HCI Exam - 80%

18/05/2016 - D Day

2PM (14:00) in Melbourn

Check Seat Number @ <http://examtimetables.cit.ie/>



[KNOWLEDGE](https://www.youtube.com/watch?v=0GIwTG8V-Ko)

**Good Luck. Have Fun**

**See everyone in august**

KEEP PAPER SUMMARIES TO ONE PAGE AT MOST.

**DON'T COPY AND PASTE TEXT**

**NO WALLS OF TEXT EITHER CAUSE BRUH -\_-**

## Paul’s Notes

<http://mcom.cit.ie/staff/Computing/prothwell/home/tfohcida.html#Articles>

## 2014 Paper

<http://exampapers.cit.ie/PastExams/Computing/KWEBD_8_Y4/2014%20Semester2/KWEBD_8_Y4%20SOFT8010%20Theoretical%20Foundations%20of%20Human-Computer%20Interface%20Design.pdf>

## 2015 Paper

<http://exampapers.cit.ie/PastExams/Computing/KWEBD_8_Y4/2015%20Semester2/KWEBD_8_Y4%20SOFT8010%20Theoretical%20Foundations%20of%20Human-Computer%20Interface%20Design.pdf>

<http://mcom.cit.ie/staff/Computing/prothwell/hci/mscsd/exercises/WEB4HCIRepeat2015.pdf>

# Commonly Asked Exam Questions

**2014**

b) Summarise the main elements of **Cognitive Load Theory** and explain the three cognitive load types.

Cognitive load theory was designed to provide guidelines intended to assist in the presentation of information in a manner that encourages learner activities that optimize intellectual performance. It’s common knowledge that the mind can become overpowered when keeping track of too many things at once. What is ‘too many’? A paper written in 1956 by Miller looked into quantifying this. What he found was there is a ‘magical number’ 7±2 which means the average human can handle 7±2 items in conscious working memory. The cognitive load theory provides a classification of the types of cognitive load and they are as follows:

* **Intrinsic cognitive load** is the effort associated with a specific topic. *Tic Tac Toe vs Chess*
* **Extraneous cognitive load** refers to the way information or tasks are presented to a learner. *Extra time spent finding something on the screen.*
* **Germane Cognitive Load** (GCL): this load is a necessary part of the learning. The designer of the instructional material will purposely create problems to solve and the like to ensure that the learner is exposed to the correct kind of information and practice required to learn. E.g. *exercises at the end of each chapter in a textbook.*

*\*\*\*Could probably talk about recognition instead of recall for this question\*\*\* go for it!*

[20 marks]

**2015**

b) Summarise the main elements of **Cognitive Load Theory** and explain the three cognitive load types.

[30 marks]

**2014**

a) Write a short proposal for the introduction of ONE of the following methods into the software development process: ‘**Cognitive Walkthrough**’ OR ‘**Context of Use**’. In your proposal give a brief summary of the method and present arguments in favour of its introduction. You should outline the benefits and be realistic about the costs. [30 marks]

### **Context of use**

* who the target users are,
* what kinds of tasks these users will wish to use the software for?
* what kind of equipment and environment will the software be used in.?

1. Environment

* Physical Environment

2. Tasks

* Tasks user will do every day

3. User Category

* User Role
* Indirect User
* Direct user
* Supporting user
* Monitoring user
* User skills & knowledge
* Physical Attributes (Age, gender)
* Job characteristics (training, education levels, computer expertise)

Why we should hold a context of use *(don’t talk about cons I guess in this answer)*

1. Helps find target users, tasks which the user will use, the type of equipment used as well as the environment they are used in.
2. Useful later when evaluating the prototypes developed using heuristic analysis and cognitive walkthrough.
3. Aids usability evaluation.
4. Aids requirements capturing.

### **Cognitive Walkthrough**

[Suh Dude version](https://www.youtube.com/watch?v=pIHYPaoh79I): The **cognitive walkthrough** is a usability evaluation method in which one or more evaluators work through a series of tasks and ask a set of questions from the perspective of the user.

A cognitive walkthrough can be psychologically very difficult. Audience members may be criticising the walkthrough which can be a tough pill to swallow. However hard that maybe cognitive walkthroughs are very useful in finding problems.

Method to running a successful cognitive walkthrough:

1. **Do not explain the prototype**, the screens of the prototype should explain themselves.
2. **Stick to the defined tasks** of the walkthrough, do not complicate the task as this may lead to confusion.
3. **Take notes** as the users run through the walkthrough. Take this notes visible to make the audience member feel like a special bunny
4. **Audience questions are just problems to note**. This is not a requirements gathering session. Not the question/problem raised and move on.
5. **Don’t try and defend your prototype**. It may have taken you ages to develop but the audiences criticisms are what we want so don’t blame the user if they’re agitated with the prototype.
6. **The quality of the prototype matters**. Audience members will much prefer a good looking prototype to a rushed, hand drawn one. More sophisticate prototypes make it easier for them.
7. **Well-defined tasks are important**, otherwise the audience will go onto to discuss the general prototype instead of sticking to their defined task.
8. **Audiences may need training** in the domain of the software (depending on how complex it is).
9. **Keep the audience sweet** throughout the walkthrough. Like previously mentioned don’t be defensive about the prototype, open yourself up to criticisms and make the audience feel like they're really helping.

Why we should hold a cognitive walkthrough *(don’t talk about cons I guess in this answer)*

1. Cognitive walkthroughs find task orientated problems
2. Helps define user’s goals and their assumptions about the software.
3. Can be used early in the development process and keep production on the right path.

