# Crystal Clear

## A Human-Powered Methodology for Small Teams

Alistair Cockburn

Agile Software Development Series

Alistair Cockburn and Jim Highsmith, Series Editors

"Crystal Clear is beyond 'Agile' since I think that Crystal Clear is Agile + Flexible + Practical. 'Flexible and Practical' is important for us. This is not theoretical methodology but practical methodology that lots of people are doing. When I read *Crystal Clear,* I felt that this was easy to understand. This book helps us from software process hell to successful software development through practical examples and useful samples."

*—Basaki Satoshi, Schlumberger*

"Alistair Cockburn shows how small teams can be highly effective at developing fit-for-purpose software by following a few basic software development practices and by creating proper team dynamics. These small teams can be much more effective and predictable than much larger teams that follow overly bureaucratic and prescriptive development processes."

*—Todd Little, Sr. Development Manager, Landmark Graphics*

"I find Cockburn's writings on agile methods enlightening: He describes 'how to do,' of course, but also how to tell whether you're doing it right, to reach into the feeling of the project. This particular book's value is that actual project experiences leading to and confirming the principles and practices are so...well...clearly presented."

*—Scott Duncan, ASQ Software Division Standards Chair and representative to the US SC7 TAG and IEEE S2ESC Executive Committee and Management Board and Chair of IEEE Working Group 1648 on agile methods*

"*Crystal Clear* identifies principles that work not only for software development, but also for any results-centric activities. Dr. Cockburn follows these principles with concrete, practical examples of how to apply the principles to real situations and roles and to resolve real issues."

*—Niel Nickolaisen, COO, Deseret Book*

"All the successful projects I've been involved with or have observed over the past 19 or so years have had many of the same characteristics as described in *Crystal Clear* (even the big projects). And many of the failed projects failed because they missed something—such as expert end-user involvement or accessibility throughout the project. The final story was a great read. Here was a project that in my opinion was an overwhelming success—high productivity, high quality, delivery, happy customer, and the fact that the team would do it again. The differing styles in each chapter kept it interesting. I started reading it and couldn't put it down, and by the end, I just had to say 'Wow!'"

*—Ron Holliday, Director, Fidelity Management Research*

"*Crystal Clear* gives me both essential and practical ways of creating and navigating motivated software development teams. At the same time, it increases the quality of engineering life!"

*—Kenji Hiranabe, Project Manager, Eiwa System Management, Inc.*

## Technique 5. Daily Stand-up Meetings

The daily stand-up meeting is a short meeting to trade notes on status, progress, and problems. The key word is short. The meeting is not used to *discuss* problems, but to *identify* problems. If you find you are discussing how to solve problems in your daily stand-up, raise your hand and ask that problem resolution be dealt with right after the stand-up meeting with only those people who have to be there.

The term "daily stand-up" comes from the Scrum methodology (Schwaber 2002). The idea is to get rid of long so-called status meetings where programmers ramble on for five or ten minutes about a piece of code they are working on or where management people ramble about various project initiatives. To keep people from rambling, the convention is that the meeting takes place standing up, so people can't fall asleep, type on their laptops, or doodle on paper. We want them sensitive to the passage of time.

The authors of Scrum suggest that each person simply answers three questions:

- What did I work on yesterday?
- What do I plan on working on today?
- What is getting in my way?

The daily stand-up meeting is amazingly effective for disseminating information; highlighting when someone is stuck; revealing when the item someone is working on is too low a priority, off-topic, or adding unrequested features; and generally keeping the group focused and on track. It is simple, and it has been added into projects using every imaginable methodology with good effect.

## *Technique 6. Essential Interaction Design (Jeff Patton)*

Most authors on agile development don't say much about the external design of the system,[8] giving the impression we think it is unimportant. Speaking at least for myself, I don't write about it because I have so little personal experience with this specialty, even though I recognize its importance.

For that reason, I include here four adaptations of usage-centered design (Constantine 1999) that Jeff Patton created for the agile context. Jeff highlights and simplifies the activities needed to define, design, and test the system's *personalities*[9] and its *interaction contexts*. These techniques are well suited to projects where people are collocated and have to get a lot accomplished in a limited time. (Just because you have easy access to expert users doesn't mean you get to waste their time!)

