8. Use Cases

24 September 2008

Bob Glushko

Plan for ISSD Lecture #8

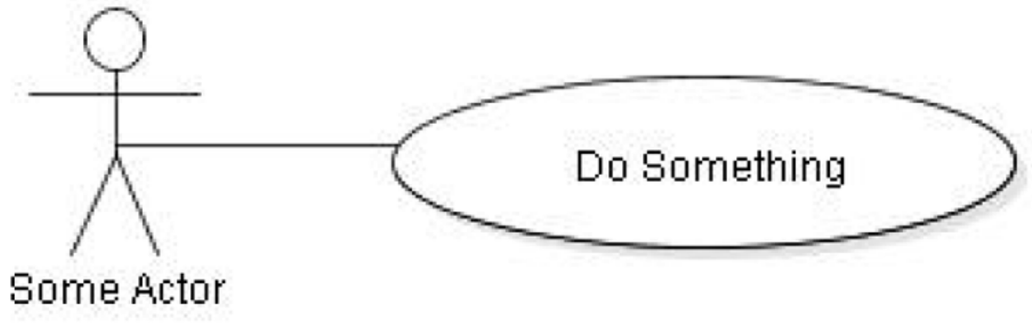
Motivating Use Cases

Parts of Use Cases

"Essential" Use Cases

Setting Priorities

This is NOT a Use Case



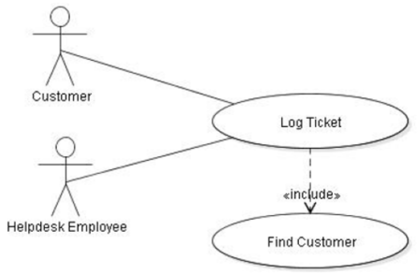
It is a USE CASE DIAGRAM; it serves as an index to a use case

This IS a Use Case

Log Ticket

Main Success Scenario:

1. The Helpdesk Employee Finds Customer using its name or address
2. The Helpdesk Employee enters the details of the Ticket
3. …



What IS a Use Case?

An outside-in or "black box" depiction of a system's functionality from the perspective of an "actor"

Use cases define possible sequences of interactions or procedures between actors and the system that achieve some goals or otherwise result in successful outcomes for users (and often, other stakeholders)

The full set of use cases for a system defines its scope and boundaries wrt other actors

Use cases can be grouped to facilitate communication with different stakeholders, to indicate priority of functional requirements, or to enable incremental or parallel implementation and testing

Why We Need Use Cases

We need to specify the actions and interactions between actors and systems in a way that:

- ... is from the user's or business point of view, not the system's
- ... can be readily understood by all stakeholders so they can be used to elicit and verify requirements
- ... doesn't assume or imply anything about how functions are implemented
- ... is complete and unambiguous

Why Not Just a Text Specification?

Text is a suitable format for specifying a use case

- ... but any text specification in "natural language" can easily be incomplete, inconsistent, or ambiguous
- ... which is why "best practices" for writing text specifications include "simplified writing" and "business rules" (with controlled grammar, vocabulary, and logic flow)

Use case diagrams serve as minimal and abstract "pointers" to the text part of the use case specification

Parts of a Use Case

Scenario -- the narrative description

Properties -- metadata to facilitate the use and understanding of the use case

Diagram -- If use case scenarios are carefully written and organized, diagrams are not essential, but they can provide a more understandable view of how a set of interrelated use cases fit together

The Scenario

A scenario is a sequence of interactions between an actor and the system

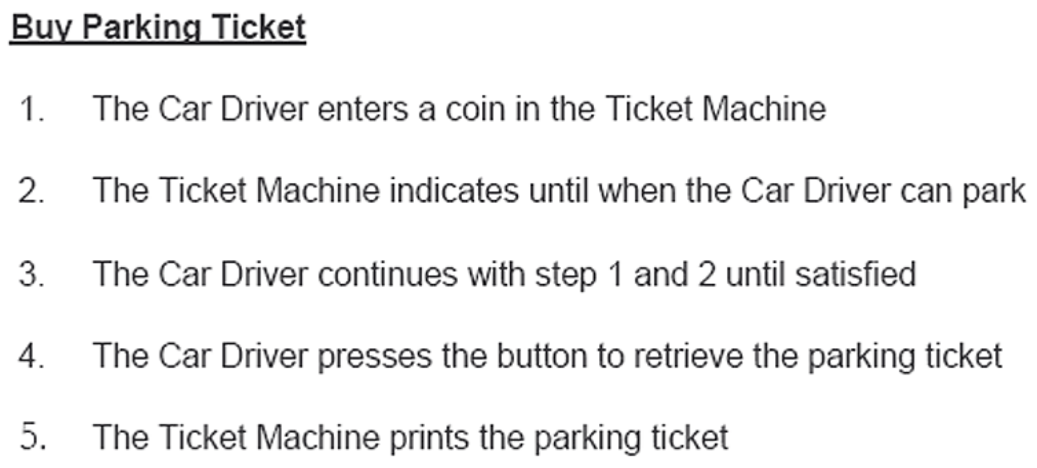
It defines the "success" or "happy" or "expected" situation after the actor initiates the first interaction until some "unit of functionality" has been produced

