1. Requirements elicitation and task analysis

1.1. HCI and software development

1. The software engineering life cycle

The waterfall model:
Requirements elicitation - specification - design - coding - testing - maintenance
User interface design is NOT represented as a core activity.
(Interface design has some role in requirements elicitation and testing).

2. The HCI life cycle

Because the traditional model doesn’t clearly identify a role for HCI at any point, user interface concerns are "mixed in" with wider development activities.
Problems: HCI is ignored / relegated to the later stages as an afterthought.
The later one postpones HCI, the higher the cost of introducing usability issues.

HCI development model:
Initial design - formative evaluation - summative evaluation
Interface design drives the whole process.
By spotting user requirements early, there will be less demand for code modification.

1.2. Mechanisms for HCI in software development

The ISO9000 standard sets out approved procedures for software development.
Design rules = rules a designer can follow in order to increase the usability of the product.

Rules are classified according to their authority and generality.
Authority = an indication of whether or not the rule must be followed or if it’s only a suggestion
Generality = an indication of the extent to which a rule can be applied.
Guidelines & standards are two kinds of design rules.
Standards: High in authority and limited in application.
Guidelines: Lower in authority and more general in application.

1. Design guidelines

Guidelines help provide a framework that guides designers towards making sound decisions.
Guidelines = lists of rules about when and where to do things / not to do things.
E.g. 'Do not have more than 10 items in a menu'.

Problems:
* You need a large number of rules in order to cover all possible problems that may occur
* It is difficult to know what to do when you have to transgress a guideline

Design rules = instructions that can be obeyed with minimal interpretation by the designer.
E.g. 'Date fields should be in the form DD-MM-YY'.

Directing principles = the best guidelines because they’re high-level and widely applicable.
E.g. ‘Maintain consistency and clarity’.
They must be interpreted into unambiguous rules appropriate for the particular system.

a. Low-level detailed rules

The Smith and Mosier rule guidelines list dos and don’ts for interface design.
Companies develop their own guidelines, with commercial motivations, not just usability.
E.g. Apple’s guidelines help you produce a system that looks like other Apple products.
E.g. