21. Iteration [1]

10 November 2008

Bob Glushko

Plan for ISSD Lecture #21

The Design of Iteration

Reminder: Iterative Methodologies

Iteration and Design Contexts

Iteration in Organizational and Architectural Contexts

"Continuous Stakeholder Alignment"

Iteration

Every design effort inevitably has some work that need to be done more than once

An iteration is the cycle by which a work product, artifact, or design is revised after evaluating feedback from a stakeholder

Iteration is most often described as a characteristic of a design methodology, but we can apply it at broader or narrower scopes

Designing Iteration

What is the project scope to which iterative methods will be applied?

What are the processes in each iteration?

What is the "cycle time" of the iteration?

The "metamodel" for feedback

The Scope of Iteration

Project level

- Iteration as the foundation of a design and project management philosophy
- This is the scope at which Chapter 5 in OiSD is written to describe "continuous alignment with stakeholders"

Methodology level

- Iteration in the design activities
- This is the level when a project is following "document engineering" or a "spiral methodology" in which both the methodology as a whole and its constituent activities are iterative

Activity level

- Iteration within each design activity
- "Agile programming" - the essence of the activity is rapid iteration

Iteration Cycle and Processes

By definition iteration involves building something, getting feedback on it, evaluating the feedback, and then revising what was built

The relative time dedicated to each of these activities is an important project management and planning decision that reflects the design and organizational contexts

So we could discuss "iteration patterns" or "best practices" and "worst practices" (see Fairley and Wilshire in "Iterative rework: the good, bad, and the ugly")

The "Metamodel" for Feedback

An iterative cycle by definition involves getting feedback, which we use to determine which direction to move or which actions to take given the current state or design

How to get feedback

Who to get feedback from

What to get feedback about

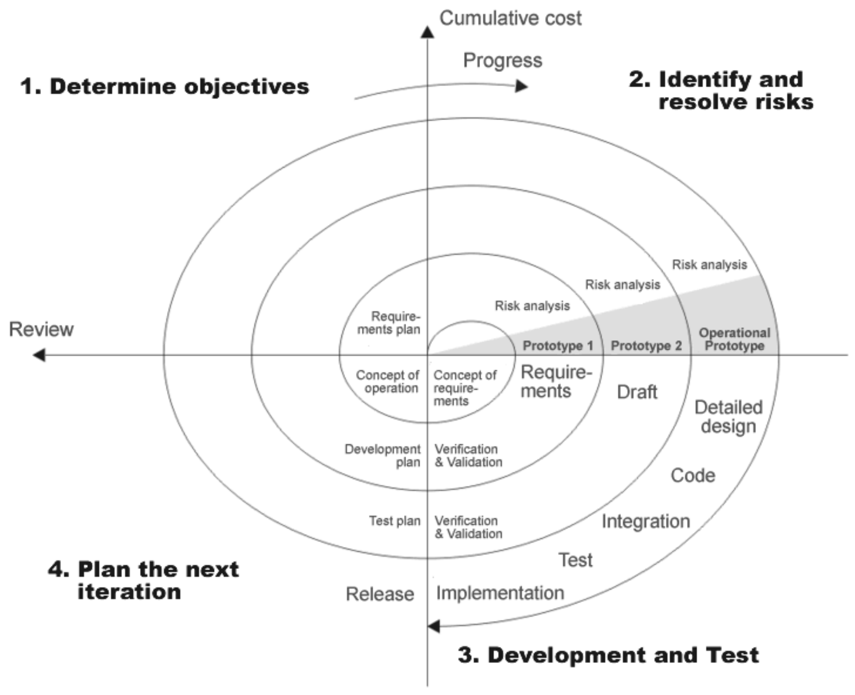
What to do with the feedback you get

Reminder: Iterative Methodologies

In contrast with "sequential" methodologies, iterative ones assume that reworking or revisiting previous analysis and design activities is inevitable and desirable

The goal of iterative methods is "get enough right at each step to know which step to take next"

Spiral Methodology (Boehm, 1988)



"Agile" Methodologies

"Agile" or "extreme programming" methods for software development have become very popular methods in the last decade and are a specialized form of iterative methods used by small design teams

These methods deprecate up front investment in scoping and requirements specification, and rely on very rapid coding and testing cycles to incrementally develop software

Lessons from "Enterprise Transforming Projects that Don't Kill the Enterprise"