The interactions should connect the actor's goal to another actor or system that has some responsibility or capability to satisfy it

There should be no significant time gaps in the sequence of interactions

If iteration is required or allowed, it is typically specified simply and somewhat informally, not using programming language constructs

Simple Use Case Scenario with Iteration



Buy Parking Ticket

1. The Car Driver enters a coin in the Ticket Machine
2. The Ticket Machine indicates until when the Car Driver can park
3. The Car Driver continues with step 1 and 2 until satisfied
4. The Car Driver presses the button to retrieve the parking ticket
5. The Ticket Machine prints the parking ticket

Actors

Specifying an actor determines a point of view on the "system" - because the system is just a type of actor

Actors identify a role of a person, system or some other entity that is outside of the system

Actors initiate (or supply the "trigger" for) use cases to obtain something from the system or to satisfy some other goal

Documenting the Interactions [1]

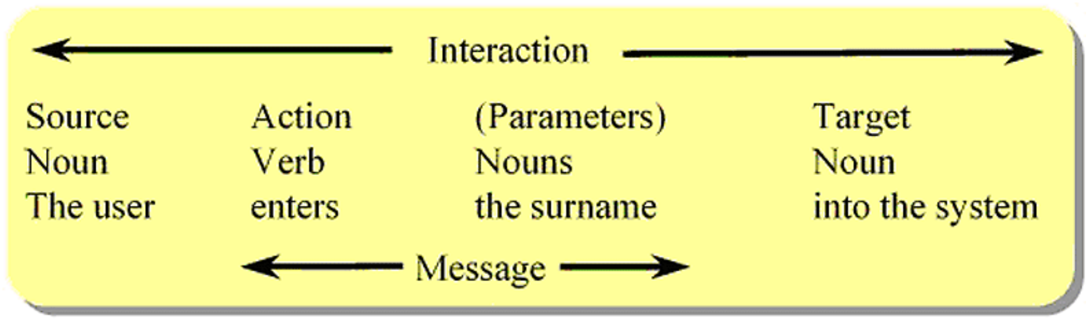
Interactions can often be thought of as information or a message being sent from the actor to the system (or vice versa)

This naturally implies a "actor-verb-noun" name structure for the use case (e.g., "Customer Places Order" or "Customer Cancels Order")

It can be helpful to qualify the noun (e.g. "In-stock inventory" or "International shipment")

Avoid generic verbs like "do," "process, or "get"

"Anatomy" of an Interaction



Documenting the Interactions [2]

Write simple declarative sentences using the active voice ("as though it works")

Passive sentences are harder to understand, and they usually omit the actor, which can create ambiguity or continuity problems

Numbered sentences are preferred to a more narrative, full paragraph style because it makes it easier to refer to exception cases

Time as an Actor

Most of the use cases will describe actions or interactions that don't occur at specified times

Other use cases are more predictable, "clock-driven" actions that start at scheduled times

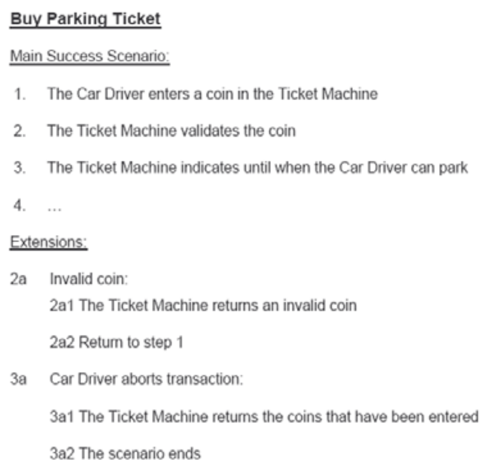
The actor in these use cases can be a "pseudo-actor" ("Schedule Manager" Publishes Monthly Calendar)