**2015**

a) Write a short proposal for the introduction of ONE of the following methods into the software development process: ‘**Cognitive Walkthrough**’ OR **‘Context of Use**’. In your proposal give a brief summary of the method and present arguments in favour of its introduction. You should outline the benefits and be realistic about the costs. [30 marks]

**2014**

b) In her paper on ‘**user involvement**’, **Kujala** identifies ‘access to users’ as a recurring difficulty. Briefly explain this problem. [10 marks]

# Kujala “Access to users”

* **Gaining direct access to users is difficult**
* **It is difficult to find users willing to let you watch them work.**
* It can be a tough sell to get developers out of the office and into the field…
* Users are reluctant to let the researchers watch them work
* **finding users willing to let the researchers watch them work took much longer than finding users for typical usability sessions**
* users rarely did real work when the researchers came to visits. It was **difficult to arrange their day so that they had real work to do when observed**
* identifying appropriate users was difficult
* **obtaining access to users and motivating users**
* designers spent lots of time contacting users and arranging meetings
* users were very busy
* **some users lacked confidence or motivation and were reluctant to talk to the developers**
* identifying appropriate users
* one team “moved their offices so that they were in constant contact with users

Fully discuss how, in a phased introduction of usability techniques, ‘**support calls**’ and ‘**site visits**’ might be used to help in this. [20 marks]

c) Why should the introduction of usability engineering into an existing software development process be **phased**? [10 marks]

**2015**

c) Why should the introduction of usability engineering into an existing software development process be **phased**? [10 marks]

In her paper on ‘**user involvement**’, **Kujala** identifies ‘access to users’ as a recurring difficulty. Briefly explain this problem. [10 marks]

Fully discuss how, in a phased introduction of usability techniques, ‘**support calls**’ and ‘**site visits**’ might be used to help. [20 marks]

**Support Calls**

* problem identification, leading to a revised design
* improved usability testing results
* Calls fell from 45 minutes to 10 minutes
* Motor Company changed 90% of their accounting software for their small car dealerships as a result of usability testing, before it took the car dealers three help-line calls merely to get started

**Site Visits**

* Greater understanding of the product
* Downstream tech­nology understanding
* Improved mutual under­standing and work relationships among all the stakeholders

**Pros**

* customers usually viewed the visits as a form of respect and appreciation
* the decisions of software developers were more likely to match the needs of the users
* determin­ing priorities of the product

**Cons**

* the amount of raw data collected during the study can be overwhelming
* impacting the design can be difficult if field-oriented methods are not an accepted part of the development process
* It was difficult to arrange their day so that they would have real work to do when observed.
* users don’t do much work when the development team are visiting because they are a distraction.

**2014**

b) Briefly explain the role of a mental model in a user’s interaction with software. [10 marks]

“The system image must make the design model clear and consistent.” With the aid of a diagram, fully explain the meaning and relationships between the following terms: **system image**, **design model,** **mental model**, **conceptual model**. [20 marks]

**2015**

b) Briefly explain the role of a **mental model** in a user’s interaction with software. [10 marks]

“The system image must make the design model clear and consistent.” With the aid of a diagram, fully explain the meaning and relationships between the following terms: **system image**, **design model**, **mental model**, and **conceptual model**. [20 marks]

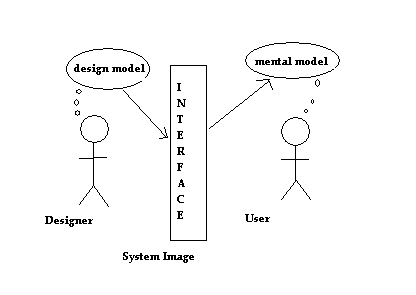
The goal of a mental model is to discover what a user knows about a system and how they reason about it’s function. With this it is possible to predict learning time, likely errors and relative ease of tasks.

Designers have their own model of a system, the **design model.**

Designers can write down a description of their design model and the resulting document is referred to as the **conceptual model.** It describes how the designer understands a system is to function. **Design model -> conceptual model**

The **system image is the actual implementation** or embodiment of the design (including interface components, documentation, instructions, and labels).

The **user's mental model is built through interaction** with the system interface



**2014**

c) Many factors influence the formation of a user’s mental model of a software application. Fully discuss how TWO of the following factors have such an influence: i) **Causality**; ii) **Perceived affordances**; iii) **Interface Metaphors**. [40 marks]

**2015**

c) Many factors influence the formation of a user’s mental model of a software application. Fully discuss how TWO of the following factors have such an influence: i) **Causality**; ii) **Perceived af**

**fordances**; iii) **Interface Metaphors**. [40 marks]

**Causality**

The visible effects of actions will lead people to assume a causal connection. When you do A and B happens immediately afterwards and close by, you tend to assume that A caused B. Mental models are fundamentally models of what causes what. If an interface is careful to ensure that the feedback given suggests the right link between causes and the effects they cause, then users will be able to build accurate models.

In all things when people are trying to understand something they observe causality. For example, "if I push this button the document is printed." So there should be a clear connection between actions and effects so that users can make the right connections. Multiple factors and complicated pre-requisites tend to blur the connections and so should be minimised or made obvious e.g. allowing the 'paste' option only when there is something to paste.

**Perceived Affordances** are the actions a user perceives to be possible. A perceived affordance is a relationship between an agent (the user), an object (the UI widget)and the agent’s task (the job they are trying to do).A user will view a UI widget with a specific job in mind; if the widget looks like it allows that job to be done then they will try it.

Definition: Perceived affordances are the apparent properties of objects that indicate the sorts of operations and manipulations that can be done to those objects.

