Agile UX toolkit

Skills to successfully put user-centered design into agile projects

Desirée Sy & John Schrag

UPA 2011

Tutorial leaders

John Schrag
john.schrag@autodesk.com
@JVSchrag

Desirée Sy
desiree.sy@autodesk.com
@DesireeSy
Baseline

Where we’re starting from, where we’re going to

Baseline

• Agenda
• Who’s here?
• Agile terminology
• Transitioning problems
Agile UX toolkit (day-long tutorial)

morning

afternoon
Agile UX planning

Agile UX tactics

new activities

changes to familiar activities
Who are we?

“Adapting Usability Investigations for Agile User-Centered Design”

“Case Study of Customer Input For a Successful Product”

Autodesk (was Alias)

• Commercial, shrink-wrapped software
• 3D graphics, highly interactive
• Non-standard UI
• Users: Creative professionals
• Generative, open-ended tasks
UX team

Manager

Interaction Designers

Graphic Designers

UX developer (intern)
established UX team

example agile team
full-time*, co-located

*except for managers & graphic designer

full-time*, distributed

*except for managers & graphic designer
Baseline

- Agenda
- Who’s here?
- Agile terminology
- Transitioning problems
Agile manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

http://agilemanifesto.org/

Agile vs. waterfall

- Adapted from Cutter Consortium. “Agile Software Development”
Agile development cycles

“Story”
not the same old “user story”
“Customer”
In agile terminology, 

not (necessarily)

the person who buys your product

“Interaction Designers and Agile Development: A Partnership”
Lynn Miller. UPA 2006.

product owner

= 

product manager

+ 

dev lead

+ 

interaction designer
**product owner**

= every team member

**Other agile definitions**

Iteration/Cycle/Sprint/Stage (sorry!)
Scrum coach/Scrum master
Scrum meeting/stand up
Feature cards/Tasks
Retrospectives
Feature Points
Velocity
Working version
agile qualities
iterative
incremental
continuous
collaborative

Baseline

• Agenda
• Who’s here?
• Agile terminology
• Transitioning problems
Transitioning problems

1. No Iteration Zero
2. Planning is not iterative
3. Design/implementation not incremental
4. Iterations not really finished
5. Using same methods the same way
6. The full team isn’t available full-time
7. Part of the team is agile, part isn’t
8. Teams need help with agile processes
9. Political or management problems

Tools for these problems

1. No Iteration Zero
2. Planning is not iterative
3. Design/implementation not incremental
4. Iterations not really finished
5. Using same methods the same way
6. The full team isn’t available full-time
7. Part of the team is agile, part isn’t
8. Teams need help with agile processes
9. Political or management problems
“Cultural change isn’t easy. You need time, patience, & iteration.”

Baseline

• Agenda
  • Who’s here?
  • Agile terminology
  • Transitioning problems
Agile UX planning

Agile UX tactics

Planning framework
Iteration zero
BREAK
Iteration planning
Planning to change
LUNCH BREAK

Product planning
Release/Initiative planning
Planning across iterations
Planning within iterations
Iterating on the plan
Agile UX tactics

LUNCH BREAK

Parallel track workflow
Incremental implementation
Design chunking
Agile usability testing
Agile user feedback
BREAK
Agile communication

Timing of agile design
Incremental implementation
Incremental design

Agile communication

Planning framework

Big picture AND iteration focus
Agile UX planning

Planning framework
Iteration zero
Iteration planning
Planning to change

Product planning
Release/Initiative planning
Planning across iterations
Planning within iterations

Iteration zero
Where is the team going?
Before iteration zero

Products start before iteration zero, development starts at iteration zero

Before development begins, still need to:
- Identify a possible product concept
- Determine who would buy it
- Scope out the size of the market
- Investigate competitors

• This information is the input to iteration zero.
Project CHARIOT: Business goal background

- Make it easy for mobile device users to find, qualify, and donate to charities that are meaningful to them.
- Kiva-like business model; we are funded by optional add-on fees that we request – a small percentage of each donation.
- By providing easy access to efficiency information, we will drive charities to be more efficient.
- By making it easier to give effectively, we will empower our user to change the world.

Project CHARIOT: Persona 1—Sonia
Project CHARIOT: Persona 2—Drake
Iteration zero: What is it?

• Goal: Launch the development phase of the project

  - First iteration of an agile project
  - No coding done
  - No designs are created

“You need to maximize the amount of work not done.” (XP idiom)

Iteration zero: Why have one?

Solves issues with:

• Feeling that there is no “big picture”
• Not knowing what to work on first
• Doing unnecessary work that is not used
• Thrash – having the same arguments during every planning session

• The existence of a iteration zero is a strong predictor of success and satisfaction on an agile project.
Iteration zero: Who is involved?

Minimal participation:
- Product manager / BA (business rep)
- Design lead (end-user rep)
- Development lead (technology rep)
- Documentation lead (learning rep)
- Project manager
- QA lead

Iteration zero: What gets done?

Goal: Get *just enough* information and shared understanding to begin development

Actions:
- Share the Product Vision with the team
- Fill out product charter, including a Project Mission
- Identify risks & trade-offs
- Define roles & responsibilities

This information is the input to the initial planning session.
1. Define a shared Product Vision

- **Purpose:**
  - Alignment between all stakeholders on *what is being created*
  - Used to make decisions throughout the project
  - Keeps the project focused

- **Approach:**
  - Lead by Product Manager (Business)
  - One meeting, between 1-2 hours long
  - All stakeholders involved

- **Considerations:**
  - Once set, normally doesn’t change through the life of the product
  - Must be accepted by all stakeholders

---

**Product Vision = “DNA”**

1. **What is it?**
2. **Who is it for?**
3. **What is the primary differentiating reason people choose it?**
4. **Why is it better than its competitors?**

This should be *explicit* and *shared*
“Elevator statement”

For <type of person> who <has a particular need> <product name> is a <class of product> that <primary value proposition>

Unlike <competitor>, <differentiator>

(from Crossing the Chasm, by Geoffrey A. Moore)

Product Vision: Purpose

A good Product Vision will help you decide:
• What to drop from the backlog
• How to rank the backlog
• What “done” means
Who are these for? Why do they buy?

Rolex
Rolex knock-off
atomic clock
diamond Breguet watch
stopwatch
plastic LCD

Project CHARIOT: Vision
To enable mobile device users, anywhere, to change the world through the power of charity.
Project CHARIOT: “Elevator Statement”

- For anyone who owns a smart mobile device
- Chariot is a mobile application
- that lets you find and donate to any charity quickly, securely, and effectively.
- Unlike other charity apps, Chariot is not tied to a single cause.
- Unlike individual donation websites, Chariot brings together in one place the tools that you need to find, identify and qualify the charities you want to support.

“Everyone on the team should know the content of the Product Vision”
2. Define Project Mission

- **Purpose:**
  - Alignment on what is being created *during this release/initiative*
  - Used to make decisions throughout the project
  - Keeps the project focused

- **Approach:**
  - Lead by Product Manager (Business)
  - One meeting, between 1-2 hours long
  - All stakeholders involved

- **Considerations:**
  - Broken into tactics later
Mission (release goal)

Mission is always about the *business*.