Exceptions and Extensions in Use Cases

Exceptions to the "success" or "happy" scenario -- where things don't go as expected -- are documented as an alternate route in the scenario that are usually called "extensions"

Statements should be numbered using a scheme that clearly indicates where they branch from the success scenario (e.g., 2a to follow statement 2)

Use Case Extension -- Example



Buy Parking Ticket

Main Success Scenario:

1. The Car Driver enters a coin in the Ticket Machine
2. The Ticket Machine validates the coin
3. The Ticket Machine indicates until when the Car Driver can park
4. …

Extensions:

2a Invalid coin:

2a1 The Ticket Machine returns an invalid coin

2a2 Return to step 1

3a Car Driver aborts transaction:

3a1 The Ticket Machine returns the coins that have been entered

3a2 The scenario ends

Inclusions or Sub-Flows in Use Cases

When a use case must invoke or otherwise "require the behavior" of another, the "called" use case is said to be "included" in the "calling" one

Inclusion is the mechanism for "use case reuse"

The convention for inclusion is to underline the name of the included case in the calling scenario

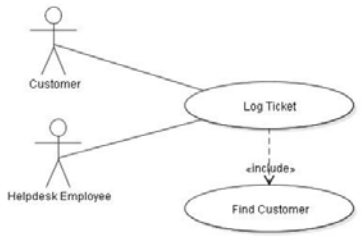
But don't overdo it -- if deeply-structured functional decomposition is built into your software, it will lose most of the benefits of object-oriented software design

Inclusions in Use Cases -- Example

Log Ticket

Main Success Scenario:

1. The Helpdesk Employee Finds Customer using its name or address
2. The Helpdesk Employee enters the details of the Ticket
3. …



Templates

"What you need to write down depends on the situation, especially on who is writing it and for what purpose"

The Oracle white paper contrasts "casual" and '"dressed" property templates

The amount of detail in a use case often increases over time, especially when they are developed in a top-down and iterative manner

Use Case Property Template ("Middleweight")

Scope

Stakeholders who care about the case

Primary Actor - the stakeholder who initiates the case

Description -- what the use case is supposed to do

Level

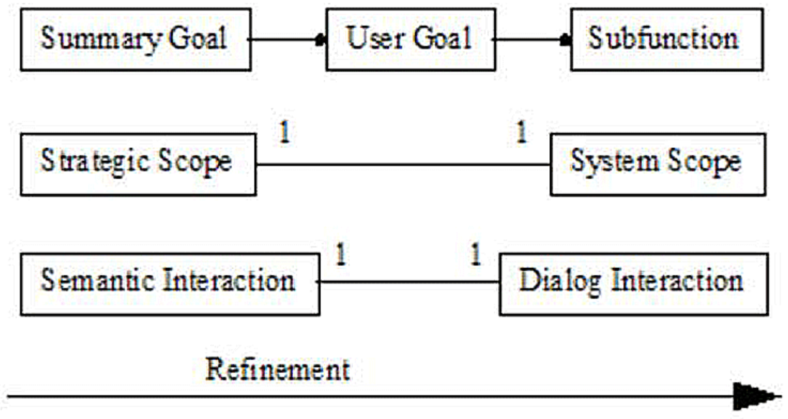
Preconditions - what conditions must be met before the case starts

Postconditions - what conditions must be met for the case to succeed

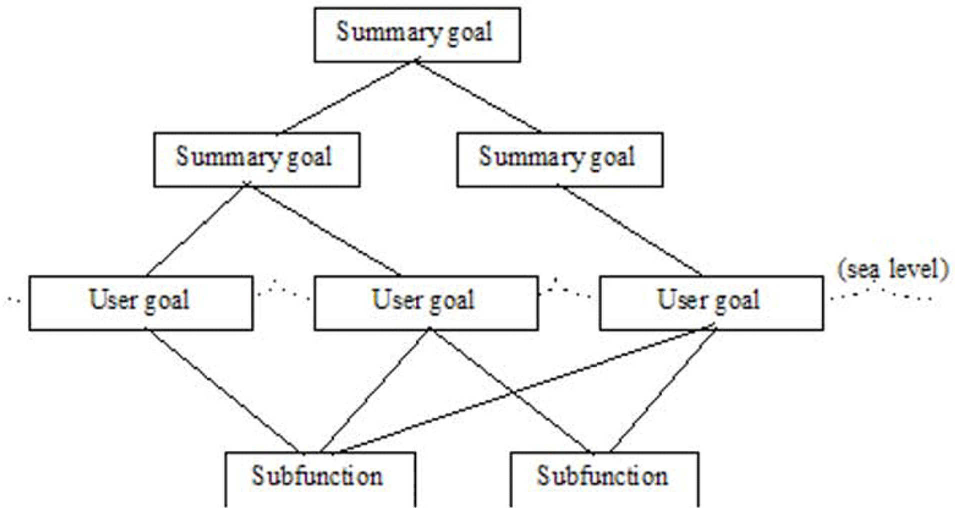
Trigger - the event that starts the case