A chair affords sitting

A Button affords pushing … etc.

**2014**

c) “**Prototypes** are valuable tools in developing user interface designs especially if care is taken to ensure that they are built *early* and *often* using *simple* techniques.” Discuss this claim paying particular attention to the phrases in italics. [30 marks]

**2015**

b) “**Prototypes** are valuable tools in developing user interface designs especially if care is taken to ensure that they are built *early* and *often* using *simple* techniques.” Discuss this claim paying particular attention to the phrases in italics. [30 marks]

**Early**

Importance of early prototyping - easier to redesign; - weeds out problems before implementation and release; - aid in requirements gathering; - encourage discussion among designers; - avoid people’s reluctance to discard the prototype.

**Frequent**

Importance of frequent prototyping - allow for successive improvements; - successively more complex prototype techniques so get variety of feedback; - aid in further requirements gathering with more informative versions; - encourage experimentation.

**Simple**

Importance of simple prototyping - Cheaper, quicker, and easier to build -> so more chance they’ll be done and done early and frequently; - not as much invested in the prototype work so -> easier to criticise, easier to discard, more likelihood of changing based on criticism.

# 

# 

# Section 1 :Introduction to Human-Computer Interaction

### HCI Definition

* An area of study in computing which deals with the **design** of computer systems as **usable tools** which support **people** so that they can carry out their **tasks efficiently and safely.**
* Computer is a tool in the **achievement of a user’s goals.**

### Five Usability Attributes - *L.E.M.E.S*

1. **L**earnability………………Ease of learning for **novice users**.
2. **E**fficiency…………………Steady-state performance of **expert users**.
3. **M**emorability……………..Ease of using system by **intermittent/casual users**.
4. **E**rror Rate.……………….Error rate for minor and catastrophic errors.
5. **S**ubjective Satisfaction…...How pleasant system is to use/aesthetics.

### Typical Ways of Measuring Usability - *L.E.M.E.S*

* **Learnability**: pick novice users of system, measure time to perform certain tasks. Distinguish between no/some general computer experience.
* **Efficiency**: decide definition of expertise; get sample expert users (difficult), measure time to perform typical tasks.
* **Memorability**: get sample casual users (away from system for certain time), measure time to perform typical tasks.
* **Errors**: count minor and catastrophic errors made by users while performing some specified task.
* **Satisfaction**: ask users' subjective opinion (questionnaire, interview), after trying system for real task.

### Goals of HCI (U.E.T.S.F)

* Improve the **usability** of software: Make software easier to learn and use - (not always in harmony).
* Improve the **effectiveness** of software: Make software better at doing the work it does.
* Improve the **task-efficiency** of software: Improve the efficiency of the user’s achievement of their task (i.e. not the efficiency of the program code being used as a tool in that task but the steps and structure of how the user makes it do the job.)
* Improve the **safety** of software: A well designed interface reduces errors. Sometimes errors can be dangerous.
* Improve the **functionality** of software: Making systems better equipped with the tools necessary for tasks. (e.g. providing searching and sorting functions as standard.)

# Section 2: Human Factors/Psychology

## Constructivist Theories

Seeing is a constructive activity where our minds construct what is perceived using:

1. information from the environment (based on sensed data).
2. previously stored knowledge.

##### **Constructivist Theory: Gestalt Theory**

Breaks into 5 sections (P.S.C.C.S)

* **Proximity**: We tend to see items that are close together as a ‘group’ (e.g. constellations of stars).
* **Similarity:** We tend to see items that are similar in colour or shape or size (etc.) as a ‘group’ (e.g. on a map with motorways marked in red, small roads in yellow).
* **Closure:** We tend to assume that unfinished known shapes are actually finished (e.g. when reading handwritten O’s)
* **Continuity:** We tend to assume that a partially hidden square is in fact square, it’s just we can’t see it all (e.g. overlapping notices on a notice board.)
* **Symmetry:** Symmetry is fundamental to how we view the world; we tend to assume that the space between two symmetrical borders is somehow more of a thing than the spaces to either side.

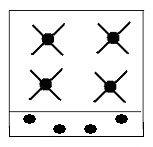
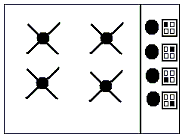
## Ecological Theories: Affordances

#### Perceived and Real Affordances

Summary: Affordance theory states that the world is perceived not only in terms of object shapes and spatial relationships but also in terms of object possibilities for action (affordances) — perception drives action.

* A **false affordance** occurs where it appears as if something is possible but in fact it is not (e.g handle on a push door or pressing a button that doesn’t do anything, like using your TV remote to turn the TV on, but it doesn’t work for some reason).
* A **perceptible affordance** occurs where it appear as if something is possible and in fact it is (e.g handle on a pull door).
* A **correct rejection** occurs where it appear as if an action is not possible and in fact it is not (e.g no handle on a push door).
* A **hidden affordance** occurs where it appear as if an action is not possible but in fact it is (e.g no handle on a pull door, looking at a beer won't tell you to open up another one ).

**Diagram to use : The cooker and the nobs layout.**



In the right hand example, the positioning of the knobs vis-à-vis the burners helps us

to identify the correct knob/burner pair (Gestalt).

The fact that a knob looks like a knob helps us understand that it should be turned

and not pulled (affordance).

### Human Attention

**Focused or divided:** concentrating on one task or trying to keep tabs on many different sources of input.

**Voluntary or involuntary:** Turning attention to something or being distracted.

**Crowded room example.**

### Structuring information (R.B.G.A)

1. Relevance: e.g. a stepped interface
2. Balance of information
3. Grouping and Ordering
4. Attention grabbing

### Human Memory

#### **Cognitive Load Theory (I.E.G)**

Bruh version : In cognitive psychology, cognitive load refers to the total amount of mental effort being used in the working memory.