A software system presents *personalities* to its various end users through its help, information, and speed. It might be designed to be fast and efficient, for example, or warm, friendly, and informative. These are the personalities it presents to the outside world. Most systems present different personalities to different users, depending on those users' backgrounds and needs. Except most designers don't develop the personalities in any deliberate fashion. They simply throw together whatever they have on hand, and the personalities of the system are just whatever happens to come out.

There are probably one or two *focal* user groups that the sponsors really want to see satisfied with the new system. The team needs to find out who those are, what they need to accomplish using the software, what personality is best suited for each, and be sure they are properly served by the system. They need to detect, write down, and periodically recheck who those are.

Jeff illustrates with the example of a chain of retail stores. The cashier uses the system daily, gets familiar with it, and prioritizes for speed in registering sales. The store consultant, on the other hand, won't use the system very often, won't be fluent with its interface, but knowing all aspects of the store's operations, will want to do more functions than the cashier. The cashier wants a fast-and-efficient personality to work with; the store consultant wants an informative and helpful one.

When working through their tasks, the users will want to see information in clusters, which the interaction design community refers to as *interaction contexts*. Part of interaction design is detecting the clustering of information suited to each user's tasks,

---

[8] Patton (2002) wrote an experience report about adding interaction design to agile development, and Cohn (2004) has a chapter on usage-centered design.

[9] My term. It seems to me the interaction design community is missing this higher-level concept of their work.

and looking for commonality across those. The developers will convert those to user interface settings, often with a set of information matching an interaction context.

*Essential interaction design* produces

- Shared understanding among the sponsors, users and developers who were present in the room as to the roles and tasks to be addressed by the system and the relative priorities of each
- Paper-and-pen collages ("reminding markers" in cooperative-game language) showing roles, tasks, and interaction contexts, marked with notes so that the team can deliver them in business value order.

Jeff articulates four techniques, in all.

1. Essential Interaction Design (the Workshop)
2. Deriving the UI
3. Usability Inspection (during Design)
4. QA Testing the System Personalities

The first is essential interaction design itself, done either near the beginning of the project or the start of an iteration. After that, he presents short techniques for designing the screens themselves, for reviewing the UI with the users, and for testing internally that the finished software is appropriate for the user roles that will be using it.

Here are the techniques as Jeff describes them.

## Essential Interaction Design (the Workshop)

The approach is based on the usage-centric design approach (Constantine 1999) except that this initial requirements elicitation and design process can be completed in a couple of hours to a couple of days. When an experienced facilitator is used, no advanced training of the business people is required. The techniques can be applied during initial requirements elicitation, when building incremental release plans, while defining the general form of the software, deriving user interface from use cases, acceptance testing, and even during end-user usability evaluation, if need be.

What follows is a step-by-step outline of the initial requirements elicitation and design process. The overall goal of the technique suite is to *hasten the discovery of requirements by bringing together people from each critical aspect of the project and allowing them to explain what they know to each other.*

*Step 1. Get the right people into the room.*

We're replacing weeks of interviewing and field research with conversation during this meeting, so it's critical the meeting involve members of each constituency the system will be serving, including those developing the system. Remember, if you're the facilitator, you aren't interested in what participants say to you, but rather what they say to each other.

In the workshop you will need selected key project stakeholders, selected key users, domain experts, and members of the development team, including test/QA people.

Eight to twelve people are good. It's sometimes hard to limit attendees to that number, but do it.

Set participants in a comfortable workspace with a big worktable and lots of wall space for hanging posters. Include markers, tape, 3″ × 5″ index cards, and lots of snacks to keep the less busy people occupied.

You're hosting a party.

*Step 2. Capture and annotate user roles.*

The term "actor" refers to any of job title, user role, or a mixture of both. We are after "roles," phrases that indicates what particular users' goals are. I always look for "thing doer" phrases to describe a role, or even, "*adjectived* thing doer," to make it more specific (only in English can a noun describing an adjective be used as a verb in the past tense!).

For example, we might find an "order taker" in the store situation, or a "special-order taker," or even better, a "hurried special-order taker." To handle customers who are angry because the special order is late, we find a "late-order researcher" or a "complaint handler." (People with such titles as sales associate and manager perform all the roles.) The extended role name makes it easier to keep the role clear enough so that anyone, with or without special domain knowledge, can understand what the role might be doing. It lets us know the person performing in this role is in a hurry and needs functionality and a user-interface that can support that.