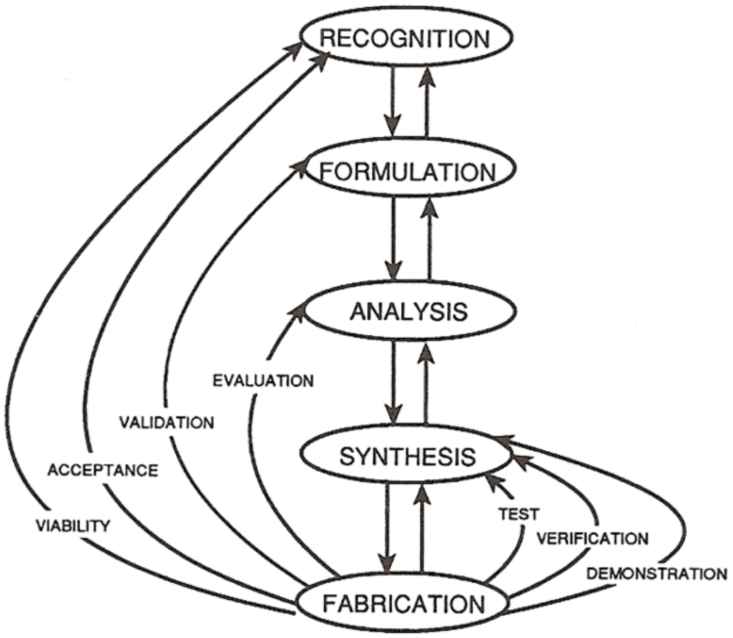
Deliver Frequently

- Actual benefits from deployed software NOW always beat hoped-for future ones
- Project sponsors need justification for continued investment
- Early deliveries can focus on exploring new technology capabilities to be exploited in later ones

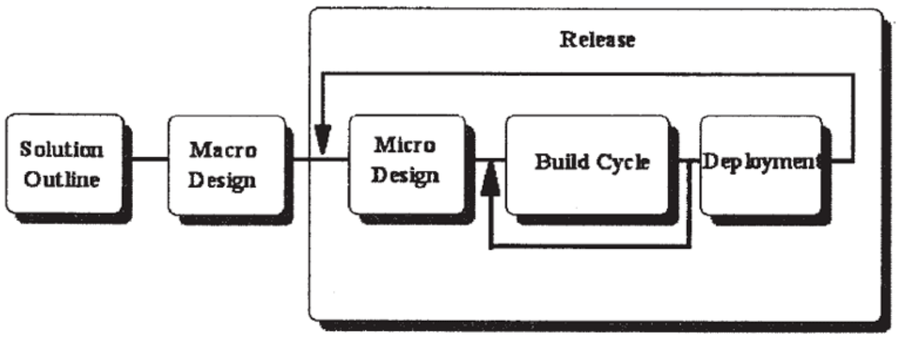
Expect Surprises

- Changes to requirements should be seen as inevitable and desirable, reflecting better understanding, not as negative impacts on a plan
- Planning should be a continuous process