It answers the question:

> “Why are you doing *this*, and not something else?”

---

Project Mission: Purpose

A good Project Mission will help you decide:

- What to drop from the backlog
- How to rank the backlog
- What “done” means for the Release/Initiative

---

<table>
<thead>
<tr>
<th>Business</th>
<th>Release</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Everyone</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Mission (release goal)

A Mission is feature-agnostic.

It can be re-stated as:

“We don’t know what features will do this, but [MISSION] will be the effect”

Example Project Missions

– “Remove the top obstacles that prevent people who download the product from purchasing it”

– “Neutralize competitive advantages of Product X”

– “Introduce our product to a new vertical market”

– “Increase the throughput of our users by 40%”
Project CHARIOT: Mission

Given the background information on Chariot (on handouts), come up with a clear mission for the first release (in 12 weeks).

• Remember that this goal should help by making business priorities clear for later decision-making.
• You can ask the Product Managers (today’s facilitators) any questions you like to help with this.

“The Mission adds the business context for the release/initiative. This informs design priorities, and defines success.”
3. Identify risks & trade-offs

- **Purpose:**
  - Alignment between all stakeholders for release/initiative
  - Easy to grasp fundamentals

- **Approach:**
  - Lead by Dev Lead
  - One meeting, between 1-2 hours long
  - All stakeholders involved

- **Considerations:**
  - Document kept as short as possible
  - Replaces the Product Requirements Definition

---

**Risks**

Risks can include

- Hardware limitations or opportunities
- Critical staffing or other resources
- Expected deliverables from other teams or vendors
- External business developments
- Reliance on unproven technology
**Project CHARIOT: Risks**

Given the background information on Chariot (on handouts), what are some of the risks and assumptions?

---

**Trade-off matrix**

<table>
<thead>
<tr>
<th></th>
<th>Fixed</th>
<th>Desired</th>
<th>Flexible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scope</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quality</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cost/Resources</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Only one of these items can be Fixed, and one Desired. The others are Flexible.

This chart is simple acknowledgement of *reality*. 
4. Define roles & procedures

- **Purpose:**
  - Get the team ready to go on day one
  - Determine roles and procedures
  - Set up physical environment

- **Approach:**
  - Usually lead by Dev Lead and Scrum Master
  - Usually done in the same meeting as the Project Data Sheet

- **Considerations:**
  - Good time to make changes based on last retrospective

---

<table>
<thead>
<tr>
<th>To do</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set up communication mechanisms</td>
<td>Physical vs virtual board?</td>
</tr>
<tr>
<td></td>
<td>Stand-up meeting time and format?</td>
</tr>
<tr>
<td></td>
<td>Preferences for email or Skype or MSN?</td>
</tr>
<tr>
<td>Identify project planning</td>
<td>Who will attend?</td>
</tr>
<tr>
<td>mechanisms</td>
<td>How will planning meetings be held?</td>
</tr>
<tr>
<td></td>
<td>How many iterations are planned per meeting?</td>
</tr>
<tr>
<td></td>
<td>How is Technical Debt handled?</td>
</tr>
<tr>
<td></td>
<td>Who owns what acceptance criteria?</td>
</tr>
<tr>
<td>Assign agile roles</td>
<td>Who is the scrum master?</td>
</tr>
<tr>
<td></td>
<td>Who fills the Product Owner role?</td>
</tr>
</tbody>
</table>
Agile UX planning

Planning framework
Iteration zero
BREAK
Iteration planning
Planning to change
LUNCH BREAK

Product planning
Release/Initiative planning
Planning across iterations
Planning within iterations

Initial planning meeting
What’s first?
Initial planning meeting: What is it?

• Goal: Create the *first speculative high-level schedule*

• This will be used by agile team to:
  – Clarify team trajectory
  – Prioritize Iteration One work for each team member
  – Identify & mitigate design, implementation, and dependency risks

• This will be iterated on by the agile team going forward
What does the initial plan look like?

– Story and feature cards arranged along timeline
– Feature cards have estimates
– Cards have acceptance criteria
– Iteration boundaries marked along the timeline
– Risks are identified
– Dependencies are identified
– There is detail only for Iterations 1 and 2

Tools and processes

• Tools
  – Planning board
  – Story cards and feature cards
  – Using the trade-off matrix

• Working with user stories
  – Placing stories in the timeline
  – Fitting stories into timeboxes
Planning board

– User stories posted
– Arranged from Now to Future along timeline
– Iterations 1 + 2 in detail
– Farther in future, much less detail
– Timeboxes labelled
– Completely public for all team members
– Always up to date
– Artifact of (not substitute for) communication!

Example: Planning board
Example: Planning board detail

Story cards and feature cards*

- Just-in-time detail for stories
  - Initially, used only one type of feature card
  - Grouped in tasks/activities (affinity diagram)

- But there were problems with this:
  - Cards with different granularity
  - Some stories were hard to estimate
  - Some stories weren’t stories, but work detail
  - Some stories weren’t for users, but team

*indicates a particular way that our UX team developed of working; this isn’t meant to represent a general agile practice
Story cards and feature cards*

- We used different types of cards:
  - Feature cards have tight estimates
  - Owned by developers
  - (If needed) story cards are higher-level
  - Story cards are “broken” into feature cards

- Current iteration + 1 are feature cards
- Later iterations are either feature or story cards

Types of story cards*

User stories with estimates:
- Known strategy and tactics, but loose estimates

User stories that can’t be estimated:
- Design risk (“Design spike”)
- Engineering risk (“Spike”)\(^1\)
- External dependencies\(^1\)

\(^1\)Won’t be discussed today
Using the trade-off matrix

<table>
<thead>
<tr>
<th>Fixed</th>
<th>Desired</th>
<th>Flexible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scope</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quality</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cost/Resources</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Impact on planning board of fixed or flexible:
  – Schedule
  – Resources
  – Features
  – Stability / quality

Trade-off matrix: Flexible schedule
Trade-off matrix: Flexible resources

Trade-off matrix: Flexible features
Trade-off matrix: Flexible quality

Cycle 1
- Flexible quality
- 10:00 a.m. to 30:00 a.m.
- 30 FP

Cycle 2
- Flexible quality
- 2-15 November
- 30 FP

Cycle 3
- Flexible quality
- 16-27 November
- 30 FP

Cycle 4
- Flexible quality
- 16-26 December (Vacation)
- 15 FP

Cycle 5
- Flexible quality
- 4-18 January
- 30 FP

Cycle 6
- Flexible quality
- 30 FP

Trade-off matrix: Flexible quality

Cycle 1
- Flexible quality
- 10:00 a.m. to 30:00 a.m.
- 30 FP

Cycle 2
- Flexible quality
- 2-15 November
- 30 FP

Cycle 3
- Flexible quality
- 16-27 November
- 30 FP

Cycle 4
- Flexible quality
- 16-26 December (Vacation)
- 15 FP

Cycle 5
- Flexible quality
- 4-18 January
- 30 FP

Cycle 6
- Flexible quality
- 30 FP

Drop work on a feature related to quality.
Placing stories on the timeline
Factors that affect where stories are on timeline