Brainstorm user roles onto index cards—"card-storming," as Constantine and Lockwood (Constantine 1999) call it. After card-storming, refine your role list by removing duplicates and verifying the names are clear. Good names are important. We won't be relying on pages of documentation to describe each role, so the name is all you've got. Make sure it's expressive.

You are likely to name roles that you might consider stakeholders of the system, not necessarily users. Don't discard these. It's important to consider the goals of these people. It's important to ask how the software could support those goals. I often find that roles that might have been set aside as stakeholders actually need functionality in the system that supports proving to them that their interests were protected.

On the index card, write the role's primary goal or goals under the role name. What constitutes success for this user in using the system? What constitutes failure? We will keep checking that the requirements we capture really help user roles meet their goals.

Write only what constitutes success from the user role's perspective, not his boss's or some other stakeholder's. For example, a call center employee in a credit card service center, a "credit card application taker" might have a goal to take applications quickly enough and accurately enough to avoid a manager's attention. The manager, on the other hand, might want to improve speed and accuracy and reduce customer complaints. Other stakeholders may be concerned with up-selling credit insurance policies. The manager's and stakeholder's goals aren't necessarily those of the "credit card application taker." Make sure the appropriate goal finds its way onto the appropriate role card. If you want to capture the other stakeholders' needs, consider adding some roles, such as "efficiency watcher" and "up-selling watcher," and create role cards for them.

Write on each card a subjective estimate of that role's business value or importance, high, medium, or low. I mark H, M, or L on the lower left corner of the card.

Write on each card an estimate of how often the role will use the system. This can be high, medium, low, or hourly, daily, monthly, quarterly, yearly, or some other frequency you find suitable. I write these on the lower right hand corner of the card.

***Figure 3-11*** *The role modeling session. Be sure to listen for the conversations that occur during this process. (Courtesy of Jeff Patton)*

*Step 3. Construct the user role model.*

In this step you understand and mark the relationships, dependencies, and collaborations between roles. The result is the agile version of a *role model*.

Using a sheet of poster paper, the participants will cluster the user role cards on the table. The only important instruction to give is, "Try to cluster. Keep roles similar to each other close together, dissimilar roles farther apart." Once the model "feels" right, fix the cards to the poster paper with tape.

The roles probably got clustered together because they have similar goals or participate in the business process at similar times. Circle each cluster of roles. Label each with some indication of why those roles got clustered.

Draw a line from each cluster to other clusters where a relationship exists. I might, for example, draw a line from a cluster marked "order takers" to one marked "order fulfillers" and label it "sends orders to."

Make any other notes of interest on the model, possibly including a notable business rule, user role characteristic, or relationship across user roles.

*Step 4. Identify the focal roles.*

Participants now vote for the roles (not role clusters!) they consider being most critical to the success of the software (including also the roles where dire consequences

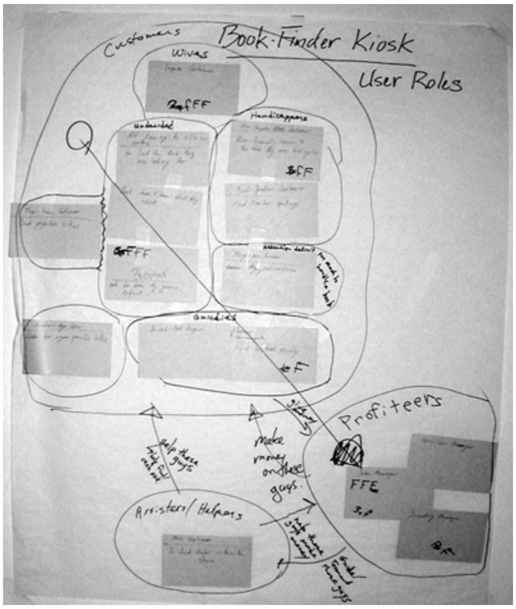

***Figure 3-12*** *A sample role-model. To those present during its creation, the marked-up sheet brings back a flood of conversation and information. (Courtesy of Jeff Patton)*