It is a matter of common knowledge that the mind can get overpowered when trying to keep track of too many things at once. But what is ‘too many’? A famous paper in psychology described experimental results to quantify this. Miller’s 1956 paper ‘[The magical number 7 ± 2](http://mcom.cit.ie/staff/Computing/prothwell/hci/papers/MillerMagic.pdf)’[2](http://mcom.cit.ie/staff/Computing/prothwell/hci/mscsd/Notes/HCID2.html#sdfootnote2sym) claimed that the average human could only comfortable handle 7±2 items in conscious working memory at any one time.

* **Intrinsic cognitive load** is the effort associated with a specific topic. *Tic Tac Toe vs Chess*
* **Extraneous cognitive load** refers to the way information or tasks are presented to a learner. *Extra time spent finding something on the screen.*
* **Germane Cognitive Load** (GCL): this load is a necessary part of the learning. The designer of the instructional material will purposely create problems to solve and the like to ensure that the learner is exposed to the correct kind of information and practice required to learn. E.g. *exercises at the end of each chapter in a textbook.*
* *Germane load is the a mixture of between* Intrinsic Extraneous so you make it isnt two hard of a an interface to learn and not to easy. That way the user can learn. Changle and award.!!!

### Recognition is better than Recall

Less learning off commands and more learning what a button does

Useful icons that have meaning , so they understand that a house is a house will bring you home

Such as **Mappings**.

The relationship between the representational form (concrete/abstract symbols) and the underlying concept (concrete object or abstract idea) is a mapping

##### Four types of mapping used (R.E.S.A)

* 1. **Resemblance**: image resembles its real-life counterpart. E.g. *Road sign of falling rocks indicate fallen rocks on the road ahead*
  2. **Exemplar**: where the image shows some concrete object that exemplifies the concrete object intended. E.g. *Road sign of Knife and Fork indicate a restaurant is up ahead*
  3. **Symbolic**: where the image shows a symbol used to indicate something abstract. E.g. *Picture of a cracked glass indicates item is fragile (prone to breakage)*
  4. ***Arbitrary***: where the image used bares no common-sense relation to what it refers to what it refers to and therefore must be learned. E.g. *Radiation symbol ☢*

## Interface Metaphors

#### Metaphors in Interfaces

Interface metaphors can be: (V.V.O.P)

* verbal (e.g. instructor says ‘imagine the file is being stored in a folder in a filing cabinet’)
* visual (e.g. Apple Mac trash can).
* overall (e.g. desktop in Apple Mac, PC.)
* partial (e.g. menu, window, scroll bar, etc.)

#### Generating Metaphors

Erickson proposes a methodology for generating interface metaphors:

1. Understand the primary functions of the system and their implementation;
2. Identify the areas of the system with which users have problems - observation of users using prototypes;
3. Finally generate metaphors which match the model for 1 and 2.

Sometimes metaphors don’t scale well – e.g. if the amount of data increases.... (e.g. website map, larger icons for larger files).

Metaphors are dependent on cultural background.

## Mental Models

Research here is motivated by the desire to discover what users can know about a system and how they reason about its function. If this is known it may be possible to predict:

* Learning time,
* likely errors,
* and the relative ease of tasks.

**Diagram reminder: Pizza Experiences from class.**

You’re starving, and you have a frozen pizza. You need the oven temperature at 200 degrees as fast as possible. Do you turn the thermostat to:

1. 200?
2. The top?
3. 100 then 150 then 200?

#### **Two types of mental model**

1. **Structural** : Structural models model how the device works in terms of the internal mechanics of its component parts (‘how-it-works’). Example of a bicycle will help you to fix the brakes but it won’t tell you how to ride a bike.

Also you could know how to fix a car but not how to drive.

* *limited* by the fact that they contain no procedural knowledge and hence cannot inform the user of how to use the device;
* *more general* because they allow the user to infer the effect of any actions on the device;
* *most useful in diagnostics and troubleshooting*;

1. **functional**:Functional models model the procedural knowledge necessary to use the device (‘how-to-use-it’). Example. a functional model of a bicycle will tell you how to ride a bike but it won’t help you to fix the brakes.

* *built up from knowledge of similar domains*;
* *limited* because they are structured around a particular set of tasks (context dependent);

**How are mental models formed?** Though the user’s interaction with the system including the factors below

**Mapping**

As we have seen, graphical encodings must be mapped onto the idea they represent. Some mappings are natural and suggest the right idea and these lead to accurate mental models. Some are weird and suggest the wrong idea – these lead to problems.

(p.c.f.c)

Examples:

* resemblance mappings employing concrete objects are mostly straightforward;
* volume controls going from L to R or bottom to top; cooker example

**Perceived affordance:** GUI widgets suggest certain operations. So users will assume those operations have to fit in somehow to their mental model. Gui widget needs to suggest the right operations or else the user will get an inaccurate mental model

**Constraints**, on the other hand, are the *inverse* of affordances. They limit the way an object can be used. Constraints can be used to avoid usage errors or minimize the information to be remembered.

**Familiarity** with similar devices Familiarity allows the transfer of previous experience

**Causality** The visible effects of actions will lead people to assume a causal connection. When you do A and B happens immediately afterwards and close by, you tend to assume that A caused B.

# 

# Section 3:Usability Engineering

Usability Engineering is an *iterative process* to improve usability of a system.