Timeline: Product requirement

• Consult Vision and Mission

• Look at both story cards and feature cards within stories

• Prioritize along timeline
Agile UX planning

Planning framework
Iteration zero
BREAK
Iteration planning
Planning to change
LUNCH BREAK

Product planning
Release/Initiative planning
Planning across iterations
Planning within iterations

‘Capability thinking’

• Clustering, ordering, and determining thresholds of problems within the backlog is the biggest contribution of design to Release Planning.
• Keep thinking at the problem definition level, rather than jumping to solutions.
Timeline: User’s workflow completion

• Cluster
  – Group Stories within a Capability
  – Identify user workflows with >1 Story

• Order
  – Place the essential Capabilities earlier
  – Place essential Stories in a Capability earlier
  – Arrange Features in Story in importance within iterations

• Thresholds
  – Determine the “minimum viable” version of the Capability
    (minimum Stories and Features in Stories)
  – Track “minimum Story set” for each Capability

Timeline: Development risk

• Technology investigations (“Spike”)
  – Time limits instead of estimates
  – Goal: Break the Story into features with estimates
  – Often have downstream dependencies
  – Tend to be placed in early iterations
  – Marked with red card*
  – Place in timeline at time when research occurs
Timeline: Dependencies

- Technology dependencies
  - The more stories depend on a feature, the earlier it’s placed

- External dependencies*
  - Marked with yellow card
  - Place in timeline when team will know how to break it into feature cards with estimates
  - (Work on this story occurs in earlier iteration than where placed.)

Timeline: Design risk

- Design investigations (“design spike”)
  - Risk = Hard to estimate
  - Low design risk features go in early iterations
  - Multi-iteration design investigations marked as blue cards*
  - Place in timeline when design is complete enough to estimate on feature cards
  - (Work on this story occurs in earlier iteration than where placed.)

- Relative ordering of design cards
  - Exception to the “dependency” heuristic is design dependency
  - Place a design “hub” later to reduce UI re-work
  - Balance design risk with workflow completion & product requirement
Project CHARIOT: Initial Plan

- There is time for 4 iterations of 3 weeks each.
- There are 3 developers:
  - Mary – (velocity 0.66, or 10FP in 3 weeks)
  - Omar – (velocity 0.53, or 8FP in 3 weeks)
  - Ivan – (velocity 0.46, or 7FP in 3 weeks)
- Omar is on vacation in iteration #3, and so will only be completing 2FP of work that iteration.
**Project CHARIOT: Initial Plan**

<table>
<thead>
<tr>
<th>Iteration 1:</th>
<th>Iteration 2:</th>
<th>Iteration 3:</th>
<th>Iteration 4:</th>
</tr>
</thead>
<tbody>
<tr>
<td>25 FP</td>
<td>25 FP</td>
<td>19 FP</td>
<td>25 FP</td>
</tr>
</tbody>
</table>

**Project CHARIOT: Lay out Planning Board**

Based on the charter, the chosen persona, the release goals, the development resources, and the backlog, lay out the story cards for this release.

If a story card does not have a time estimate on it, you can create a 1-day task to estimate the feature and put that on the schedule.
Placing stories on the timeline
Factors that affect where stories are on timeline

Agile UX planning

Planning framework
Iteration zero
Iteration planning
Planning to change

Product planning
Release/Initiative planning
Planning across iterations
Planning within iterations
Iterating on the plan
Planning to change
How design contributes to iteration planning

Pre iteration zero → Iteration zero → Initial planning meeting → Iteration one → Planning meeting → Iteration two → Planning meeting

We’ll discuss how you work during iterations in the afternoon
Iteration planning: What is it?

- Goal: Get each team member ready to work on next iteration
  - Calculate development velocity
  - Add new information (to re-prioritize cards)

Planning meetings: Who is involved?

- Minimal participation:
  - Product manager/BA (business rep)
  - Design lead (end-user rep)
  - Development lead (technology rep)
  - Documentation lead (learning rep)
  - Project manager
  - QA lead
Planning meetings: Why be involved?

- **Product designers:**
  - Represents the end-user
    - Brings knowledge of workflows, not just features
    - Brings feedback from the field
  - Responsible for their designs
    - Sets priorities on design aspects
    - Makes sure that important pieces of a design are not dropped
    - Makes sure dependencies are noted
    - Makes trade-offs between designs (drop aspect on one to be able to get another)
  - Responsible for their commitments
    - Places design risk cards in the proper iteration
    - Make sure user-facing working version demonstrates a workflow

Preparing for the planning meeting

**Design:**

- Get feature cards for potential new work based on
  - Designs for next iteration
  - Customer feature requests
  - Usability feedback collected during this iteration

**Development:**

- Calculate velocity numbers (burn down rate)
- Make estimates for feature cards
During Planning Meeting

Design brings:
- Prioritized feature cards with estimates

PM brings to the meeting:
- Velocity (burn down rate)

PM brings to the meeting:
- Priority changes
- Release goal changes

Planning meeting: Between iterations
– Planning happens around the planning board

<table>
<thead>
<tr>
<th>Iteration 2</th>
<th>Iteration 3</th>
<th>Iteration 4</th>
<th>Iteration 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature card</td>
<td>External dependency card</td>
<td>Design risk card</td>
<td>Feature card</td>
</tr>
<tr>
<td>Feature card</td>
<td>Feature card</td>
<td>Feature card</td>
<td>Design risk card</td>
</tr>
<tr>
<td>Feature card</td>
<td>Feature card</td>
<td>Feature card</td>
<td>Dev risk card</td>
</tr>
<tr>
<td>Feature card</td>
<td></td>
<td>Feature card</td>
<td></td>
</tr>
</tbody>
</table>
Planning meeting: Between iterations
- Iteration 3 was planned out in detail in the last planning meeting

<table>
<thead>
<tr>
<th>Iteration 3</th>
<th>Iteration 4</th>
<th>Iteration 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature card</td>
<td>Design risk card</td>
<td>Feature card</td>
</tr>
<tr>
<td>External dependency card</td>
<td>Feature card</td>
<td>Feature card</td>
</tr>
<tr>
<td>Feature card</td>
<td>Feature card</td>
<td>Design risk card</td>
</tr>
<tr>
<td>Feature card</td>
<td>Feature card</td>
<td>Dev risk card</td>
</tr>
<tr>
<td>Feature card</td>
<td>Feature card</td>
<td></td>
</tr>
</tbody>
</table>

Planning meeting: Design risk cards
- Design risk card in iteration 4 can now be broken down

<table>
<thead>
<tr>
<th>Iteration 3</th>
<th>Iteration 4</th>
<th>Iteration 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>External dependency card</td>
<td>Feature card A</td>
<td>Feature card</td>
</tr>
<tr>
<td>Feature card</td>
<td>Design risk card</td>
<td>Feature card</td>
</tr>
<tr>
<td>Feature card</td>
<td>Feature card B</td>
<td>Design risk card</td>
</tr>
<tr>
<td>Feature card C</td>
<td>Feature card</td>
<td>Dev risk card</td>
</tr>
<tr>
<td>Feature card</td>
<td>Feature card</td>
<td></td>
</tr>
<tr>
<td>Feature card</td>
<td>Feature card</td>
<td></td>
</tr>
</tbody>
</table>
Planning meeting: Design risk cards
– Broken out cards are placed in iterations based on workflow and priority

Card movement
– Same forces that acted on the cards in the initial planning meeting affect new cards in the between-iteration planning meetings.
Planning meeting: Usability feedback

– Field work adds change cards and new feature requests

Iteration 3
External dependency card
Feature card
Feature card
New feature request
New feature request
New feature request

Iteration 4
Feature card A
Feature card B
Feature card
New feature request
New feature request
New feature request

Iteration 5
Feature card
Design risk card
Dev risk card
Feature card C

Planning meeting: Usability feedback

– Realign the board with the new cards

Iteration 3
External dependency card
Feature card
Feature card
New feature request
New feature request
New feature request

Iteration 4
Feature card A
Feature card B
Feature card
New feature request
New feature request
New feature request

Iteration 5
Feature card
Feature card
Feature card
External dependency card
Feature card
Project CHARIOT: Planning for Iteration 2+

Based on the new information (in your handouts), re-plan iterations 2, 3, and 4.