Refinement of Use Cases

Work from the "top down," beginning with the highest-level or most abstract user / stakeholder goals



Use Case Structure -- The "Ship"



Scope and Levels of Use Cases

A challenge when creating use cases is maintaining a consistent level of abstraction

Some consensus that three levels of abstraction are useful, but not on the exact definitions or names

- System level, summary, or business goals -- this is the level when use cases document business processes
- User level goals, semantic level -- these correspond to "meaningful" activities and units of work
- Subfunction or technical goals -- often steps in a user level case, or interactions with specific functions of the system

Abstract use cases can describe abstract and high level goals, which can lead to some definitional confusion wrt the notion of "scenario"

The "Semantic Interface" of "Essential Use Cases"

It is usually unnecessary and inadvisable to write use cases that go past the "semantic interface"

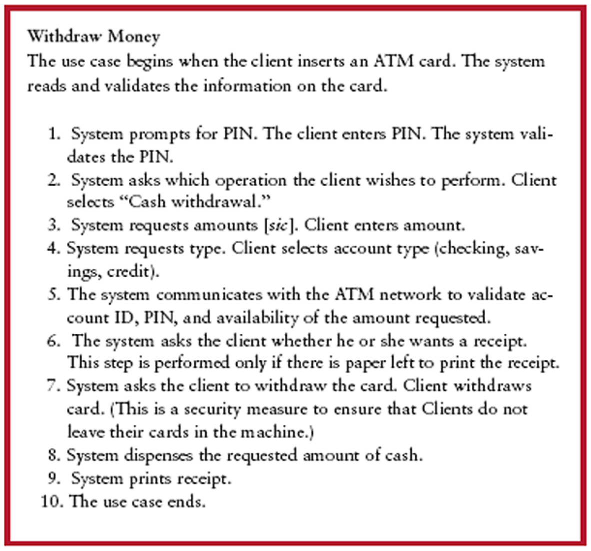
At this level, interactions with the system are described in terms of achieving work goals, in a technology-neutral and "context-free" way to preserve implementation options

This will usually mean taking a more abstract and idealized view than just describing the concrete behaviors that people are currently doing, which should give system designers more options in achieving the goals

Constantine calls these kind of use cases "essential" ones, resulting in "usage-centered" as opposed to "user-centered" design

Use cases that are "deeper" or "lower" in detail are not describing user goals

"Conventional" Use Case -- "Withdraw Money"

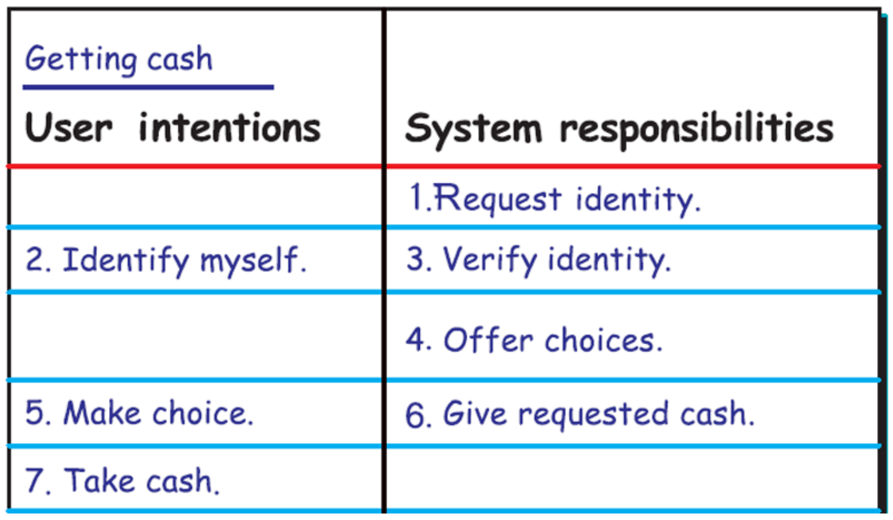


Withdraw Money

The use case begins when the client inserts an ATM card. The system reads and validates the information on the card.

