Interaction Design Meets Agility

Practicing Usage-Centered Design in an Agile Software Development Environment

Jeff Patton
jpatton@acm.org
Our Goal Today: Understand The Practice of Interaction Design on Agile Projects

- What is Interaction Design?
  - "Almost all interaction design refers to the selection of software behavior, function, and information and their presentation to users."
  - Alan Cooper, The Inmates are Running the Asylum

- This tutorial focuses primarily on discovery and selection of software behavior using Usage-Centered Design’s techniques in a collaborative work session
- Given an appropriately selected set of software behaviors, plan the incremental development and release of this software
- Discuss what other interaction design techniques might be applied during an agile development process
- What goals did you bring today?

Most software development methodologies seem to pick up after requirements are already resolved, ROI is justified, and budgets and deadlines are already set.

Recently popularized Agile Development approaches both allow and suggest that requirements be “emergent” – that they emerge over time from a close collaboration between customer team and development team. But, exactly how does the customer team arrive at appropriate functionality to implement? And, how exactly does the customer team communicate the important parts of those requirements to the development team?

This tutorial focuses on using a User Centered Interaction Design approach, specifically Constantine & Lockwood’s Usage-Centered Design, to select appropriate functionality, and then feed it forward into an agile development environment.

***

You’re Participating in a Process Miniature

- In about two hours we’ll run through a collaborative requirements work session
- You’ll get an idea how the process “feels” to practice
- You’ll begin to understand how the process may be practiced full size

"New processes are unfamiliar and perplexing. The longer the process, the longer before new team members understand how the various parts of the process fit with each other. You can speed this understanding by shrinking the time taken by the process. This is the Process Miniature."

"Run the entire process in a very short time period (a few minutes to a few days)."

-- Alistair Cockburn

You’ll learn collaborative interaction design and project planning by doing it – at least a little bit of it.

While participating, think about how you might attempt some of these approaches in your organization.

- What sorts of challenges would you encounter?
- What sorts of problems could it solve?

Also take note of the 3x5 card sorting activities.

- Are there other tasks where 3x5 card sorting and modeling might be useful?

***
A collaborative requirements session can be used when a software development project is in its very earliest stages to assess feasibility.

It can be used after requirements have been gathered to validate those requirements, and build models that aid in interaction and user interface design.

It can be used purely to help build understanding of product functionality within the team responsible for building the software.

A collaborative work session has the feeling of a many staged game. When starting a new project seems daunting, it’s relieving to lean on a repeatable process that quickly and easily makes sense of the work to be done.

### Goals For The Collaborative Requirements Work Session

- Build foundational understanding of project business requirements
- Build models that will support detailed design and testing, and effective priority management as the project progresses
- Build effective extended team communication
- Collaboratively discover requirements
- Build a plan that indicates approximately:
  - How much effort it might take to build the software
  - How much elapsed time it might take
  - How to most effectively break the software into incremental releases

### Anatomy of the Collaborative Requirements Work Session

<table>
<thead>
<tr>
<th>Meeting Preparation</th>
<th>Preparation Phase</th>
<th>Task Modeling</th>
<th>Incremental Release Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Domain Discussion</td>
<td>Role Modeling</td>
<td>Interaction Conflict Modeling</td>
</tr>
</tbody>
</table>

- Deliverables:
  - Informal Role Model
  - Informal Task Model
  - Context Model
  - Incremental Release Plan
  - Glossary
  - Business Rules
  - Other Lists and Issues
Whenever choosing to use collaboration as an approach to any activity, choose collaborators with different perspectives on the problem. Choose collaborators who bring a variety of skills. Include collaborators who are representative of the constituencies being served by the software and those involved with building it.

When eliciting requirements from a group of knowledgeable business people, I find what they have to say to me less interesting than what they have to say to each other. Choose participants that have interesting things to say to each other.

*The chicken says to the pig “Let’s start a restaurant.”*
*The pig asks “What will we serve?”*
*Bacon and eggs!” exclaims the chicken. “I don’t think that will work.” says the pig. “You’re participating, but I’m committed.”*

When involving a group of collaborators in any activity, try to make sure everyone understands who’s here to offer input and advice, and who’s actually responsible for ultimately delivering results. It’s a mistake to let a “participant” create the plan for those “committed.”

***

Set up the meeting room before participants arrive.

Give your design session participants “means, motive, and opportunity” to participate. You’ve provided some motive by bringing them here to participate. Paper, markers, and 3x5 cards give them a means to capture their ideas. Have plenty of these supplies ready and placed in the center of the table to allow people the opportunity to capture an idea when it’s fresh on their minds.

