Use Case Evaluation (UCE): A Method for Early Usability Evaluation in Software Development

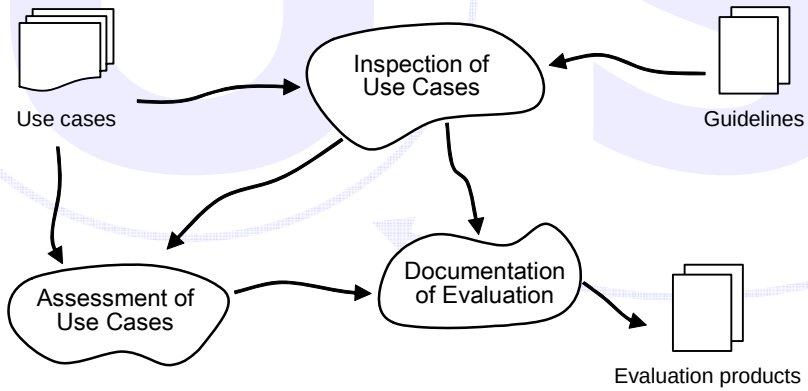
Kasper Hornbæk
Department of Computer Science
University of Copenhagen

Rune Thaarup Høegh, Michael Bach Pedersen & Jan Stage
Department of Computer Science,
Aalborg University

Aim and Motivation

- Use Case Evaluation (UCE): Usability evaluation based on use cases
- Usability problems are cheaper to solve early in the development process
- Identifying usability problems early in the process is difficult with the current software development practice
  - Usability work is usually separated from core software development activities
  - Most usability work takes place late in the software development process
- Use cases
  - Available early in the development process
  - Already part of many development methods
  - Valuable means for integrating usability in the software development process

Use Case Evaluation (UCE)



- Fully dressed use cases (Cockburn) are recommended
- List of guidelines assist the inspection (11)
  - Based on heuristics from Heuristic Evaluation (9)
  - Supplemented with guidelines from other methods (2)
- Evaluation product
  - Assessment of the usability of the system; a list of usability problems
  - Assessment of the quality of the use cases

Usability Problem

- Definition: ”An aspect of the system that will hinder or delay the user in completing a task, be difficult or impossible for the user to understand, or cause the user to be frustrated”.

Example of Fully Dressed Use Case (partial)

Use case 02: Registration of blood glucose measurement (cabled connection)

CHARACTERISTIC INFORMATION

Goal in Context: The user wants to transfer the result of his newly taken blood glucose measurement from the blood glucose measurement device to the HealthMonitor.
Scope: Enterprise
Level: Summary
Preconditions: A successfully conducted blood glucose measurement. A successfully installed HealthMonitor (Use case 01)
Success End Condition: The blood glucose measurement result is successfully transferred to the HealthMonitor
Failed End Condition: The blood glucose measurement result is not transferred to the HealthMonitor
Primary Actor: A person with a physical health condition that needs daily monitoring.
Trigger: The user wants to register a blood glucose measurement

MAIN SUCCESS SCENARIO

1. The step-by-step instruction for transferring blood glucose measurements from the measuring device to the HealthMonitor is found
2. The two devices is connected using the relevant cable
3. The blood glucose measurement device identifies the connection, and shows the message "PC" in the display
4. The HealthMonitor shows in the display that a transfer is ongoing
5. The HealthMonitor shows in the display the amounts of measurement results that were available in the blood glucose measurement device, and the HealthMonitor also shows how many of those measurement results that have been transferred. The information on the display of the HealthMonitor is further more read aloud to the user by the HealthMonitor.

Procedure for Inspection of Use Cases

- One or more evaluators

- Brainstorm
  - Use cases inspected one by one
  - Note problems that may be predicted
- Systematic inspection based on 11 predefined guidelines
  - Use cases inspected one by one
  - Note problems that may predicted while employing the guidelines
    - Where may a guideline be breached?
- Fruitful to go over all use cases at least twice
- Asses the overall quality of each of the use cases

- If more than one evaluator
  - Create a joint problem list

Guidelines for Use Case Evaluation

No	Guideline	Explanation
1	Visibility of system status	The system should always keep users informed about what is going on, through appropriate feedback within reasonable time
2	Match between system and the real world	The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions and make information appear in a natural order.
3	User control and freedom	Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.
4	Consistency and standards	Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.
5	Error prevention	Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Eliminate error-prone conditions or handle them gracefully.
6	Recognition rather than recall	Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another.
7	Flexibility and efficiency of use	Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users.
8	Help users recognize, diagnose, and recover from errors	Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
9	Avoid hard mental operations and lower workload	Do not force the user into hard mental operations and keep the user's workload at a minimum.
10	Avoid forcing the user to premature commitment	Do not force the user to perform a particular task or decision until it is needed. Will the user know why something must be done?
11	Provide functions that are of utility to the user	Consider whether the functionality described is likely to be useful to users and whether functions/data are missing.

Empirical Study I

- 4 evaluators with 2-8 years of experience in HCI after obtaining masters degree
- Health care application (HealthMonitor)
  - Monitors elderly persons' medical conditions in their home

- Four fully dressed use cases was described for the HealthMonitor
  - Avg. 472 words long and consisted 6-19 steps

- Evaluators received descriptions of
  - The experimental procedure to be followed
  - The UCE method
  - The four use cases
  - An explanation of the HealthMonitor's general use context

Empirical Study II

- Usability problems reported by
  - Title
  - Place(s) where found
  - Related use case
  - Guideline breached
  - Severity rating (cosmetic, serious or critical)
- Matching of problems into a joint problem list
- General assessment of the use cases
- Evaluators opinions on using the UCE method

- Comparison with think-aloud usability evaluation of the HealthMonitor
  - Five user sessions
  - Analysed by Instant Data Analysis (IDA)
  - Analysed by conventional video based analysis

Results I

Table 2. Problems predicted with the use case inspection (UCE) in relation to those found in think-aloud tests (TA). Percentages are relative to the total number of problem types (93).

Predicted with UCE	*Found with TA*	*Number of problem types*	*Problem category*	*Number of problem types in category*
YES	YES	22 (24%)	-	-
NO	YES	32 (34%)		
			Impossible or hard to predict	20 (21%)
			Predictable but missed	12 (13%)
YES	NO	39 (42%)		
			Relevant problem	14 (15%)
			Problems avoided in UI	9 (10%)
			Not a problem	16 (17%)

Results II

Table 3. Usability problems categories distributed by areas ($N = 93$). Note that a problem may be related to several areas, so column sums are higher than the number of problems.

Types	*Total*	*Found with TA*	*Found with UCE*
Dialogue	38	25	30
External factors	4	1	3
Graphical User Interface	27	15	20
Installation of equipment	38	29	19
Procedure / task flow	38	18	26

Discussion

- Large portion of usability problems identified through both UCE and conventional method
- Several other usability problems were assessed as being useful

- Additional benefits from inspection based on use cases
  - Early focus on usability issues in a natural way
  - May uncover and emphasize non-functional requirements
  - May improve overall quality of the use cases
- Still need to be empirically documented

- Potential improvements on UCE:
  - Style of writing use cases
  - Used guidelines
  - Inspection across use cases for inconsistencies

Limitations & Further Work

- Did not assess impact of UCE evaluation in a real-life context
- Not a strict experiment
  - Participants not randomly assigned to think-aloud or UCE
  - Partly conducted by authors who also had developed the method

- Follow-up study by other researchers necessary
  - Possibly with non-expert participants

- Despite the limitations
  - Our paper suggest that inspection of use cases may help introduce effective usability evaluation early in the software development process

Questions?