Pre-iteration zero → Iteration zero → Initial planning meeting → Iteration one → Planning meeting → Iteration two → Planning meeting

✓ ✓
LUNCH

Agile UX planning

Agile UX tactics
Agile UX planning

- Planning framework
- Iteration zero
  - BREAK
- Iteration planning
- Planning to change
  - LUNCH BREAK

- Product planning
- Release/Initiative planning
- Planning across iterations
- Planning within iterations
- Iterating on the plan

LUNCH BREAK

Agile UX tactics

- Parallel track workflow
- Incremental implementation
- Design chunking
- Agile usability testing
- Agile user feedback
  - BREAK
- Agile communication

- Timing of agile design
- Incremental implementation
- Incremental design
- Agile communication
Agile UX tactics

Parallel track workflow
- Incremental implementation
- Design chunking
- Agile usability testing
- Agile user feedback
- Agile communication

Timing of agile design
- Incremental implementation
- Incremental design
- Agile communication

Parallel-track workflow
a.k.a. Staggered sprints
Agile design timing: Parallel tracks

Developer track: Focus is on production code
Interaction designers track: Focus is on user contact

Iteration 1: Developer track

Underlying architecture work
Critical features with little user interface design required
Iteration 1: Interaction designers
Design, create prototypes, usability test, and iterate (RITE method)
Field studies to understand user needs (contextual inquiry)

Iteration 2: Developers
Take the verified designs and start making them a reality
Iteration 2: Interaction designers

Usability test completed code for integration and implementation issues

- Iteration 0:
  - Plan and gather customer data

- Iteration 1:
  - Implement high dev cost low UI cost features
  - Design for iteration 2
  - Gather customer data for iteration 3
  - Test iteration 1 code

- Iteration 2:
  - Implement Designs
  - Design for iteration 2
  - Gather customer data for iteration 3
  - Test iteration 1 code
  - Design for iteration 3

Use data gathered in the last iteration to create designs for next iteration
Iteration 2: Interaction designers

Usability test completed code for integration and implementation issues
Use data gathered in the last iteration to create designs for next iteration
Field studies for detailed information needed for upcoming iterations

And so on...

Constant communication between the two tracks is essential for success
These are not just hand-offs
Agile UX tactics

- Parallel track workflow
- Incremental implementation
- Agile usability testing
- Agile user feedback
- Agile communication

Timing of agile design
- Incremental specifications
- Incremental design
- Agile communication

Incremental implementation
- Getting to complete workflows, one DONE at a time
What if your story is too big?

• The work can’t be completed in one iteration
• You need to break it down
• What are the criteria?
• How do you prioritize the feature cards?

Big Design - Waterfall

• One big design document contains everything
• Everyone signs off
• Dev builds it until they run out of time
• QA doesn’t test until Dev has run out of time
• Result:
  – whatever they built first is completed
  – details are left out, quality issues identified too late
  – holes are left in the design
• Much of your design effort is wasted
Big Design - Agile

• Break the story into small pieces, where each piece confers incremental value to the user.
• Determine the minimum first step
• Schedule the pieces in order of importance
• Design incrementally, as if the next piece were the final one
• Change your future plans between iterations if you have learned new things

Benefits of being incremental

• When development runs out of time/resources, the shipped solution
  – Delivers maximum value
  – Has a complete design without holes
  – Has much higher quality
  – Has no wasted design work
Mistakes to avoid

- Designing all the detail up front
- Not *thinking* about the full design up front
- Not breaking things down far enough
- Not delivering a complete (sub) story each iteration – “now the user can...”

Example 1: Doorway

- User story: The user can get in and out of her house easily.
- Completion Criteria:
  - Secure
  - Insulated
  - Lets light in
  - Allows large furniture items to pass
  - Fits with house décor
  - Works even without keys
Example 1: Doorway

• Initial Rough Design:
  - Beautiful Colonial Door
  - Unbreakable translucent window
  - Programmable digital lock
  - Steel deadbolt
  - Metal-clad on the outside
  - High R-Value

Example 1: Doorway

• What is the minimum work that will give the user incremental value towards their goal?
• What needs to be designed for that?

• What is the next smallest item that will give the user an added capability?
• What needs to be designed for that?
Fitting this to your process

• The purpose of incremental implementation is to get feedback early and often.
• After each iteration, gather feedback.
• These questions can affect your breakdown:
  Who evaluates your product?
  Is it always the same people?
  Are your target users internal or external?

Fitting this to your process

You may get feedback from:
• Internal ‘expert users’
• Beta testers under NDA
• The general public (after release or open beta)
• Internal users in a protected ‘sandbox’
• Internal users after general deployment
Fitting this to your process

Before releasing, consider:

• Are you getting the feedback you need?
• Is there enough completed for an external user to evaluate?

Sometimes you may want to hold back certain work until more is done.

Make it easier for the team

• Write *staged specifications* -- a best guess at breaking the design into 1-iteration Story increments
• Then “break” the Stories with developers into Feature cards. Remember: they own the Feature cards. But you need to know how to map those back to Stories & Capabilities.
Exercise: Stove

- As part of building an entirely new kitchen, your team is delivering a natural-gas kitchen range system.

- What is the absolute minimum first deliverable to internal users/testers?
- What is the absolute minimum first deliverable to external beta users?
Stove exercise. Continued

- Gas burners
- Controls for the burners
- Range hood
- Fan for range hood (requires electricity)
- Light for range hood (requires electricity)
- Automatic sparker to light burners (requires electricity)
- Cook timer (can be set to ring after some length of time)
- Oven (box with door and interior burners)
- Controls for oven (sets temperature)
- Convection fan for oven (improves evenness of cooking)
- Self-clean feature (turns oven on very high for 1 hour)
- Safety lock for oven (prevents someone from opening door when it is too hot)
- Oven light (illuminates interior of oven)
- Burner covers (hides the burners for aesthetic reasons)
- Gas leak alarm (sounds if a gas leak is detected)
- Gas line cutoff (allows you to turn off the gas where it enters the range)
- Electrical connection (required for timers, lights, alarms, and sparkers)
- Drawer for holding pots and pans
- Toothpick holder
- Ability to use bio-gas

Agile UX tactics

- Parallel track workflow
- Incremental Implementation
- Design chunking
- Agile usability testing
- Agile user feedback
- Agile communication
- Timing of agile design
- Incremental implementation
- Incremental design
- Agile communication
Design chunking

When it takes more than one iteration to design

from

Jeff Patton’s

Agile Roots 2009 keynote
output

we build this

outcome

we want this

solving problems

NOT

building solutions
Pay equal attention to “discovery” practice and “delivery” practice.

discovery:
problem definition
some problems take >2 weeks to solve

(but they can still be solved in 2-week increments)
Feature design life cycle

Research → Problem definition → Design → Specification → Code → Software → Check implementation

Single-iteration design

Research → Problem definition → Design → Specification → Code → Software → Check implementation

Problem C (for iteration+2) → Solution B (for iteration+1) → Solution A → Cumulative solution

1 iteration
1 iteration, but >1 feature

Single-iteration design Planning Board

D (on n-1 software)
Multi-iteration design

= 

>2 iterations needed for research + design

*just enough so that developers can estimate*
Multi-iteration design Planning Board

Why do multi-iteration design?