* Requirements gathering:
  + Competitive Analysis
  + Know the User (Context of Use)
* Design:
  + Task Analysis
  + Prototyping
* Usability Evaluation:
  + Heuristic Evaluation
  + Cognitive Walkthrough
  + User Testing

## Competitive Analysis

Competitive analysis is a requirements gathering exercise.

* Analyse competing products.
* “Intelligent borrowing” of ideas from other systems.
* Identify the good and bad features and record reasons for choice.

## Context of Use

Performing a Context of Use (CoU) analysis helps you to find out:

* Who are the target users?
* What are the kinds of tasks for which these users will wish to use the software?
* What kind of equipment will be used and in what kind of environment?

## The Application of Context of Use

There are two major uses for Context of Use analysis:

* As an aid to requirements capture;
* As an aid to usability evaluation.

## Elements of the Context of Use

* WHO will use the software (User Analysis)?
* WHAT will they do with the software (Tasks)?
* WHERE will they use the software (Environment)?

## Prototyping

**Paper Mockups**

* Sketches
* Printouts
* Early feedback
* Maximum feedback for minimum effort
* Do not write code

**Interactive Prototypes**

* Scan in hand drawn sketches
* Clickable elements - HTML
* Retains throwaway, casual look - encourages critique and discussion

**Working Prototype**

* Fake data
* Simple algorithms - who knows what the fuck this means
* Prototyping tools

**Dimensions of Working Prototypes**

**Vertical:** In depth functionality for a few features

**Horizontal:** Full interface, no functionality

**Scenario**: Only features + functionality along a specific path

**Parallel Design:** Explore design alternatives - designers should work independently, then compare draft designs.

**Early Prototyping**

* Easy to redesign.
* Weeds out problems before implementation.
* Aids in requirements gathering.
* Encourages discussion and critique.

**Frequent Prototyping**

* Successive improvements
* More complex prototypes.
* Experimentation.

**Simple Prototyping**

* Cheap, quick, easy.
* Not as heavily invested.
* Easier to critique and discard

## Usability Evaluation

* **Usability Inspection** : Inspection of interface design using heuristics and judgement (no user tests).
* **Usability Testing**: *Empirical* (observation and experimentation) testing of interface design with real users.
* **Heuristic Evaluation**: Small set of evaluators examines interface and judges its compliance with recognised usability principles.
* **Cognitive Walkthrough**: Task-oriented walkthrough based on formal cognitive model of user behaviour.

## Usability Heuristics (Nielsen’s 10 Fucking Heuristics. Kill me now)

* Visibility of system status
* Match between system and the real world
* User control and freedom
* Consistency and standards
* Error prevention
* Recognition rather than recall
* Flexibility and efficiency of use
* Aesthetic and minimalist design
* Help users recognize, diagnose, and recover from errors
* Help and documentation

**Performing an Heuristic Evaluation**

* Use prototype
* Evaluator works alone
* Notes taken
* 2 passes: general flow, specific
* Group debriefing session

## Pros and Cons of Heuristic Evaluation

Pros:

* cheap;
* intuitive;
* usable early in development process;
* finds many problems;
* finds both major and minor problems.

Cons:

* may miss domain-specific problems;
* difficult to spot missing elements.

### Cognitive Walkthrough

* Task orientated walkthrough of interface
* Tries to identify how well interface supports novice users
* Evaluators / participants try top spot areas or steps where there may be difficulty for the user

**Preparation**

* Identify user population
* Define suite of representative tasks
* Create interface / prototype
* Action sequence for each task (list of steps)

**Step**

1. Will the user be trying to achieve the right effect? - Will they want to select this action?
2. Will the user know the correct action available? - Is the control for the action available?
3. Will the user know that the correct action will achieve the desired effect?
4. If correct action is taken, will the user see that things are OK? - Is feedback provided?

**Pros and Cons of Cognitive Walkthrough**

Pros:

* finds task-oriented problems;
* helps define users' goals and assumptions;
* usable early in development process.

Cons:

* time-consuming;
* some training required;
* needs task definition methodology;
* applies only to ease of learning problems.

### **Notes on Holding a Cognitive Walkthrough Meeting**

* Do not explain the prototype
* Stick to well-defined tasks
* Take notes
* Audience questions are just problems to note
* Don’t try to defend the prototype
* The quality of the prototype matters
* Importance of the prepared tasks
* Domain training may be needed
* Keeping the audience sweet
* CWT is not a training session
* Keeping to the task steps
* It is psychologically very hard to do this

## [Nielsen's 10 heuristics](http://mcom.cit.ie/staff/Computing/prothwell/HCI/SOFT6002/notes/HA.html)

1. [**Visibility of system status**](http://mcom.cit.ie/staff/Computing/prothwell/hci/soft6002/notes/HA.html#_Toc224716156)**:** The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.
2. **Match between system and 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, making information appear in a natural and logical 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.
5. **Error prevention:** Even better than good error messages is a careful design which prevents a problem from occurring in the first place
6. [**Recognition rather than recall:** Minimize the user's memory load by making objects, actions, and options visible.](http://mcom.cit.ie/staff/Computing/prothwell/hci/soft6002/notes/HA.html#_Toc224716161)
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. **Aesthetic and minimalist design:** Dialogues should not contain information which is irrelevant or rarely needed
9. [**Help users recognise, diagnose, and recover from error:**](http://mcom.cit.ie/staff/Computing/prothwell/hci/soft6002/notes/HA.html#_Toc224716164) [Error messages](https://www.nngroup.com/articles/error-message-guidelines/) should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
10. **Help and Documentation:** Even though it is better if the system can be used without documentation, it may be necessary to provide help and [documentation.](http://mcom.cit.ie/staff/Computing/prothwell/hci/soft6002/notes/HA.html#_Toc224716165)

