. Changing the name or location of a menu item in a product that has been released can be very expensive. The primary tangible side-effects are

- documentation or on-line help changes;
- time users waste trying to find or understand the changed feature;

- time users waste because of mistakes caused by the change;
- the consequential cost of errors, especially if external customers are involved;
- increased support calls.

The intangible consequences include

- increased frustration and stress;
- loss of confidence and fear of change;
- helplessness, loss of control;
- resistance to change;
- confusion and inefficiencies as everyone within the company adapts to the changes as well.

Naturally these side-effects vary according to the type, quantity and frequency of changes, but generally speaking, alteration of the user interface needs to be carefully controlled.

XP: But we currently let the team change the user interface on an ad-hoc basis. I don't know how we could control this within an Agile process.

UCD: There are several components needed:

- contexts of use;
- personas;
- user stories;
- conceptual model;
- paper prototypes;
- usability testing.

Contexts of use I mentioned before. Once we have these we can identify some personas and then build a conceptual model from the user stories. *Personas*[5] are quite a powerful way of identifying the important types of

[5] For a lengthy discussion of personas, see [Cooper 2003].

users for our system and then characterizing them in such a way that the development team can relate to their needs and expectations. We all have a problem with seeing things from another person's perspective. Personas are one way of redressing the balance.

Here's a practical example. I was recently conducting usability tests on a government web site (to avoid embarrassment, I won't say which government). The site was intended to provide information on employment law and related issues for both employees and employers. One of my test participants was a builder. Let's call him Rob. Rob was not one to mince his words. The site's designers may have thought they had used terminology that most people would be comfortable with, but of course they were heavily influenced by the civil-service speak used by the government department concerned. Referring to a fictional persona for Rob, complete with photo-library image and pithy quotes from his usability session would have clarified the situation immediately. Rob was not impressed with the terminology. Some of his comments do not bear repeating in polite company, but it is not difficult to imagine the effect of the phrase "would Rob understand that?" if uttered in the middle of a design discussion.

Personas are used to represent important groups of users, but in a much more tangible and accessible way than user profiles. Most projects have only a handful of personas.

XP: Personas sound a little touchy-feely. Are they really worth the effort?

UCD: Like I said, it's all a question of perspective. If your team has no trouble putting themselves in their users' shoes (and you have usability tests to prove it), then maybe you needn't bother with personas. However, most developers end up creating applications for a mythical "perfect user". I happen to have an old drawing of one with me – look at Figure 2.

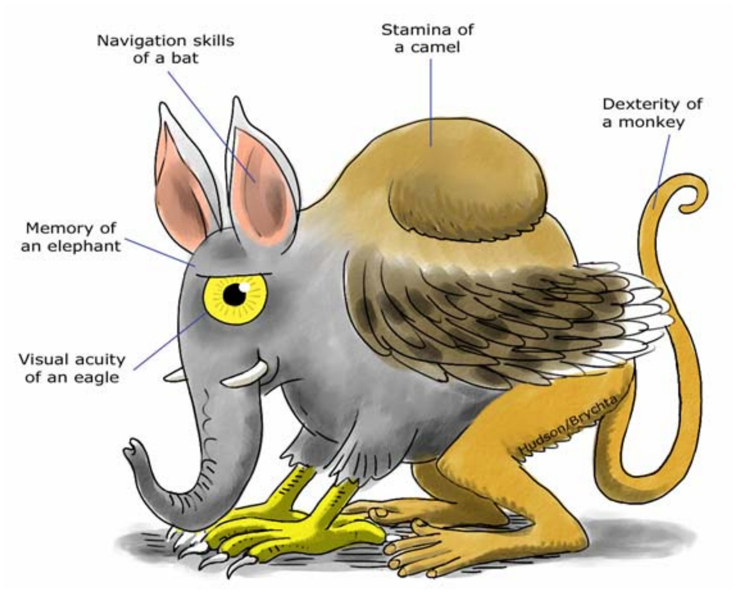

Figure 2, the mythical "perfect user"

The temptation is to believe that users

- are working in a quiet, ordered environment with no interruptions or distractions;
- will remember everything they have ever done on a computer;
- are motivated to solve any problems that come up without regard to their mental well-being;
- have no need for breaks, meals or sleep;
- only make mistakes through spitefulness;
- understand the internal workings of the system just as its designers do.

XP: Okay, I get the picture, but one thing you've just mentioned puzzles me. We are already producing user stories. Why are users still struggling with the resulting features?

UCD: Like use cases, user stories are intended to describe something useful that a user wants to do. The main difference between use cases and user stories is that use cases don't always deal with users – they can describe interactions between any arbitrary "actor" and a system – plus use cases frequently get very complex in trying to describe all possible outcomes. On the other hand, user stories are deliberately kept small and simple (with more stories created as needed to describe complex situations). Despite their differences, user stories suffer from many of the same issues that have caused confusion with use cases. The bottom line is that when writing use cases and user stories you need to ask yourself questions like these:

- Would real users do that?
- How would they know?
- Where does that information or understanding come from?
- Is the required behavior consistent?
- Does the story fit in with their work flow?
- Is it reasonable to expect that the whole story can be completed without interruption, or is greater flexibility required?

Some of these issues require an understanding of human-computer interaction to be addressed properly, not something that most developers have experience of. However, the savings in development time and effort are usually well worth having a specialist involved.

There is also a large underlying question of how much implementation detail should be included in user stories. If user stories are being written early in the process, too much detail can be a distraction and leads to premature design decisions. It is also difficult to simplify a design when there is too much detail involved – the “can’t see the wood for the trees” scenario.

Another reason for omitting implementation detail in early user stories is that we want to build a users’ *conceptual model* from them. This is a simple diagram showing the things that users need to think about (entities) and the relationships between them. See Figure 3 for an example.

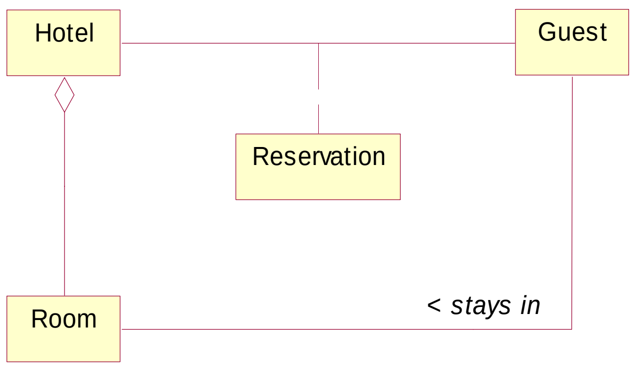

Figure 3, conceptual model for hotel check-in

XP: But that's a UML class model!

UCD: Yes and no. It uses the UML class model notation but they are conceptual entities – things that users need to think about regardless of how our application is implemented – not classes. Most of these entities will probably end up as classes in the implementation of the underlying system, but that doesn't concern us. Later on we add "views" – normally windows or dialogs – that show information and the controls users need to perform tasks. By the way, this first diagram is what Cook and Daniels[6] refer to as an essential model, although I prefer to call it conceptual since it represents our users' conceptual model.

XP: It looks a little simplistic.

UCD: At this stage, yes. But let's add some views and attributes in Figure 4.