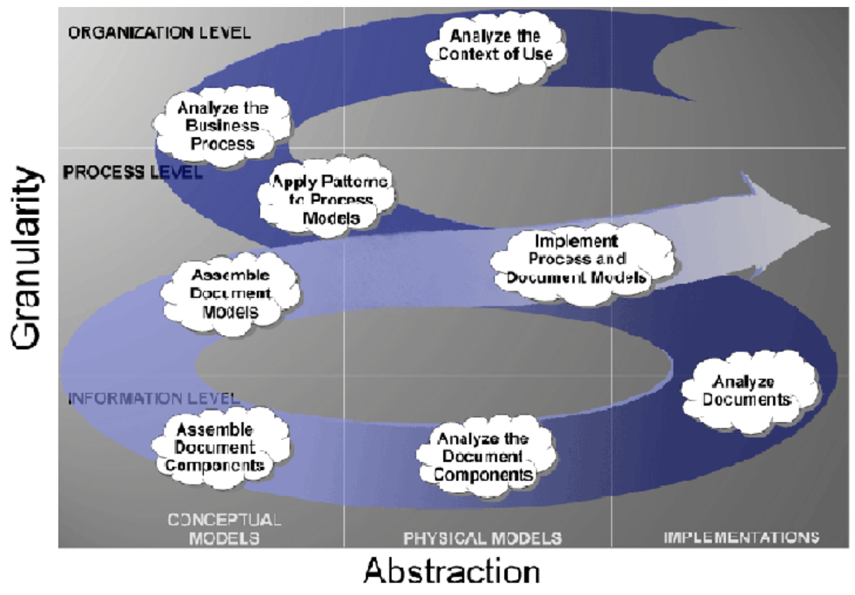
"Design for Success" Methodology



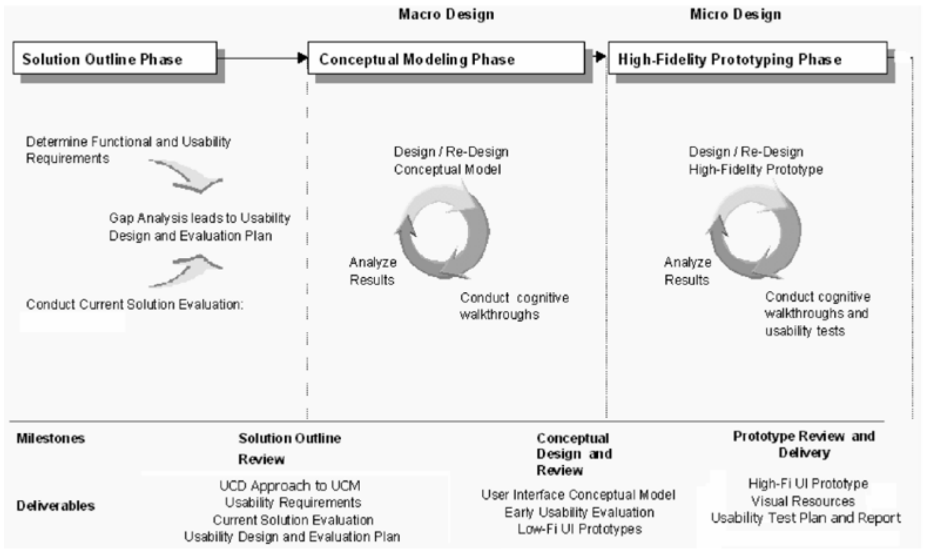
"IBM Global Services" Methodology



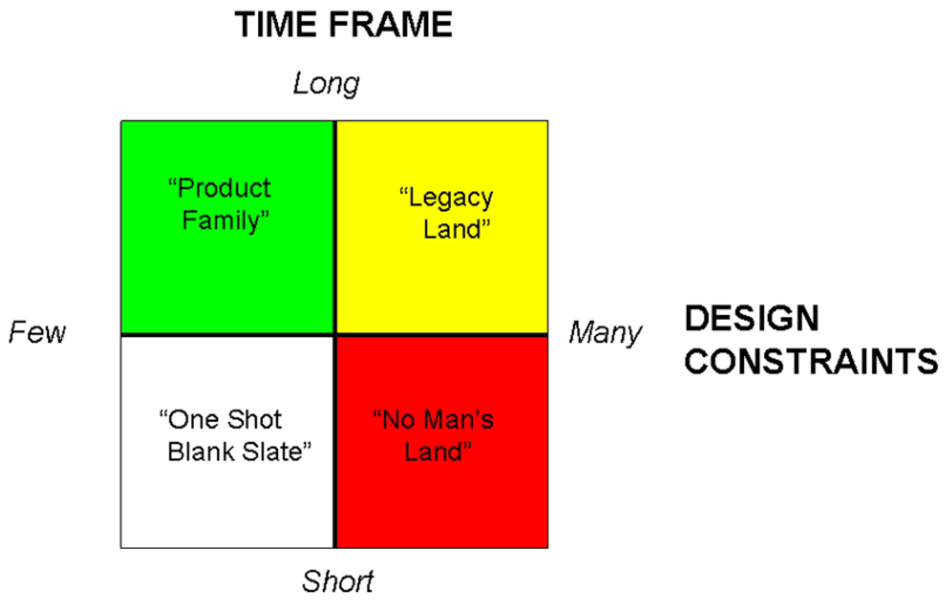
"Document Engineering" Methodology


ORGANIZATION LEVEL
Analyze the Context of Use
Analyze the Business Process
PROCESS LEVEL
Apply Patterns to Process Models
Implement Process and Document Models
Assemble Document Models
INFORMATION LEVEL
Analyze Documents
Assemble Document Components
Analyze the Document Components
CONCEPTUAL MODELS
PHYSICAL MODELS
IMPLEMENTATIONS
Granularity
Abstraction