- To mitigate requirements risk
- To mitigate estimation risk

Not an excuse to do big design.

You still must do just enough design
Multi-iteration design

3. Look at Planning Board to break Design Chunks.
4. Adjust Design Spike cards on Planning Board.
5. Do design activities (research/prototype/validate)
6. Write staged specifications
7. “Break” into feature cards with developers, and add UX criteria
8. Check implementation

PLUS: Between each iteration, re-calibrate
Multi-iteration design

3. Look at Planning Board to break Design Chunks.
4. Adjust Design Spike cards on Planning Board.
5. Do design activities (research/prototype/validate)
6. Write staged specifications
7. “Break” into feature cards with developers, and add UX criteria
8. Check implementation

PLUS: Between each iteration, re-calibrate

1. Consult the Vision & Mission

Iteration Zero:
Product-level goals
Release-level goals
Capability list (“Ability to”)
Multi-iteration design

3. Look at Planning Board to break Design Chunks.
4. Adjust Design Spike cards on Planning Board.
5. Do design activities (research/prototype/validate)
6. Write staged specifications
7. “Break” into feature cards with developers, and add UX criteria
8. Check implementation

PLUS: Between each iteration, re-calibrate

2. Derive Capability-level design goals

- **Articulate problems** at the “ability to” level
- Defined through *chunked research*
- Carry forward as iteration design goals
- Used to chunk designs
- Used to stage specifications
- Used to define “done”
Capability goals example

• Brush resize:
  – First 5 minutes: Learn without documents
  – Resizing without Brush Editor
  – One control for size, not 2-5 controls
  – Keep focus in-canvas (not in dialog)
  – Keep canvas clean
  – Stylus only

Incremental and continuous research
“Create, maintain, and promote a group of people to donate for a cause through Chariot”

• At the end of iteration 2, many users express interest in this story
• There isn’t enough time in this release to begin to implement this, but the agile team agrees this looks like the Mission for v2.
• What can you research in iterations 3 and 4 to generate a list of capabilities with goals you can bring to Iteration Zero planning for v2?
3. From Planning Board, chunk design

- Break capability into stories
- Look at the list of capability goals. Group them
- Which goals can you investigate with early incremental working versions/prototypes?
- Layer iteration-sized investigations and prototypes for different capabilities

3. From Planning Board, chunk design

- Split the design investigations into small chunks based on what user resources you have available
- Use user proxies (internal & external) to give you feedback on code that doesn’t do full workflows
- Don’t do detailed design for each chunk until you get to that iteration
Design chunking

• Just as you staged the implementation, you can stage your design activities
• There are criteria that will help you place certain design chunks into early-, mid-, and late-stage chunks.

Early-stage design chunks

To investigate/prototype/test:
• solutions that don’t require domain knowledge
• operation-level, not workflow-level of user task
• prototypes that need a lot of facilitator intervention
• solutions you can check independently of (no dependency on other design goals)
• prototypes outside of the main build (no change penalty)
• **Fundamental** designs (other designs will be built on top of these designs)
• Designs that will be re-used in other contexts
Early-stage design chunks

Examples:

• Algorithmic prototypes (SketchBook Pro rotate)
• Specific type of web input field or interaction (type-ahead text entry, date input, credit card input, etc.)
• New types of interaction (iPhone/iPad touch-based colour picker)

Litmus test: Is this a prototype where you would learn something by driving it?

Late-stage design chunks

To investigate/prototype/test:

• solutions that require specific contexts (specific users, environment- or device-dependent)
• solutions that depend on other technology or build on a prior implementation
• workflow-level of user task
• discoverability or learnability goals
• higher fidelity/lower facilitator intervention
• Hub designs (designs that depend on other fundamental designs)
Late-stage design chunks

Examples:
• First-experience or installation investigations
• Multiple agile teams on same product
• Need to support next Mac OS X (or iOS)
• Hub
  (SketchBook Pro Brush Palette = Brush Resize, Custom Brushes, and Brush Defaults Editor)

Litmus test: Is this a prototype where only a specific user or context will validate?

Mid-stage design chunks

To investigate/prototype/test:
• solutions that require partial domain knowledge (user proxies are acceptable)
• solutions that combine previous implementations
• part of a workflow known to end-users
• gather workflow-level scenarios for late-stage design
• Parallel designs
  (designs that can be developed independently)
• other designs that aren’t obviously early- or late-stage
Mid-stage design chunks

Examples:
• Graphic design (during interaction design)
• Emotional response to the look of a UI (e.g., does a look reinforce brand characteristics?)
• Mix and match design chunks per session (SBP: Interactive Brush Outline + 3 other mini-designs with animation students)

Litmus test: Is this not an early- or late-stage design chunk?

Multi-iteration design example

Design goals:
• Design chunks:
  – Brush resize with hotkey
  – Brush resize with stylus

• First 5 minutes: Learn without documents
• Resizing without Brush Editor
• One control for size, not 2-5 controls
• Keep focus in-canvas (not in dialog)
• Keep canvas clean
• Stylus only
Multi-iteration design example

• Design chunks:
  – Brush resize with hotkey
  – Brush resize with stylus (look)

Design goals:
  – First 5 minutes: Learn without documents
  – Resizing without Brush Editor
  – One control for size, not 2-5 controls
  – Keep focus in-canvas (not in dialog)
  – Keep canvas clean
  – Stylus only

• Design chunks:
  – Brush resize with hotkey
  – Brush resize with stylus (interaction)
  – Brush resize with stylus (look)
  – “Workflow” prototype

Design goals:
  – First 5 minutes: Learn without documents
  – Resizing without Brush Editor
  – One control for size, not 2-5 controls
  – Keep focus in-canvas (not in dialog)
  – Keep canvas clean
  – Stylus only
**Exercise: Design chunking**

- NOT Project Chariot
- Inventory system used on factory floor of warehouses.
- Capability: Line manager on factory floor can add an item with a UPC barcode into database.
- Scenarios: Specific to line manager and warehouse setting

---

**Exercise: Design chunking**

- Re-installs at the warehouses only every 16 weeks (roughly 3x a year)
- Current iteration: Iteration 3.
- Next warehouse install: Iteration 12
- 25 line managers near you during 2-day training session: Iteration 6
- On-site visit at a warehouse: Iteration 9
- At your office: QA engineers, support desk staff
**Exercise: Design chunking**