Important details will emerge from conversations during this design meeting. Questions will come up that can’t be answered. Sometimes a topic will come up that there just isn’t time to discuss. Record all of these things in feed-forward bins: simple lists posted on the wall. Doing this allows collaborators to continue to participate without being preoccupied by their potentially lost “great idea” or “serious concerns.”

***
Use this time to establish a general level of domain understanding. I like this process to take an hour or less.

There will be a temptation to dive into a detailed discussion of the business problem here. Don’t let that happen. Assuming there are at least a couple experts at the table, and at least one person who will be a legitimate user of the software, settle for short overview of the domain. The upcoming modeling activities will bring forward details important to us as the meeting progresses.

I struggle with telling groups that we’re doing a “preconception purge.” It somehow makes the information discussed in this part of the design session seem less valuable. While some ideas discussed may be crazy, impossible notions that could never be implemented, it’s surprising how many times what looked like a crazy notion initially became quite feasible later.

A valuable question to ask at this time is: “If the software were magic, what would it do?”

Sometimes when sitting down to design, someone at the table may have complaints about an existing system this software will replace or improve. This is a good time to hear these people out and record what they have to say. It will let them participate more effectively in the session, and it could come in handy later.

Make sure you record preconceptions on a feed-forward bin on the wall.
Our Design Problem:

Barney’s Books Information Kiosk Project

You’re on the in-house software development team for Barney’s Books. You and your team have been called in by the operations management team to discuss a new piece of software that Barney’s needs written. Before you can ask too many questions, the operations management people start telling you what they want.

They remind you that: Barney’s is a largish national retail chain. Their stores contain, on average, 20,000 square feet of floor space, that houses over 20,000 unique titles of books. In addition the retailer carries a number of related items such as periodicals, greeting cards, board games, stationary, and journals. While the store is conducive to browsing, it’s tough at times to find a particular item quickly. Customers who know what they want have a hard time finding it in the store. Often the title they’re looking for isn’t in stock, and in those cases Barney’s would like to special order it for them.

Today the only way to get help locating or special ordering an item is to wait in line for a cashier, or trap a sales associate in the aisles. Currently, sales associates hate to leave the safety of the cashier desk. A walk from the cashier desk to the back office can often take 10 minutes to a half hour as a result of all the folks stopping them to ask for help finding an item. The folks at the information desk stay pretty busy all day fielding questions as well.

The management of Barney’s believes they can enhance the customer’s experience at the store and ultimately sell more product by creating self-service touch-screen information kiosks within the store. At these kiosks, customers could get answers to their questions about the items they’re looking for and their locations. Optionally if the item isn’t in stock, they could arrange for it to be special ordered. If the item is in stock at another Barney’s store, they could find this out as well, and arrange for it to be set aside so they can pick it up at the other Barney’s store.

While discussing the domain consider:

- Who are the people who will be using this system?
- Why would they use it?
- What goals do they have?
- What kinds of activities might they do to meet their goals?
- What happens if they don’t meet their goals? – Who loses?
- What happens when they do meet their goals? – Who wins?
- Are there users who monitor and protect the interests of other users?
The types of customers coming into the store vary immensely. Some may be very comfortable with information kiosks while others may have never used one before. Some may be using the kiosk to quickly find a book title; others may be using it as an alternative way to browse the available book titles in the store.

Executives at the Barney’s corporate office believe they can enhance store sales by “suggesting” like-titles to customers looking for a specific title. They believe they can enhance store sales by encouraging customers to special order titles they don’t currently have in stock. They believe it would be valuable to know how often customers look up or ask for titles not currently in stock. They also believe they can reduce labor costs at the store a bit by allowing customers to help themselves. So these executives can feel comfortable they’ve spent their money wisely, they’ll expect statistics on how many people, by location, are using the kiosks. It would be valuable to know how many times customers looked closer at suggestions made by the kiosk. It would be valuable to know how many special orders were placed through the kiosk.

Your design and development team has been given the task to design and build this new information kiosk. Barney’s already has massive databases of the items they carry, inventory systems that tell them which and how many items are in stock at each location, and order entry systems to place special orders. Your team will need to integrate this information in a piece of kiosk software.

The operations management team doesn’t have specific functional requirements past those discussed here. They’re looking for an estimate from your design and development team that suggests the functionality they should build and the timeframe it will take to build it. They’d like to see a functional kiosk in stores as soon as possible. In fact, if you can’t get something functional in stores within couple months, we’ll find another team that can.

What will you build? And, how soon before we can see something running and put it into pilot stores?

Show me the money:
While considering requirement, many designers make the mistake of only considering direct users of the software. Those with concerns about the effectiveness of the software and/or its ROI are often set aside as “stakeholders.” Stakeholder concerns are often addressed in the software’s detail design by ensuring the software captures necessary information to meet stakeholder needs.

Consider promoting your stakeholders to users. If the stakeholder were a user, what could the software do to demonstrate to these stakeholder-users how much money it’s earning for them? Should stakeholder-users have access to current information on software performance? Should they receive warning when things aren’t going well?
User Roles Represent The People Interacting With Our Software

- Ways to identify people:
  - Job titles
  - Actors
  - User Roles
- Choose role names that unambiguously describe an activity or goal
  - Use the form "thing-doer" such as "Lettier Sorter"
  - Where it adds valuable information, use the form "adjective-thing-doer" such as "Disgruntled Lettier Sorter"

The first step to any user-centric design approach is identifying who the users are. When starting this process with business people, domain experts, or anyone who’s participated in a more traditional design approach, the first thing you’re likely to hear are job titles.

When a job title is suggested, ask the suggester what someone who holds that title would do with this software. Then ask if there are other people who hold different job titles that might also do the same thing. Usually the answer is yes. Folks quickly understand how ambiguous a job title is.

The thing-doer form is easy and unambiguous. Right now I’m a document-writer. If you’re reading this you might be a tutorial-handout-reader. If you’re reading this while I, in the role of tutorial-presenter am talking, you might be a disinterested-tutorial-handout-reader. It’s fun to make thing-doer names. And, as your design goes forward, you’ll appreciate how clearly these kinds of names communicate the goals and activities of the person in the role.

***

Brainstorm User Roles on to 3x5 Cards

**Goal:** Gather a superset of candidate user roles

- Choose a recorder or two depending on the group size
- Participants shout out role names, recorders write them down and toss them into the middle of the table
- The most important rule of brainstorming: do not discuss or comment on ideas, simply record them
- Before starting, decide how to stop
  - Time-box the activity
  - The microwave popcorn rule – 2-3 seconds between each idea pop

It’s the quantity of ideas that matter in brainstorming.

If you’re participating in this brainstorming activity, be sure to shout out something silly. I like using “lion-tamer.” It fits the thing-doer form perfectly, and it’s likely to get a laugh, or at least a groan. But it helps brainstormers not take their work so seriously and potentially helps them think a little outside the box.

You’ll often get role suggestions for people that clearly won’t be using this software. That’s good. Such suggestions may help someone at the table think of a person they might not have otherwise.

***
During brainstorming, quantity was important. At this next stage, we need to focus on quality.

- Make sure every role name is unambiguous
- Make sure goals are clear

Concise, unambiguous role names are important. The more clearly the name of the role describes it, the less supporting documentation you’ll need.

That said, there’s a tremendous amount of detail that can be added to a user role. Depending on the risks on your project and the time available, you may wish to go back and add more details. But, right now, using the 3x5 card forces you to note only the two or three interesting things about each role.

For more details that can be added to a user role, see Constantine & Lockwood’s Software for Use, or www.foruse.com.

Modeling is where things really get fun. This isn’t a structured modeling activity like you may have participated in before. Keep things informal. Let the group invent their own meanings for card positions, clusters, and relationship notations.

Think about what you can infer from the model.

Often roles higher on the model are perceived to be more politically important – is this true?

Often models are organized chronologically left to right – roles participating in the business process sooner appearing on the left.

Sometimes roles are arranged in a hub and spoke fashion where critical roles that everyone interacts with are in the middle, and all those roles interacting with them are arranged in a circle around the outside.

Does your role model fit any of these patterns?

Can you infer anything about the design of this software as a result?
There are many ways to describe the things people might do with their software. We’ll use the neutral term “task” here.

Sadly, many functional design processes begin here with tasks. Doing so potentially throws away the rich information about users and goals we got from doing the role model first.

Just as role names are important, the names you give your tasks are equally important. Think about the users’ intentions and goals when naming your task. When I walk up to an ATM machine, my intention isn’t to enter my PIN number; my intention is to get cash. If entering a PIN after inserting a credit card is what I have to do to identify myself to the bank, then, I’ll live with that. But, that’s the system’s concern, not mine.

Good task names trap the user’s intention or goal – not their specific actions. After understanding the user’s goals, you the designer can determine the most effective way to help them meet their goals.