## [Other Heuristics and Guidelines](http://mcom.cit.ie/staff/Computing/prothwell/HCI/examples/Guidelines.html)

### Shneiderman's 8 Golden Rules of Interface Design

1. Strive for consistency
2. Cater to universal usability
3. Offer informative feedback
4. Design dialogs to yield closure
5. Prevent errors
6. Permit easy reversal of actions
7. Support internal locus of control
8. Reduce short term memory load

## [Links between Heuristics and Theory notes](http://mcom.cit.ie/staff/Computing/prothwell/HCI/SOFT6002/notes/HE-Theory%20Links.html)

* Visibility of system status: Appropriate feedback
* Match between system and the real world
* User control and freedom
* Consistency and standards
* Error prevention
* Recognition rather than recall
* Aesthetic and minimalist design
* Help users recognize, diagnose, and recover from errors

## 

# Section 4: Technological Factors/GUI/Task Analysis

# **Technological Factors**

## **Interaction Styles**

#### Command-response

Advantages: Fast, Suits trained specialist user.

Disadvantages: Cryptic, Hard to remember, Difficult for new or non-specialist user.

#### Form-filling

Advantages: Familiar, Suits new/infrequent user.

Disadvantages: Constrained - designed for particular tasks. Frustrating for frequent user.

#### Question-answer sequences

Advantages: Familiar, Suits new/infrequent user.

Disadvantages: Constrained - designed for particular tasks. Frustrating for frequent user.

##### Menu

Pop-up: Appear when user initiates some action e.g. right click in windows.

Pull-down: Menu is selected from menu bar.

Menu Issues:

1. Need for self explanatory option names.
2. Need to be organised (often hierarchical); how do you choose what goes at each level?
3. How do you order options on a menu?
   1. Alphabetically.
   2. By category: e.g. a ‘file menu’ includes groups of operations: opening and closing, saving, information display, and printing. What are sensible categories?
   3. Conventional: organised according to conventions e.g. days of week.
   4. Frequency of use: good for small number of options – otherwise avoid as it decreases stability.
4. Prone to accidental mis-selection => need forgiveness.

**Direct Manipulation Attempts** to imitate real-world cause/effect relationships and so is crucial to mental model building.

Examples:

* drag & drop
* Button pushing
* Checkbox ticking

Easy to learn, remember.

Rapid feedback (should be under 0.1 second response)

Not all tasks can be described as direct manipulation of objects

3 phases: **Free** (object can be manipulated), **Captive** (start of manipulation), **Termination** (show results of manipulation)

**Visual Feedback** Rich visual interaction is the key.

**Limits on Response Times**

* 0.1 sec.: is the limit so that the system appears to react instantaneously. Important for direct manipulation, virtual world navigation.
* 1 sec.: is the limit so that the user's flow of thought stays uninterrupted. Display a busy cursor if things will take longer than 1 sec.
* 10 secs.: is the limit for keeping the user's attention on the task at hand. Display a progress indicator if things will take longer than 10 secs.

## ***GUI Widgets***

**Dialogues** Used for secondary interaction and as a way of gathering relevant information together. Also serve a teaching purpose.

**Dialogues**: Initiated by user in order to complete some task – usually involve elaborate interface. They must actually be a two-way interaction.

***Message boxes:*** Initiated by system to provide information, error messages, warnings, etc.

**Types:**

Modal: This type will disallow any other activity until dealt with. They request some decision and/or information.

Modeless: These can be ignored while continuing with other work. They request some information and offer some actions.

### **System errors**

They should contain information about the effect of the error from the user’s point of view, more extensive information, escape routes, and a possible course of action. Don’t blame the user.

### **Task Analysis**

An entire software product must be designed with the aid of systems modelling and design techniques. A part of the design is the design of the user interface. Here the designer must focus on the user and the tasks they will have to perform.

The main focus of the analysis is on the information that is being exchanged between the user and the software. Questions to be addressed here include:

1. What information is required by the software to complete the task?
2. In what order is that information required?
3. What information should be shown to the user?
4. When should the information be available to the user?
5. How can the software ensure that the information it gets from the user is accurate? This means asking the following types of questions:
   1. Will the user know the information?
   2. Could there be any confusion on the user’s part about the information required?
   3. How can the software guard against mistakes being made in entering the information?
6. How can the software ensure that the information it gives to the user is useful and understood? This means asking the following types of questions:
   1. Will the user understand the information in the way it is presented (language, tables, background technical/domain knowledge, etc.)?
   2. Could there be any confusion on the user’s part about the information shown?
   3. How can the software guard against mistakes being made in noticing and interpreting the information?
   4. Is it reasonable that the software will have access to this information?

### **Steps Involved**

The kinds of steps involved in producing a task analysis are:

1. study the activity;
2. describe it;
3. identify the tasks - a task is something like: paying for items on a website, searching some collection of data, logging in, deleting some record in a database, etc.;
4. analyse each task by breaking it down into its subtasks – a subtask might be ‘supply credit card number’, ‘select record for deletion’, etc.
5. analyse each of the subtasks in terms of information the user must supply – ask yourself questions about it like those detailed above. Record decisions made here and reasons for decisions;
6. analyse each subtask in terms of information the software must supply – ask yourself questions about it like those detailed above. Record decisions made here and reasons for decisions;
7. model the task as a hierarchical diagram;
8. identify the sequence of events;
9. walkthrough the model, that is, check it;
10. enhance the model if necessary.