IBM "Rapid Prototyping Engagement"


Macro Design
Micro Design
Solution Outline Phase
Conceptual Modeling Phase
High-Fidelity Prototyping Phase
Determine Functional and Usability Requirements
Gap Analysis leads to Usability Design and Evaluation Plan
Conduct Current Solution Evaluation:
Design / Re-Design Conceptual Model
Analyze Results
Conduct cognitive walkthroughs
Design / Re-Design High-Fidelity Prototype
Analyze Results
Conduct cognitive walkthroughs and usability tests
Milestones
Solution Outline Review
Conceptual Design and Review
Prototype Review and Delivery
Deliverables
UCD Approach to UCM
Usability Requirements
Current Solution Evaluation
Usability Design and Evaluation Plan
User Interface Conceptual Model
Early Usability Evaluation
Low-Fi UI Prototypes
High-Fi UI Prototype
Visual Resources
Usability Test Plan and Report

Reminder: Design Contexts



Iteration and Design Contexts

We can imagine "eureka" design moments when the "right" solution to a problem seems to emerge all at once, but this rarely happens in reality, and most design is iterative -- but the cause isn't the same across design contexts

In legacy contexts the starting point is well defined and design constraints are severe, so the more important question is "can or should we try to build this" rather than "what should we build" -- and the only feasible design approach is an incremental and iterative one

In less constrained contexts, there might not be an obvious starting point for the design, or there may be many equally attractive starting places, so an incremental design and development process might be needed to determine what should be built

"IT Ecosystems: Evolved Complexity and Unintelligent Design"

IT hardware and software environments have grown steadily in complexity...

- and "consist of many specialized functional components, often designed by multiple vendors"
- "interconnected in a plethora of permutations"
- "through an accumulation of small changes to prior iterations"
- or because of acquisitions, each "brought in with their own existing processes and departmental divisions"

Iteration in Organizational and Architectural Contexts [1]

Products and services for highly regulated contexts usually follow "waterfall" methodologies to ensure accountability and traceability

Similarly, when a custom solution is developed under contract with an external customer, "getting customer requirements" is an explicit initial activity (which usually involves iteration)

But once the contract is in place, changes to contracted requirements can involve formal reviews and renegotiation, so they are usually managed with a "coarse-grained iteration" so that they don't disrupt the project

Iteration in Organizational and Architectural Contexts [2]

In contrast, for systems or applications or services intended for "mass market" or "off the shelf" contexts, the requirements process is more informal and iterative because there isn't a contracted customer

Market feedback often results in a steady stream of modified/enhanced offerings

And in hosted or "software-as-a-service" architectures, the ease with which changes can be tested or backed-out strongly encourages iterative development and deployment

Iteration of Scope, Priority, and Stakeholder Identification

Identifying stakeholders and determining their priority depends on the scope of the design project

However, the scope might enlarge during the project, which can make previously excluded stakeholders relevant or raise the priority of those previously "on the fringe"

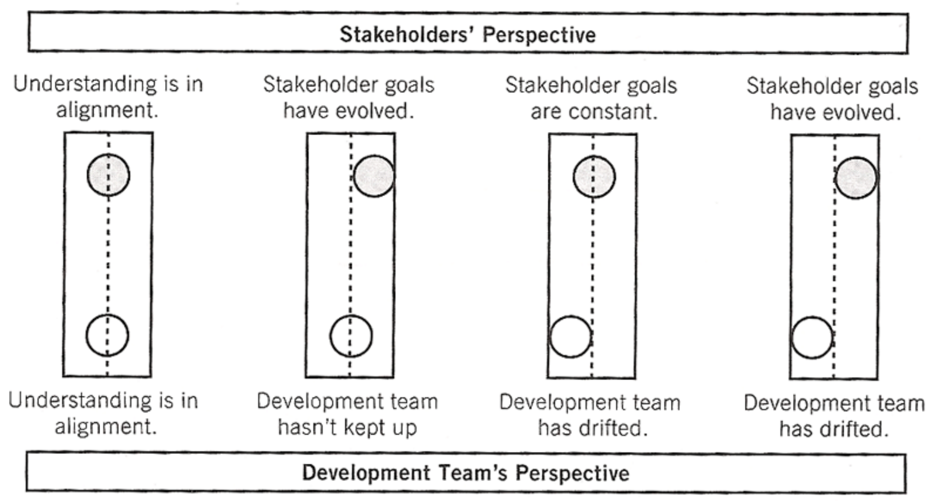
Even if the scope doesn't change, as the design evolves, it might give stakeholders more clarity about their goals or change their priority