Some user tasks covered by Scan Items:

- Operating the UPC scanner
- Manually entering a UPC number into the database
- On the warehouse factory floor, a line manager must scan an item
- Ensuring the item added to the database was correct
- Installing the UPC scanner, with out-of-box materials (including instructions)

“To make design activities incremental & continuous, craft as many early-stage design chunks as possible.”
4. Adjust Design Spikes on Board

• Try and write stories so they can be roughly estimated (that is, so they are green cards)
• A rule of thumb: Can the work be estimated in units of iterations?
• If you need to do a design spike, you should still break the story into <4 design chunks

Course corrections

– Adjust the agile design plan if needed (change order, drop a chunk)
– Add requirements as needed (remember: one in, one out)
– Move timing of green or blue cards, if needed
– Adjust the staged specifications as needed (swap parts of design for more important capabilities)
– Remove feature cards (this is the “out” part of one in, one out)
Multi-iteration design

3. Look at Planning Board to break Design Chunks.
4. Adjust Design Spike cards on Planning Board.
5. Do design activities (research/prototype/validate)
6. Write staged specifications
7. “Break” into feature cards with developers, and add UX criteria
8. Check implementation

PLUS: Between each iteration, re-calibrate

Agile UX tactics

Parallel track workflow
Incremental implementation
Design chunking
Agile usability testing
Agile user feedback
Agile communication

Timing of agile design
Incremental implementation
Incremental design
Agile communication
Agile usability testing
Rapid formative usability testing

Kinds of Testing

There are different kinds of usability testing:

- Summative
- Comparative
- Formative

Which do you use now?
“To fit into an agile development process, usability testing must also become agile”

How can testing be Agile?

Agile usability testing should:

Enable iterative improvement
Have very fast turn-around
Be time-boxed and regular
Value conversation over big documents
Engage the whole team
Summative testing isn’t agile

- Too slow and cumbersome
- Turnaround time is too long.
- Takes too long to write reports – even short ones
- Big initial problems hide other problems
- Fixing problems changes user response, invalidating previous data
- Collects information that never gets used
- Loses *behavioral insight*

Formative testing fits into agile

- Fast turnaround
- Solves problems before developers start work
- Minimizes problems seen after coding begins
- Minimal reporting
- Can work through “layers” of problems
- *Makes you a better designer*
### What’s the difference between Summative and Formative testing?

#### Different Goals

| Summative Testing | Find and measure all usability problems.  
|                  | “How usable is this product?” |
|-------------------|--------------------------------
| Formative Testing | Maximize iterations to hone in on the best achievable design.  
|                   | “How usable can we make this product?” |
### What’s the difference?

**Different scheduling**

<table>
<thead>
<tr>
<th>Testing Type</th>
<th>Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summative Testing</td>
<td>During late development or post-development</td>
</tr>
<tr>
<td>Formative Testing</td>
<td>During any/every agile iteration</td>
</tr>
</tbody>
</table>

### What’s the difference?

**Different starting point**

<table>
<thead>
<tr>
<th>Testing Type</th>
<th>Starting Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summative Testing</td>
<td>Start with finished product (or feature)</td>
</tr>
<tr>
<td>Formative Testing</td>
<td>Start with fastest/cheapest prototype that still captures basic behaviours</td>
</tr>
</tbody>
</table>
### What’s the difference?

#### Different deliverable

<table>
<thead>
<tr>
<th>Summative Testing</th>
<th>Report, containing found problems, their severity, possible statistical analyses (task time, error recovery), and recommendations for improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formative Testing</td>
<td>Interaction design verified to be complete and correct</td>
</tr>
</tbody>
</table>

### What’s the difference?

#### Different methodologies

<table>
<thead>
<tr>
<th>Summative Testing</th>
<th>The prototype and protocol remain unchanged between testers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formative Testing</td>
<td>The prototype may be updated, but the design goals remain unchanged between testers</td>
</tr>
</tbody>
</table>
When do you change the prototype?

The interaction prototype should only be changed under certain circumstances:

- The user encounters a problem that blocks him or her from completing the task.
- The prototype fails its design goals.
- You’ve seen a problem enough times to warrant a change.

What’s the difference?

**Overall:**

<table>
<thead>
<tr>
<th>Summative Testing</th>
<th>Finds problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formative Testing</td>
<td>Finds solutions</td>
</tr>
</tbody>
</table>
Example Test Run

Say you decide that you need 5 testers to verify the usability goals. Your test may go like this:

1. Test run (succeeds) – 4 more testers required
2. Test run (fails) – changes made – 5 more testers required
3. Test run (fails) – changes made – 5 more testers required
4. Test run (succeeds) – 4 more testers required
5. Test run (succeeds) – 3 more testers required
6. Test run (fails) – changes made – 5 more testers required
7. Test run (succeeds) – 4 more testers required
8. Test run (succeeds) – 3 more testers required
9. Test run (succeeds) – 2 more testers required
10. Test run (succeeds) – 1 more testers required
11. Test run (succeeds) – DONE and VERIFIED

Example Test Run

Total number of test runs: 11
Versions of prototype examined: 4

To examine four prototypes with “regular” testing methodologies to the same level of verification, you would need to conduct four tests, each with five test runs – about double the number of test runs.

Plus, you’d have four reports to write.
How many successful runs is enough?

It depends on the confidence level you want.

http://www.measuringusability.com/wald.htm

Reporting results

If you choose to write a report, do so for historic reasons, e.g.:

- In case someone asks you later “why didn’t you do X?”
- If you re-visit a feature in a later release, and need the context of earlier decisions
- If you change usability staff during a project.
Engaging the team

• Developers working on a story should see at least one usability test for that story.
• Interaction designers should be observing all tests of their designs.
• Every team member should be invited to observe.
• Key test results should be “re-enacted” for team members.

What goes in the report?

1. The significant problems found (don’t list every little thing, no one cares)
2. The changes that were made in each version of the prototype, and why
3. Did the changes fix the problems?
Waterfall Test Scheduling

1. Wait until you have a significant block of features to test.

2. Schedule a test, and recruit testers.

3. Conduct your test.

4. Report your results.

Agile Test Scheduling

1. Schedule FIRST – trust there will be something to test when the time comes.

2. Recruit continually, to build a pool of reusable testers.

3. A few days before each test, see what prototypes and features are ready, and prepare a test protocol based on that.
Agile Testing Frequency

One test per iteration would be ideal – but it depends on your resources and iteration length.

Every 2\textsuperscript{nd} or 3\textsuperscript{rd} iteration might be more realistic.

Don’t skimp! This is a powerful way to improve your designs.

Spacing out the Test Runs

Don’t schedule your testers back-to-back!

Leave enough time between test runs to update your prototypes, if needed.

Tip: Write out your test report as you go along.
What to test in each iteration

*Validate* work done in iteration the previous iteration (test the completed code)

*Improve the design* of features to be added in the next iteration (test your prototypes)

*Identify* usability bugs and features that need design refactoring
What if I run out of testers and my design still fails?