***

We’ll trap the activities people are engaged in with short, clear names and goals. We’re deferring detail design to a time closer to when the software will be written.

When writing goals, be aware of goal level. For now, ideal goals are sea level or a bit above. A common mistake is for tasks to “wander” in goal level – some very high level, some very low. Tasks with high level goals are too ambiguous to plan with and estimate. When tasks are created with mostly low level goals, you’ll find that you have a huge number of them. At this point, details usually slow things down unnecessarily.

***
The task modeling activity feels about the same as the role modeling activity.

While engaging in the brainstorming part, it’s helpful to use the role model. Looking at the arrangement of roles and goals helps brainstormers derive a good list of tasks. If there are many roles, and likely many tasks, it may be helpful to brainstorm one cluster of roles at a time.

As with role names, task names are important. The more concise and unambiguous the task name, the less supporting documentation it will require.

I’ve found it to be valuable to add details to tasks or roles after they’re placed in the model. Seeing the task or role in context helps remind me of details I might have forgotten otherwise. Try adding details like: frequency of use, business value, or environmental concerns to the task card while they’re in the model.

You may have noticed that your tasks are arranged somewhat chronologically. I often observe that tasks that occur early in a business process are at the top left of a model, while those that happen later are at the bottom right.

Consider drawing a “time line.” Draw a line starting from the first task of a business process, through each subsequent required task, until you reach a final concluding task. Does this line help people better understand the business process?
The house metaphor:
Let’s say I’ve done a task model for my daily life. And, in that task model there was a cluster of the following tasks: make breakfast, eat breakfast, pack lunch, and listen to news and weather on the radio. Those tasks likely clustered because they’re all done very near each other chronologically. Or alternatively they may just “feel” similar. It would be more convenient for me if I could do all these tasks in the same physical place. That place is an “interaction context,” and I might label this one: “kitchen.”

What interaction contexts do you see in your software?

Interaction contexts can loosely be considered “modules” of the software. When creating lists of tasks, it’s helpful to group them by interaction context.

***

Many people consider Usage-Centered Design a variation of User Centered Design. Its authors would disagree. They submit that focusing on users and their unique concerns distracts the designer from the important thing: focusing on the software supporting the usage necessary to allow all users, no matter how unique, to meet their goals.

U-CD’s modeling practices work extremely well in a collaborative setting. U-CD’s modeling practices can be easily taught to novice practitioners. As seen in this tutorial, U-CD’s models can be built very quickly.

***
Placing focus first on people and goals makes it less likely you’ll come up with a “pedestrian” or “me-too” solution.

Defer coming up with tasks till you feel comfortable that you know the users and their goals well.

Defer prototyping user tasks until you feel comfortable you’ve arrived at the correct set of tasks to support your user’s goals.

The good folks at Cooper Interactive refer to the leap from information we know to design as “jumping the spark gap.” The more you know the shorter and safer that jump.

***

Several themes reemerge in the development of agile-type development approaches:

- Recognition of people and their individual talents
- Recognition that processes don’t guarantee success
- Recognition that collaboration between people who write software and people who pay for and use it is critical
- The use of iteration and incremental delivery to reduce risk and increase feedback

***
Agile Software Development and the Agile Alliance Was Born In 2001

- XP’s success acts as a catalyst
- Meeting of 17 practitioners at Snowbird, Utah, 2001
- All agree on the term “lightweight”
- All disagree on specifics
- All agree they have something in common
- Four principles of the Agile Manifesto represent this common ground
  - **Individuals and Interactions** Over Processes and Tools
  - **Working Software** Over Comprehensive Documentation
  - **Customer Collaboration** Over Contract Negotiation
  - **Responding To Change** Over Following a Plan

The Agile Alliance united people with similar concerns under one brand name.

Those concerns are encapsulated in the four agile principles. Those principles express a value system rather than a precise set of rules.

Do you agree or disagree with any of these principles? ***

Agility Is A Value System, Not A Set Of Practices Or Rules

- There’s no specific way to be or not be agile
- Agile is the approach to a method, not the method itself

- The Pornography Test:
  “I can’t define pornography, but I know it when I see it.”
  —Supreme Court Justice Potter Stewart, 1964

- Use the four principles to evaluate a methodology’s “agility”

Any methodology or collection of practices can be executed in an “agile” way.

Any documented agile methodology such as XP, Scrum, or FDD, can be executed in a way that isn’t agile.

The value system is what distinguishes an agile approach.

***
Many ad hoc methodologies have the characteristics of agile methods. With the definition of agility being more broadly known, many companies are coming out of the woodwork with their approaches that, strangely, have always been agile.