1. System prompts for PIN. The client enters PIN. The system validates the PIN.
2. System asks which operation the client wishes to perform. Client selects "Cash withdrawal."
3. System requests amounts [*sic*]. Client enters amount.
4. System requests type. Client selects account type (checking, savings, credit).
5. The system communicates with the ATM network to validate account ID, PIN, and availability of the amount requested.
6. The system asks the client whether he or she wants a receipt. This step is performed only if there is paper left to print the receipt.
7. System asks the client to withdraw the card. Client withdraws card. (This is a security measure to ensure that Clients do not leave their cards in the machine.)
8. System dispenses the requested amount of cash.
9. System prints receipt.
10. The use case ends.

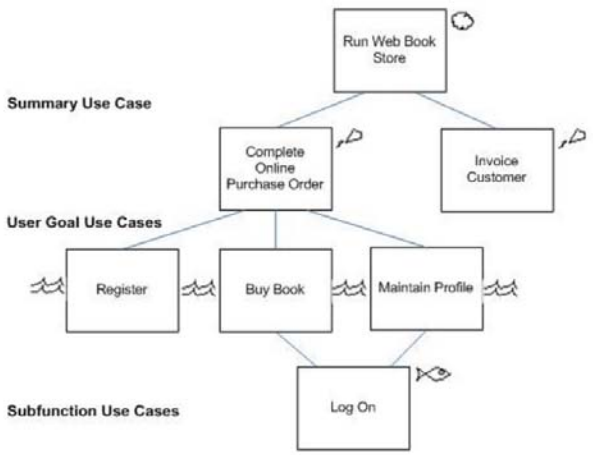
"Essential Use Case" -- "Withdraw Money"



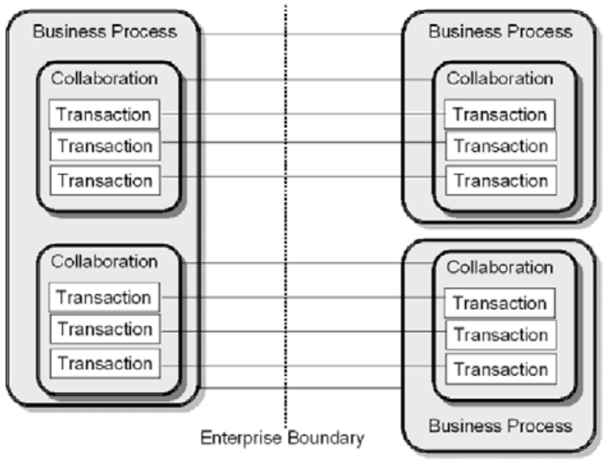
Getting cash

User intentions	System responsibilities
	1.Request identity.
2. Identify myself.	3. Verify identity.
	4. Offer choices.
5. Make choice.	6. Give requested cash.
7. Take cash.	

Levels of Use Cases -- Example



The ebXML Process Metamodel



Organizing Use Cases

By primary actor ("responsibility" diagram)

By summary use case

By implementation responsibility or schedule

By subject area

Prioritizing Requirements

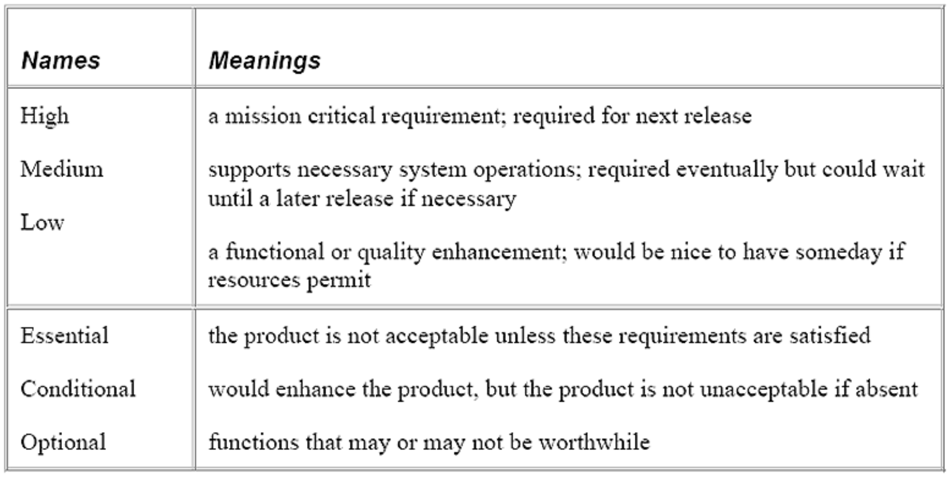
Any project with resource or schedule constraints has to set the relative priority of any requested features, use cases, or requirements