Example Atm bank

# Section 5: ISO usability Standards (Can be ignored according to Rokis)

Dialogue Principles

Guidance on Usability

Presentation of Information

User Guidance

### **Components of Usability**

The components of usability are:

1. the product itself;
2. the Context of Use (CoU);
3. the goals of the user;
4. the usability metrics.

### Usability Metrics

**Effectiveness**

Effectiveness measures the accuracy and completeness with which users achieve specified goals.

**Efficiency**

Efficiency is measured by relating the effectiveness level and the resources used. Resource can be: time, mental or physical effort, money, materials, etc.

**Satisfaction**

This is defined as freedom from discomfort and positive attitudes to the use of the product.

### **Presentation of Information**

1. Clarity
2. Discriminability
3. Conciseness
4. Consistency
5. Detectability
6. Legibility
7. Comprehensibility

### **ISO 9241 - Parts 10-17**

**10) Dialogue Principles**

* 7 general principles of good design for any UI dialogue (interaction) between user and system.
* Basis for parts 12-17.

**11) Guidance of Usability**

* Framework for applying standards, outlining notion of Context of Use (CoU).
* Guides usability measurement & testing.

**12) Presentation of Information**

* General principles on how info should be presented visually.
* Basis for style guide rules.

**13) User Guidance**

* Any info provided to the user which helps them successfully understand the software so they can complete task.

**14-17**

* Menu dialogues
* Command dialogues
* Direct Manipulation Dialogues
* Form-Filling dialogues
* Provide recommendations on different interaction styles

# 

# Section 6: Introducing Usability into an Existing Software Development Process

This section describes five phases in the introduction of usability engineering into in a Large Software Development Organisation.

These phases are:

1. **Raising Awareness**

* Key factor in introducing usability into organization.
* Demonstrates importance & relevance of usability.
* Show how usability issues affect quality & user satisfaction
  1. Discuss usability issues
     1. Regular meetings
     2. Discuss known problems
     3. Ideas on enhancements
  2. Get usability contributions recognised
     1. Point out when problems are related to usability
  3. Measure usability
     1. Benchmark on the level of usability of your project
     2. Industry standard questionnaire
  4. Observe usability testing sessions
     1. Observe usability testing first hand to see the problems the user experience

1. **Increasing Knowledge of End Users**

* The more understanding you have of the user, the better you are able to design the software for their needs.
  1. Make use of customer visits
     1. Look at user's environment
     2. Trip report
     3. Rough sketches
  2. Browse through customer complaints
     1. Highlight usability issues that customers raise
  3. Review customer data
     1. Debug info - look at the customer setup can highlight further problems.
     2. Try to understand their system.
     3. Don’t just use it to find bugs.

1. **Focus on User Interface Design**

* Work on designing the software for the user.
  1. Write user view documents
     1. User interface specification - user view
     2. Interface can be reviewed internally at any stage.
     3. Picture of screen + textual description

**Preliminary Prototyping**

* Prototyping to support developer during the UI design phase.
* Suggest ideas at beginning.
* As design matures, get more specific.
* Rough, hand drawn sketches.

**Prototyping during Implementation Phase**

* Prototype using tools like Visual Basic.
* Later stages of development.

1. **Evaluate Design Independently**

* If you don’t use the software like the user does, you won’t know if the software is designed correctly.
  1. Do heuristic inspections
     1. Use task list - list of tasks you expect real users to perform.
     2. Perform each task and look for deviations from usability heuristics.
     3. Prioritize issues + propose solutions
  2. Ad hoc usability testing
     1. More costly than heuristic inspections - More preparation + requires a tester.
     2. Give tester a task and ask them to complete it using software.
     3. Think aloud as they work through task.
     4. Recorded and reviewed later.
     5. Any issues found are sent to devs for heuristic inspections.
  3. Get test group to test for usability
     1. Include usability testing and heuristic inspections.
     2. ?????

1. **Control The User Interface Design Centrally**
   1. Review all user views
      1. Get to know all of the UI so that you can apply similar design patterns to all areas.
      2. UI should be as familiar and consistent as possible
   2. Write a menus document
      1. List and describe all menus
      2. Useful for maintaining consistency
      3. Keep it as short and as simple as possible - no pictures
   3. Use standard user interface design guidelines
      1. Java
      2. Windows
   4. Write some additional guidelines
      1. Blah blah blah
      2. Spoof it

# 

# Papers ( Full )

### Affordances – McGrenere & Ho (Affordances)

D. Norman: “an affordance is the design aspect of an object which suggests how the object should be used”

An **affordance is ‘subjective’** because it depends on the subject’s ‘frame of reference’ .

Example, a hammer affords a strong grip for a human because of the nature of its hand – it does not afford a strong grip for a cat.

An **affordance is ‘objective’** because it does not depend on the actor’s ability to perceive it. Even if I do not recognise the affordance of some widget that does not mean that the affordance does not exist.

#### **Learning affordances**

An Affordance is there even if it is unperceived but the ability to perceive it depends on experience.

#### **Direct, sub-conscious access to affordance**

Direct perception of an affordance means we subconsciously know how to use some widget because our attention is educated, we can learn to perceive it in the same sub-consciously skilled way we can learn how to balance on a bicycle or change gears in a car.