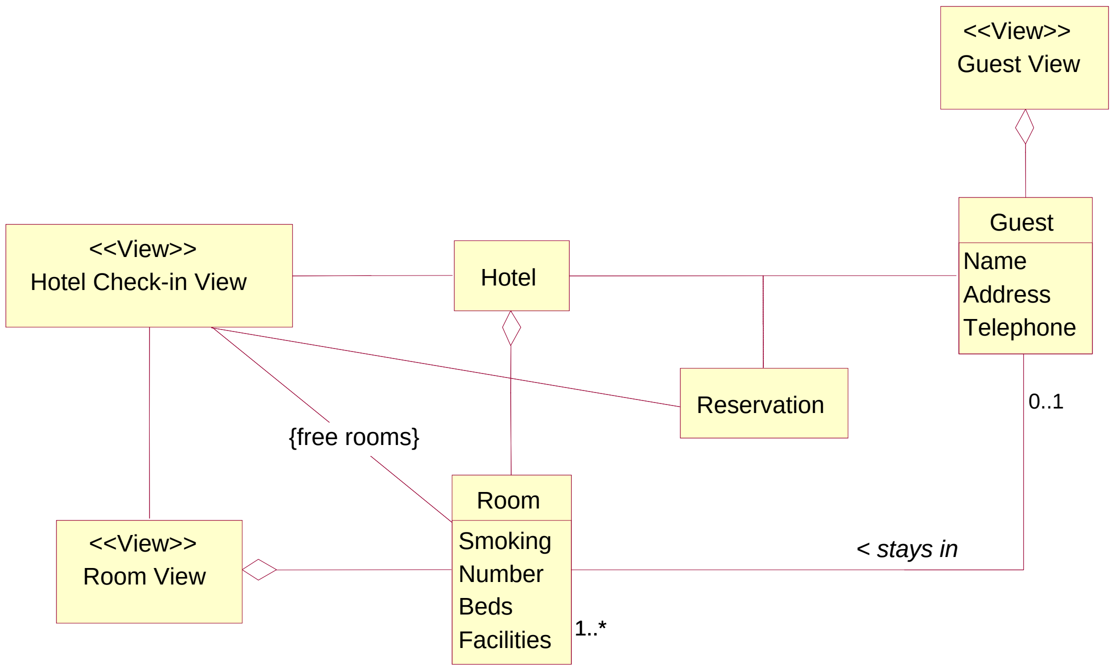

Figure 4, specification model for hotel check-in[7]

The views have been added in two different styles as either could be used at this level of modeling. The Room View and Guest View are shown in

[6] See [Cook & Daniels 1994]. Martin Fowler also discusses them as "perspectives" in [Fowler & Scott 2000]. He calls the "essential" model "conceptual" too.

[7] Based on [Roberts et al. 1998].

iconic style. This is a standard icon used by Rational Rose, normally for database views, hence its slightly odd appearance. The alternative is a little more explicit and is needed for the next level of detail. The Hotel Check-in View is shown with a stereotype of “View”. This lets us use the normal class notation and later add the required attributes and operations. The Room and Guest entities are examples of entities with attributes. Don’t forget that these are just the things that users need to know and to modify in order to reach their goals. We aren’t worried about how they are going to be represented in the computer. Also, the views do not always show everything that is present in an associated entity. An example would be that if we are developing a self check-in facility, there are quite a few details, such as the last occupier, that we would not want to show in the view. (It may well be shown in a manager’s view, though.)

One final thing. The line between the Hotel Check-in View and Room has the words “free rooms” shown in braces. This is called a constraint in UML and means that the view will confine itself to showing only rooms that are free.

XP: You mentioned another level of detail?

UCD: Cook and Daniels have three: essential (renamed conceptual), specification, and implementation. The implementation model of the user interface needs to detail what is shown and at least approximately how it is going to appear. Figure 5 shows an example for just the Hotel Check-in View. It lists the attributes of the view the user will need to see and a general description of how they will appear. The operations (only “Select room” in this case) correspond to user stories or use cases. If necessary, UML interaction diagrams could be used to explain how users interact with this view in detail.

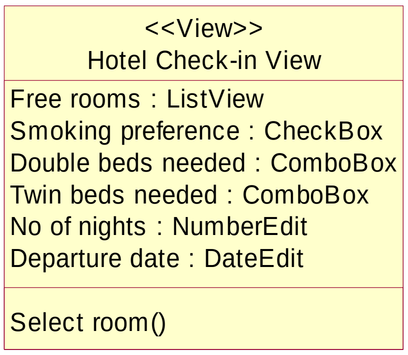
<<View>>
Hotel Check-in View
Free rooms : ListView
Smoking preference : CheckBox
Double beds needed : ComboBox
Twin beds needed : ComboBox
No of nights : NumberEdit
Departure date : DateEdit
Select room()

Figure 5, implementation model for hotel check-in view

XP: So now we have views and what they need to contain. How do we specify them visually?

UCD: This is where *paper prototyping*[8] comes in. We sketch the views on paper and do early *usability testing* with users. This doesn't involve an expensive usability lab or video recordings, just someone who is familiar with the process, plus the paper prototypes, some real users and a notebook (the paper kind!). Ideally you would have a usability specialist on your team for this as you can't use staff who have worked on the designs as they are too close to the issues.

XP: A dedicated usability specialist sounds extravagant.

UCD: Not really. There are a couple of issues here. The first is that it is important to have team members who have a background in human factors, human-computer interaction or usability. Although a certain amount of usability work can be on a do-it-yourself basis, you will still need someone who can be objective. The second is that there are lots of activities where HF, HCI and usability experience can be profitably applied: product design, user interface design, user guides, user assistance, help desk training, developer training and so on.

XP: Just one nagging doubt. Let's say we have a conceptual model elaborated to the implementation level of detail plus the paper prototypes. How do we deal with change later on in development?