Options:
1. Recruit more testers (from your regular pool) and extend the test
2. At the next iteration planning meeting, push this feature later in the schedule

Either solution is better than building something you know will fail.
Success Stories

Some other resources


Project CHARIOT: Choosing what to usability test

- You have just finished Iteration 2, and are beginning Iteration 3.
- From the Planning Board shown in the handouts, describe (in point-form) 3 or 4 different prototypes you could test during this iteration.

Agile UX tactics

- LUNCH BREAK
- Parallel track workflow
- Incremental implementation
- Agile usability testing
- Agile user feedback
- Timing of agile design
- Incremental implementation
- Incremental design
- Agile communication
BREAK

Agile user feedback
Mitigate data bias
Why is user feedback for agile hard?

- Agile community relies on self-reported data
- Recurring “reset” every iteration
- Some working versions don’t do enough
- Fewer investigation sessions
- Continuous: hard to find enough users

Why is agile user feedback better?

- Fixes in the current release (next iteration!)
- Not clustered either at beginning or end
- Focused: Less wasted effort
- Progressive: Use fewer users who will learn
- Longitudinal: Iterate on requirements
Hunter/gatherer vs. Farmer

Tactics

– User contact plan
– User proxies
– Combine user investigations
– Recurring user studies
– Bring context in-house
– Longitudinal studies
User contact plan

- Set up a pool of potential end-users
- Consult the planning board for requirements
- Consider all stages and types of contact
- Keep adding to the group

Examples: User contact plan

- SketchBook Pro
  - in-house: internal users, user proxies
  - on-site, longitudinal: Design Partners
- Showcase
  - in-house, ad hoc: internal users
  - on-site, recurring studies: monthly visits to Detroit
- Mudbox
  - in-house, ad hoc: internal users, user proxies
  - remote recurring studies: Charter Customers
  - occasional on-site studies: Charter Customers
User proxies

- Users with some, but not all, characteristics
- Use to validate early or partial prototypes
- Balance with true end-users
- Recruit with as much care as end-users
- See http://tinyurl.com/userproxy

Combine user investigations

Do contextual inquiry & usability testing in same session
- The most expensive cost of user investigations is scheduling
- Get more data in less time
- Test and investigate much smaller chunks (15 minutes of work)

“Formative usability investigations for open-ended tasks”
Desirée Sy. UPA 2006
Recurring user studies

Set up regular, recurring usability investigations
– Contextual inquiry + usability test whatever you have available at the time
– Can apply to in-house, on-site, or remote

Timeboxed design encourages:
– continuous end-user input
– lower-fidelity prototypes
– faster movement through design solution space

Example: Recurring user studies

– In-house. End-users brought in every 2 weeks
– On-site. Drove to Detroit every month.
– Remote usability testing. Every 3 iterations.
Agile is a team sport

- Design and user research must be closely coupled
- Educate the agile team on different kinds of feedback (site visit, usability test, preview, etc.)
- Give artifact data directly to team
- Use agile communication channels

Agile UX tactics

LUNCH BREAK
Parallel track workflow
Incremental implementation
Agile usability testing
Agile user feedback
BREAK
Agile communication

Timing of agile design
Incremental implementation
Incremental design

Agile communication
Agile communication

Show, don’t write. Motivate, don’t report.
Collaborate, don’t blame.
Every day.

Why is agile communication hard?

We value... working software over comprehensive documentation
--The Agile Manifesto

Traditional UX deliverables:
- are documents
- trail decision capture, instead of leading
- are a substitute for working product
- often don’t motivate the team
Why is agile communication hard?

- Communication is hard
- Fewer documents forces this issue
- Team members have to learn new skills

How is agile communication better?

- Detailed documents never worked anyway
- Genuine collaboration
- Faster to in-product solutions
What to communicate?

- UI designs to implement
- UX success criteria
  - user/task data from end-users
  - usability test results
  - check implementation
- changes and fixes for working versions
- timing of upcoming designs

When to communicate

As soon as possible

incremental & continuous
to facilitate
iterative & collaborative
Continuous & collaborative

• Talk* to developers and QA every day
• Use the team’s language
• Use agile tools (scrum, cards, acceptance criteria)
• Your job is make the developer want to solve the user’s problem
• Check what’s being coded every day (yes, it’ll be strange at first)

* or use asynchronous communication...

Agile communication modes

• Scrum
• Post-scrum problem demos
• Story meetings
• Daily walkaround
• Iteration planning
• Charter planning
• Staged specifications
• Reporting (mostly for Design)
3 example days
- From the shared Planning Board, we'll discuss what types of activity and collaboration occur for 3 different days in the Chariot life cycle
- Story meeting
- After a usability test
- Scrum after returning from an onsite customer visit

Story meetings
- Organize design meetings by story or capability. Closer stories get priority.
- Invite core team for that story.
- For stories where no coding work has started, this is a design discussion meeting.
- For stories where coding is in-progress, you are validating both the design and how that design is being implemented.
- As much as possible, re-enact user behaviour with demos. Record.
- Update bugs in the meeting, as you’re going along.
Story meetings

Validating current implementation (and work for next 2 iterations)
• Complex bugs
• New or revised acceptance criteria?
• Are there tasks that can be dropped? Remove them from bug tracking, and explain why in the item.
• New tasks for next iteration? Describe and enter them.

Get ready for next iteration planning meeting:
• Bring new tasks to decide if they go in project backlog, and understand their priority
• Where you’ll suggest that tasks move iterations or be dropped, give core team the background

Design discussions and decisions:
• Describe design and usability problems. Collaborate on solutions.
• Update the design history for the Story.
• Explain what the design follow up is (re-design and usability testing plans)

After any design investigation

Same day, after session, team members at visit agree:
– Top 3 issues relevant to current iteration. What should the team know ASAP? (e.g., problems, bugs, refined acceptance criteria)
– Top 3 issues relevant to a future iteration. (e.g., acceptance criteria, can any story be dropped or swapped, story priority)
– New capabilities & stories (not necessarily for this release)
– If there are many sessions, then collate issues relevant to future iterations
Scrum

Next scrum after session:
• Report the Top 3 issues relevant to current iteration.
• If you can’t phone in, send notes

Returning from customer visit, at next scrum after visit where you are in-person or can demo remotely:
• From collated visits, recap the Top 5 issues relevant to a future iteration.
• Describe new epics/stories, but don’t prioritize – just give the team info.
• Invite team members to the trip report meeting immediately after the scrum.
• Set up follow-up meetings with team members.

Post-scrum trip demo

Next scrum after visit where you can demo (in-person or remotely):
• “Re-enact” user behaviour to demo critical issues. Where possible, use the user’s data.
• Record this meeting (if you’re demoing software, screen capture; if it’s a whiteboard demo, video the description).
• Distribute data + other artifacts.
• Set up follow-up meetings with team members.
Distributed agile teams

- Key team members should be co-located during Iteration Zero if humanly possible
- Key team members should have synchronous voice/virtual modes during iteration planning
- Otherwise, use asynchronous chat systems shared by all team members
- Democratize the communication mode
- Have photos of all team members

Questions?
- Other communication issues that you have?
Agile UX planning

Agile UX tactics

Wrap up
Summary and Course retrospective
Planning framework

agile qualities
- iterative
- incremental
- continuous
- collaborative
Tutorial leaders

John Schrag
john.schrag@autodesk.com
@JVSchrag

Desirée Sy
desiree.sy@autodesk.com
@DesireeSy
Appendix: Transitioning from Waterfall to Agile

When development teams first switch over to agile methodologies, they often run into the same kinds of problems. Many of these problems are covered in the body of these course materials; this appendix looks into some of the other problems you might run into, and some best practices for avoiding them.