Clear priorities are essential in setting stakeholders' expectations, resolving conflicts and making tradeoffs, and planning for phased development/deployment

Priorities should be specified using a "named" value scale, not a numerical one

For large or contentious projects requirements should be prioritized using a structured, quantitative approach the relative value and cost of each requirement so that the greatest value can be produced with the smallest total cost

Some Prioritization Scales



Names	***Meanings***
High	a mission critical requirement; required for next release
Medium	supports necessary system operations; required eventually but could wait until a later release if necessary
Low	a functional or quality enhancement; would be nice to have someday if resources permit
Essential	the product is not acceptable unless these requirements are satisfied
Conditional	would enhance the product, but the product is not unacceptable if absent
Optional	functions that may or may not be worthwhile

MoSCoW Priorities

M - Must have this; non-negotiable; without this the project is a failure; all of the Ms should form a coherent set of functionality

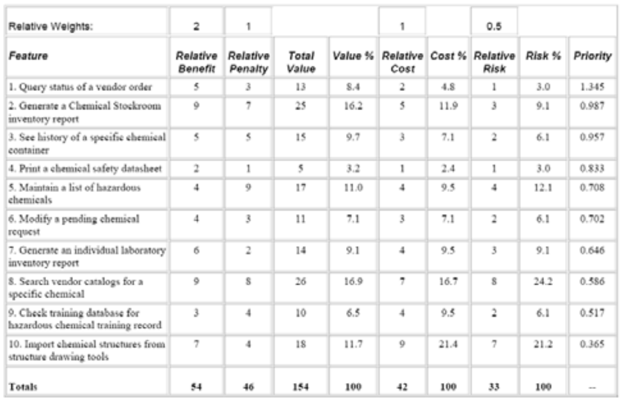
S - Should have this

C - Could have this

W - Won't have this in the current phase/release/product by would like it in the future

Wiegers' Method

From "First Things First: Prioritizing Requirements" by Karl Wiegers (http://www.processimpact.com/articles/prioritizing.pdf)



Relative Weights:	2	1			1		0.5		
Feature	***Relative Benefit***	***Relative Penalty***	***Total Value***	***Value %***	***Relative Cost***	***Cost %***	***Relative Risk***	***Risk %***	***Priority***
1. Query status of a vendor order	5	3	13	8.4	2	4.8	1	3.0	1.345
2. Generate a Chemical Stockroom inventory report	9	7	25	16.2	5	11.9	3	9.1	0.987
3. See history of a specific chemical container	5	5	15	9.7	3	7.1	2	6.1	0.957
4. Print a chemical safety datasheet	2	1	5	3.2	1	2.4	1	3.0	0.833
5. Maintain a list of hazardous chemicals	4	9	17	11.0	4	9.5	4	12.1	0.708
6. Modify a pending chemical request	4	3	11	7.1	3	7.1	2	6.1	0.702
7. Generate an individual laboratory inventory report	6	2	14	9.1	4	9.5	3	9.1	0.646
8. Search vendor catalogs for a specific chemical	9	8	26	16.9	7	16.7	8	24.2	0.586
9. Check training database for hazardous chemical training record	3	4	10	6.5	4	9.5	2	6.1	0.517
10. Import chemical structures from structure drawing tools	7	4	18	11.7	9	21.4	7	21.2	0.365
Totals	**54**	**46**	**154**	**100**	**42**	**100**	**33**	**100**	--

Readings for 29 September

Sara Corbett, “Can the cell phone end global poverty,” New York Times, 13 April 2008.

Brigitte Jordan and Brinda Dalal, “Persuasive encounters: Ethnography in the corporation,” Field Methods, 2006

David Siegel & Susan Dray, “Avoiding the next schism: Ethnography and usability,” Interactions, March-April 2005.

Donald Norman, “Workarounds and hacks: The leading edge of innovation” Interactions, July-August 2008.