Choosing an appropriate level of formality is important to agile methodologies. An additional factor that’s important when determining how to practice a particular technique is time. Long lasting projects require more formal artifacts and more detailed documentation. In long lasting projects, these artifacts must outlive the participants of the project who may come and go over time.

Different parts of a project may vary in formality requirements by virtue of having more or less people participating or having more or less risk in the resulting software.

If your project has a modest formality requirement, expending any extra effort to add formality is simply added expense.

Looking back at the informal way we’ve engaged in U-CD modeling, can you see where there are projects where this would be inappropriate? Can you see projects where this would be appropriate?

***
Iterative development cycles succeed by observing iteration results and continuously adjusting approach.

Iterations aren’t just timeboxes. Iterations give regular opportunity for reflection: process assessment and change. Adjusting the process to match changing conditions reduces project risk.

Iterations are a technical mechanism to reduce risk and increase development predictability. Releases represent a bundle of features that can be deployed into a testing or production environment. Releases help reduce risk from a business perspective.

Releases help increase ROI on software development projects by deploying software that can begin to earn a return.

Frequent releases can help software development be self-funding. The ROI for each release can help pay for successive releases.
The tasks produced in our U-CD task model can serve as features in a release plan. It’s important to have a strategy behind your release plan. It’s important to devise a mechanism to measure the success of this strategy.

The “system span” is usually the first possible release. Oftentimes features in the system span can be “split” into smaller features. This has the affect of further reducing the time to deliver the first spanning release. Even if never released to users, the system span release has other valuable benefits:

- allows QA staff to get an early start on full integration testing
- helps validate the architectural framework sooner
- acts as a reference application to allow the addition of more developers to a project
Build a Span Plan Using 3x5 Card Modeling Techniques

- Arrange feature cards on the table:
  - Order left to right by time the task is executed in the business process
  - Order top to bottom by critically to the business processes
- The top row of this model are the features that make up the first spanning release:
  - Minimum time to release is the sum of these features estimated development times
- "Slice" feature groups horizontally to make up future releases:
  - Sum estimated development times to determine release dates

Span Planning leverages card based modeling techniques to quickly arrive at a project release plan that delivers a span first, then highest value thereafter.

An ideal plan is “course grain.” It allows for on-the-fly interpretation and adjustment. Consequently, it’s hard to ask for formal “sign-off” or agreement. You’re better off asking for an acknowledgement that the plan has been delivered and is understood.

Ask for agreement that the questions on the plan will be addressed as soon as possible.

Consider writing a collaboration plan: a plan that indicates how often the customers and developers agree to get together to review progress of the software and adjust plans.

These approaches build a collaborative rather than contractual relationship between those building and those buying the software.
“Responding to change over following a plan” doesn’t mean don’t plan, it means plan often. Just as with anything you do often, you’ll get faster and more accurate at it over time.

Don’t Follow This Plan – Use Continuous Feedback to Adjust It

“If you can’t plan accurately, plan often”
-- origin unknown

- Plans are built on estimation and assumption
- Use feedback looks to validate assumptions
- Adjust assumptions & estimation based on feedback
- Adjust plans based on adjusted assumptions & estimations
- Self flexibility

Use role and task models through the duration of the project.

As changes in scope or features occur, refer back to the role and task models. Have we learned something we didn’t know then? Did we underestimate the significance of a focal role or task? Did we omit an important role or task?

Adjust your role and task models just as you adjust your release plans.

Role And Task Models Inform Design From Planning Through The Duration Of The Project

- Role and task models help create detailed design
  - Detailing tasks result in detailed workflow
  - Role information allows the design of an appropriate user interface
- Role and task models help prioritization and planning
  - Focal roles and tasks require higher prioritization
  - High frequency tasks require more attention
  - Low frequency task or less important roles are candidates for de-scoping features
Usage-Centered Design practices can be used continuously through the development of a software project.
For further details on applying U-CD techniques see: www.abstractics.com/papers

Where To Go From Here

- Interaction Design Practices
  - Software For Use – Constantine & Lockwood
  - About Face 2.0 – Cooper
  - Writing Effective Use Cases - Cockburn
- Collaborative Design
  - Requirements By Collaboration – Gottesdiener
- Agile Development
  - Agile Software Development – Cockburn
  - Lean Software Development – Poppendiek & Poppendiek
  - Planning Extreme Programming – Beck & Fowler
- Software Economics
  - Software By Numbers – Denne & Cleland-Huang

© 2004 Abstractics LLC