UCD: Well, these things aren't meant to be immutable; they just need to be changed with care. The most important issue is that changes have to be considered by a user interface team, including an HF, HCI or usability specialist. They will make sure that any changes are consistent with

- the original conceptual model;
- other features elsewhere in the user interface;
- usability and accessibility guidelines;
- your own internal style guide

XP: And what about the myriad requests for new functionality we get throughout the lifecycle?

UCD: First you should make sure that any new functionality is for one of your personas (remember those?). No product can be all things to all users and every feature has a cost in terms of increased complexity and reduced usability. This may not be self-evident when you are looking at individual additions, but certainly adding dozens of features to a user interface can make life very difficult for users. If you have established that the feature is valuable to one or more of your personas, consider how much it changes your conceptual model for the product. Introducing new high-level concepts is not something that you want to do too often,

[8] For an extensive discussion of paper prototyping see [Snyder 2003].

whereas minor changes to the operations or attributes users can perform on an entity are going to have less impact.

XP: It all sounds plausible, but how do we get started?

UCD: For a new project, the approach is pretty much what I outlined earlier:

- contexts of use;
- personas;
- user stories;
- conceptual model;
- paper prototypes;
- usability testing.

You may not feel comfortable trying to do all this on your own so you may want to consider getting a user-centered design consultant to help out. He or she could get you started on the right foot and assist at less frequent intervals as the project progresses. Note that most of the activity is early in the project, with usability testing taking place throughout.

XP: Can we cut back on other forms of testing since we'll be adding this usability testing?

UCD: Only indirectly. There are many other savings that you should notice, but software testing and usability testing are quite separate. Software testing tries to make sure that your application does what *your team intended it to*. Usability testing addresses whether it does *what users need it to*. The difference is sometimes staggering.

The savings you should notice during development will be in the amount of reworking that is necessary. Reworking may be less expensive with Agile processes than with some others, but it still isn't free – especially if major architectural changes become necessary late in development.

XP: How would we address an existing application?

UCD: The first step would be to find out whether there are any serious usability problems and what might be needed to fix them. For this you would need one or more usability evaluations. I frequently recommend an initial expert review followed by usability testing with users as required. It should also be possible to get the reviewer to provide you with a "reversed engineered" conceptual model of the application – this can be compared with the model you design. Obvious anomalies between the models, plus problem areas identified by the usability evaluations are where your team will need to focus its attention. Again, a user-centered design specialist can help you make these decisions.

If the application needs major reworking or replacement, you should treat it as a new project for the purposes of most of the deliverables I mentioned. Paper prototypes are just about the only thing you could omit. It may seem contrary to the Agile approach to cover old ground in the form of user stories, but since they drive the rest of the process, it's important that they correctly reflect users' needs in terms of the contexts of use and personas that are important to you.

There isn't a lot of literature[9] available on the integration of user-centered design with existing software development processes, so working closely with a consultant is probably inevitable just now. I hope we will see start to see user-centered methods being adopted into mainstream software development in the not-too-distant future. Our ability to deliver usable systems depends on it.

Reference List

Beyer H. & Holtzblatt K. (1998) *Contextual Design: Defining Customer-Centered Systems*. Morgan Kaufmann, San Francisco, CA.

Cook S. & Daniels J. (1994) *Designing Object Systems: Object-Oriented Modelling with Syntropy*. Prentice Hall, New York, NY.

Cooper A. (2003) *About Face*. John Wiley & Sons Inc, New York, NY.

Fowler M. & Scott K. (2000) *UML Distilled: Applying the Standard Object Modelling Language*. Addison-Wesley, Reading, MA.

Hudson W. (2001) Towards Unified Models in Object-Oriented and User-Centered Design. In: M.van Harmelen (ed), pp. 313-362. Addison-Wesley, Boston, MA.

[ISO/TC159. Human-centred design processes for interactive systems. ISO 13407:1999. 1999. Geneva, Switzerland, International Organization for Standardization.

[9] See [Hudson 2001] for a more detailed discussion of user-centered design, UML and use-case driven methods.

Roberts, Dave, Berry, Dick, Isensee, Scott, and Mullaly, John. 1998. *Designing for the User with OVID: Bridging User Interface Design and Software Engineering,* Macmillan Technical Publishing.

Snyder C. (2003) *Paper Prototyping*. Morgan Kaufmann, San Francisco, CA.