The Need for "Continuous Stakeholder Alignment"

Even if stakeholder goals and requirements are initially understood, in most design contexts the "world doesn’t stand still" and requirements and constraints change

The perspective of the design and development change can also change because of "drift" (lack of focus) and because of new ideas about meeting requirements

Alignment and Misalignment


Stakeholders' Perspective
Understanding is in alignment.
Stakeholder goals have evolved.
Stakeholder goals are constant.
Stakeholder goals have evolved.
Understanding is in alignment.
Development team hasn't kept up
Development team has drifted.
Development team has drifted.
Development Team's Perspective

The Alignment Process

Initial alignment

Plan to stay aligned

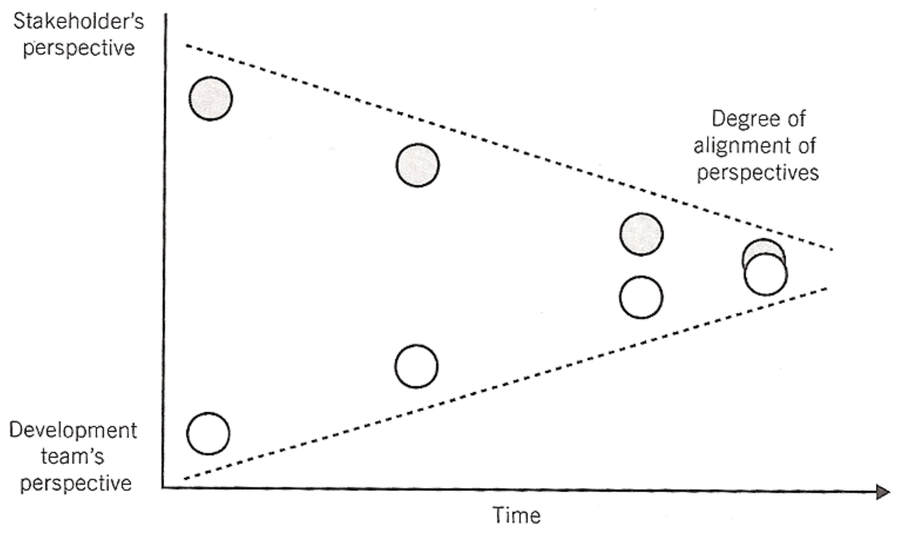
Collect feedback

Evaluate feedback

Respond to feedback

Communicate to stakeholders about your response to feedback

Increasing the Alignment


Stakeholder's perspective
Degree of alignment of perspectives
Development team's perspective
Time

Planning to Stay Aligned

- Design appropriate "dialog" methods to get effective feedback from all the stakeholders
- Plan to get feedback on the "complete" offering
- Build capacity to respond to feedback that is consistent with the kinds and amount of feedback you get
- Build time to respond to feedback into project plan

"Dialog" Methods for Obtaining Feedback from Stakeholders

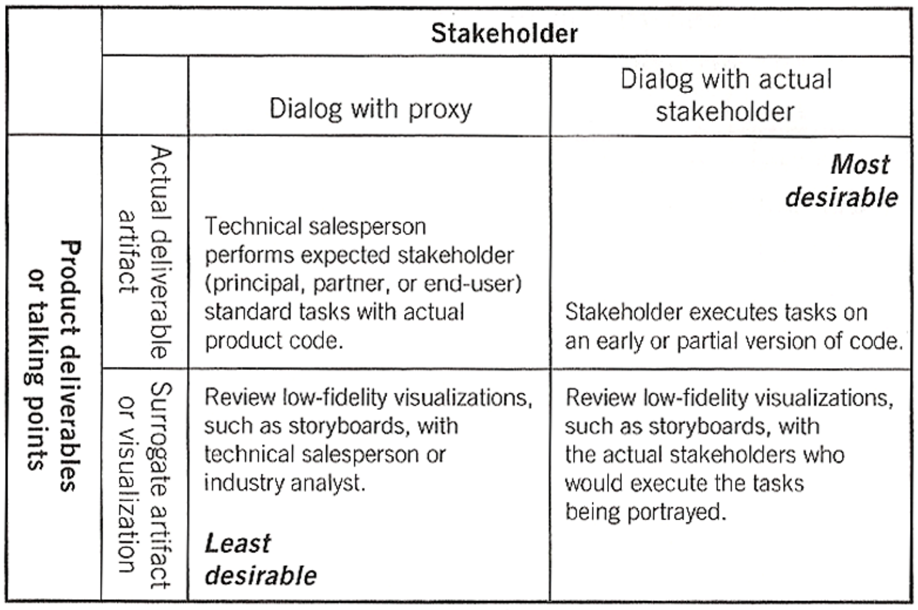
Tailor the methods to each type of stakeholder

If you ask them or show them something, make sure you've engaged them appropriately

If you observe them, make sure you're observing them doing things that matter

If you can't interact with them directly and get indirect feedback, be careful and skeptical about it

Direct and Indirect Feedback



		Stakeholder	
		Dialog with proxy	Dialog with actual stakeholder
Product deliverables or talking points	Actual deliverable artifact	Technical salesperson performs expected stakeholder (principal, partner, or end-user) standard tasks with actual product code.	***Most desirable*** Stakeholder executes tasks on an early or partial version of code.
	Surrogate artifact or visualization	Review low-fidelity visualizations, such as storyboards, with technical salesperson or industry analyst. ***Least desirable***	Review low-fidelity visualizations, such as storyboards, with the actual stakeholders who would execute the tasks being portrayed.

Proxy Stakeholders and Surrogate Artifacts

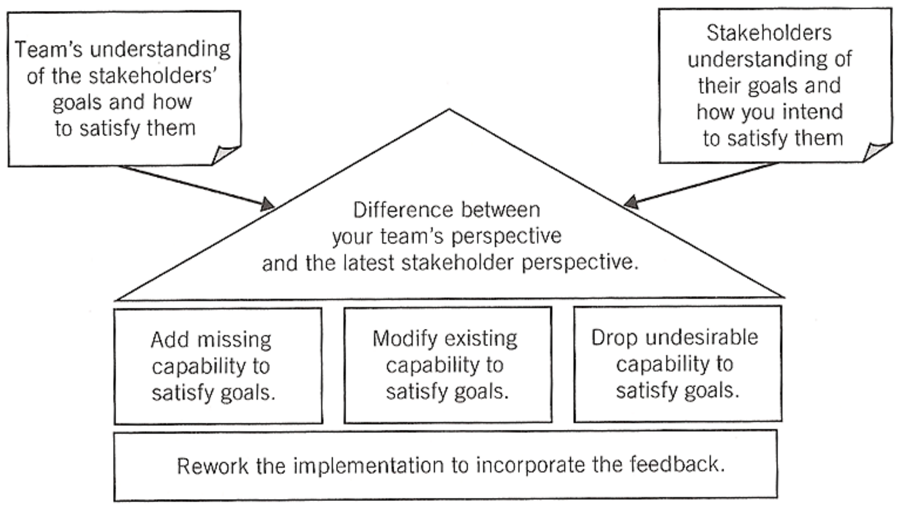
You get the most reliable feedback when you show ACTUAL stakeholders ACTUAL work products

But this isn't always possible

You get the least reliable feedback when you show a SURROGATE artifact to a PROXY stakeholder

And some PROXIES and SURROGATES are better than others

Responding to Feedback


Team's understanding of the stakeholders' goals and how to satisfy them
Stakeholders understanding of their goals and how you intend to satisfy them
Difference between your team's perspective and the latest stakeholder perspective.
Add missing capability to satisfy goals.
Modify existing capability to satisfy goals.
Drop undesirable capability to satisfy goals.
Rework the implementation to incorporate the feedback.

Responding to Feedback

"... measuring changes to design decisions as defects leads to to an unsuccessful product and unhappy developers... [who] need to be empowered to continuously improve understanding of the stakeholders"

"It does not make sense to tell your stakeholders 'We are interested in all your feedback...but we can react only to suggestions that can be implemented in a week'"

If the iterative cycle time is short, you might have to "leapfrog" feedback into future iterations

Inform your stakeholders about your response, so you can get feedback on it...

Readings for 12 November

Richard Fairley & Mary Willshire, "Iterative rework: The good, the bad, and the ugly" Computer, 2005.

Katrina Rhoads Lindholm, "The User Experience of Software-as-a-Service Applications"

Steven D. Eppinger, Daniel E. Whitney, Robert P. Smith & David A. Gebala, "A Model-Based Method for Organizing Tasks in Product Development"