Main Transition Problems

- No Iteration Zero
- Planning is not iterative
- Design/implementation is not incremental
- Iterations are not *really* finished
- Using the same methods the same way as you did before
- The full team isn’t available full-time
- Part of the team is agile, part isn’t
- Teams need help with agile processes
- Political or management problems

Looking at each of these problems in detail:

No Iteration Zero

This is the same as “not planning”. It is a common problem because some certain agile practices, such as XP and Scrum, do not allow an Iteration Zero. However, in our contact with agile and UX practitioners in dozens of companies, we have found that *having an iteration zero is a strong predictor of a designer’s satisfaction with their agile process, and a team’s satisfaction with their delivered product.* So we recommend it.

Without Iteration Zero, your whole team will not have shared and explicit understanding of the business goals. This can lead to a lot of meeting churn (since priorities of what to do and what to cut will be unclear), and even to ‘schizophrenic’ releases -- where the new functionality (while well-implemented, and well-executed) is all over the place, and helps no one significantly.

For details on how to have a successful Iteration Zero, see the section on *Iteration Zero* in the course notes.

Planning is not iterative

For agile to work properly, re-planning at each iteration is as critical as the initial planning. You must be ruthless about cutting and moving stories to make things fit on your planning board based on things you learned during the iteration.

Often, managers coming from waterfall backgrounds are reluctant to take stories off the board – they imagine that the team will make time somehow, later on. Or the managers put too many stories into an iteration, because they can’t quite believe the tracking numbers they were getting from previous iterations. They “just know” the team can do more this time.

Developers coming from a waterfall background often don’t estimate their work properly for agile. They are accustomed to estimating just the amount of time to get the first revision of the code checked in – bug fixing is expected to come later. For agile, developers have to start including in their estimates time to fix bugs and to meet acceptance.

For details how to re-plan successfully and realistically, see the section on *Planning for Change* in the course notes.

Design/implementation is not incremental

Interaction designers from a waterfall background are accustomed to designing the final vision all at once. Often, when they start working agile, they try to do this initially. However, they are forced to update their big design after each iteration, as stories are cut or re-prioritized. This creates a huge amount of wasted and repetitive work, and it can leave development hanging, waiting for design to catch up. Design needs to be delivered incrementally, with detail only being added as features come into view.

For details on how to break down design and implementation efficiently, see the sections on *Incremental Implementation* and *Incremental Design* in the course notes.

Iterations are not *really* finished

This is one of the most common transition problems. Developers are not used to working collaboratively, and like to hold on to their work-in-progress until the very end of the iteration, checking it in at the last minute. As a result, there is no time for QA to check for
bugs; no time for the acceptance criteria to be checked, and certainly no time for the developers to fix whatever problems are found. When the next iteration starts, developers start on new features, and bugs from the previous iteration are pushed to later. This way leads to eventual disaster – and you don’t get deployable code at the end of your iteration.

This is an issue that needs to be solved collaboratively between development, QA, and design. Code needs to be checked in as early as possible so that QA and acceptance can begin. Developers need time to fix bugs and meet acceptance, and this all needs to happen before the end of the iteration. If you are regularly not hitting the end of your iterations, then they have too many stories/tasks in them.

**Using same UX methods the same way as you did before**
Traditional UX methods simply do not fit will into agile processes. They need to be adapted to become *iterative, continuous, incremental,* and *collaborative.*

**Iterative** means that the methods have to expect and embrace change along the way, without ever losing sight of the goals.

**Continuous** means that the methods must be able to run regularly or continually in a sustainable way indefinitely – no last-minute heroic efforts required or wanted.

**Incremental** means that the methods deliver small but complete improvements to value, instead of half-finished features or designs.

**Collaborative** means that the whole team shares responsibility for the whole product, although the work may be divided by expertise. Your practices should encourage sharing of knowledge. It’s not about tossing a report over the wall, and thinking “okay, I’m done my part.”

For details on how to alter your UX practice for agile, see the second half of the course notes. (*Incremental Implementation, Incremental Design, Agile Usability Testing, Agile Customer Feedback,* and *Agile Communication.*)

**The full team isn’t available full-time**

In waterfall projects, the work tends to be staggered. Designers finish their work, which is then passed to Development, which is then passed to QA and Documentation. With agile, the whole team needs to be working on the same things in the same time frame.

When a development team first switches to agile, often they are ready to go, but the QA team is still looking for bugs in an earlier release, and the documentation people are documenting it. So QA and documentation are not able to collaborate with development on the current iteration. This leads to iterations not being properly tested or documented, and later bug build-up, which prevents the iteration releases from being deployable. The following cycles then become a catch-up game.

Another problem is split staff: some companies think that it is reasonable for one designers to be assigned to two three or more projects. This rarely works well, because there is a certain fixed overhead to being on each product, and that is the same whether you are part-time or full-time.

This is a problem that must be dealt with creatively at the management level.

**7. Part of the team is agile, part isn’t**

In many companies, the development team is trained in agile practice, but no one else is. Documentation, Design, QA, legal, and marketing all expect things to continue to work in the old way, and think that the developers are running out of control. Demands are put on development that cannot be met under agile – for example, marketing may ask for a firm commitment to producing a full slate of features six months or a year ahead of time.

The best way to deal with this is to ensure that everyone is trained in agile practice – not just the developers. For newly-agile teams, new deliverables and timing need to be agreed upon with other departments (ideally in iteration zero) to prevent people from being blindsided or disappointed later on.

**8. Teams need help with agile processes**

The first time a team tries to work agile, it probably won’t go all that well. Like anything, working agile takes practice, and the processes need to be modified to fit into your workplace. We strongly recommend hiring or borrowing someone with agile experience (and a positive attitude) to serve as a coach during the team’s initial efforts (and especially the daily meeting), and to help with the retrospectives.

It is also wise to choose the right kind of project for your first attempt – the right size, with a good team including senior people.
Finally, it is important to remember that a key component of successful agile is lots of face-to-face conversations. If you are doing away with the huge documents that waterfall required, then you need to replace them with something else, and conversation is what agile requires.

9. Political or management problems
These are more problems that need to be dealt with on the executive level:

- No executive support.
- No grass-roots support: agile processes imposed on development team from outside
- Team distracted with creating demos for upper management instead of working on deliverable

Getting buy-in at all levels, and setting expectation is key to success.

Good ideas
Aside from the material covered in the course, here are some other good ideas to keep in mind when transitioning to an agile practice:

- Train everyone on the product team in agile – not just developers.
- Have an agile coach
- Use meeting tools that democratize information across distributed teams – if your team are not co-located, make sure that the people in one location don’t have superior access to information.
- Do regular retrospectives and implement changes to make a process that works for you (CRITICAL!)
- Make your design process agile
- Pick a good pilot project