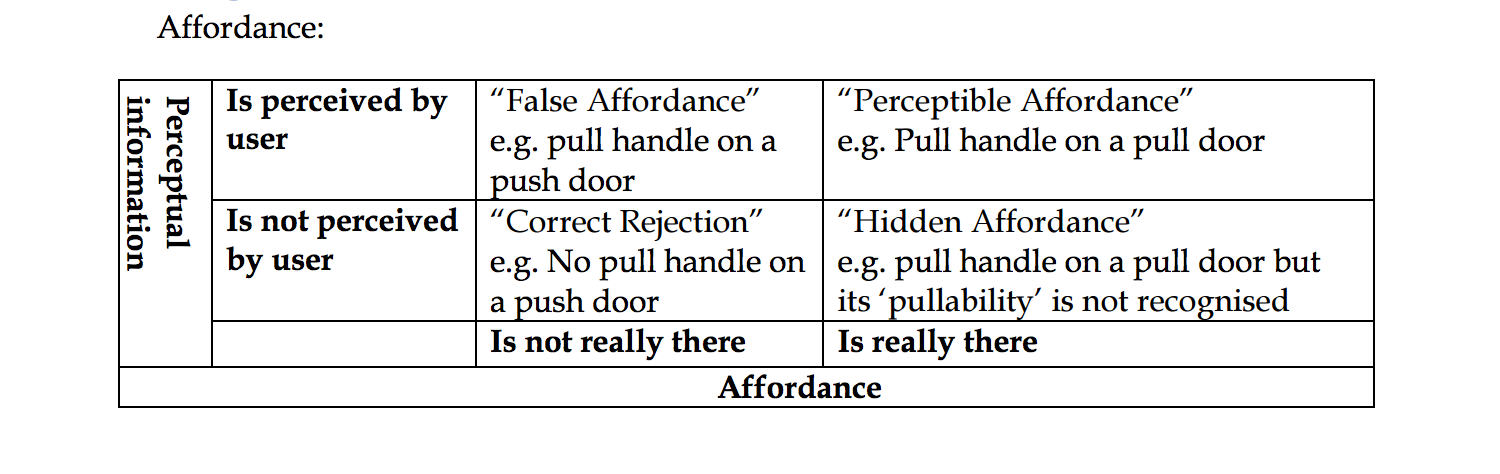
#### **Definitions**

1. Usability - making the affordance apparent and making it easy to recognise and use.
2. Utility - making the widget supply the affordance.
3. Functionality - making the software deliver the function liable to be associated with the affordance the user perceives.

So the usefulness (utility & functionality) of a widget must be programmed so that:

* The user is allowed to do tasks efficiently in order to reach their goals and
* The affordances match the user's goals.

And the usability must be programmed through graphics that allow the user direct access through learned attention – the visual design must make the affordance apparent. This is done through aspects of design including graphics, feedback, consistency, match between system and real world, error recovery, etc.)



### Three Introductory Papers

**HCI Intro**

* Need for HCI due to system complexity, productivity demands and less time for training.
* Integrate into existing SE practice
* HCI contributes to:
  + Analysis
  + Design
  + Implementation
  + Evaluation

**Susan Dray**

* For users, the UI IS the system
* Match of tasks to functionality
* Good UI
  + Improved efficiency
  + Reduced training time

**Alex Dix**

* Human error: Disaster is inherent in the design of the interface.
* UI design involves trade-offs between various goals.
* User should be involved in design
* User should be involved in evaluation
* Interfaces are difficult to program - difficult to change bad design - use throwaway designs
* Interface design is: an attempt to articulate a real world through a virtual world

### Hollendar - CLT

* CLT wants to limit load on STM as it deteriorates learning.
* ICL
  + Intrinsic complexity of material to be learned / dealt with
  + Size depends on the amount of interactivity
* ECL
  + Irrelevant material
* GCL
  + Caused by active schema creating - good for learning
* Goal of HCI is to decrease CL - through ease of learning and use
* Designing usable systems means understanding the users and tasks
* L.E.M.E.S
  + Learnability
  + Efficiency
  + Memorability
  + Error Rate
  + Satisfaction

### Erickson - Metaphors

* Metaphors allow us to take knowledge of the familiar and use it to give structure to more abstract concepts.
* Incorrect metaphors can cause difficulties - early breakdown of metaphor
* Poor metaphors - Often occur due to metaphor maker not fully understanding the system logic.
* Hard to think of good metaphors

**Inventing Interface Metaphors**

1. Understand how system works -
2. What aspects of the system provide difficulties for users -
3. Generate appropriate metaphors.

**Generating Metaphors**

1. Find ideas implicit in the problem description.
2. Use these to identify real world objects, structures that have the same sort of functionality.
3. Evaluate these based on the following criteria:
   1. Amount of Structure: The more structure the better
   2. Applicability of Structure: Which parts of metaphor might lead user astray?
   3. Representability: Represent metaphor visually, orally or with specific words.
   4. Suitability for Audience: Will the users understand metaphor?
   5. Extensibility: Can the metaphor be expanded upon?

### Kujala - User Involvement

**Benefits of User Involvement**

* Increased product quality due to increased requirements accuracy.
* Avoidance of unnecessary features - Scope Creep.
* Improved user acceptance of the software.
* Increased understanding of software -> Increased effectiveness.
* Increased participation in decision making.

**Negative Aspects**

* Amount of raw data, time spent, difficulties in analyzing.
* Acceptance of method among developers and management.
* Access to users is difficult.
* Increased requests for requirement changes late in development.
* Users lack of knowledge of design process.
* Users difficulty in articulating expert knowledge.

**Access to Users**

Gaining access to users is difficult for a number of reasons:

* Difficult to find users willing to let you watch them work. -> Takes longer to find users.
* Difficult to get developes out of office and visit users.
* Difficult to arrange days to visit users when they had real work to do.
* Identifying appropriate users.
* Getting access and motivating users
* Users were busy
* Users lack confidence or motivation and were reluctant to talk to developers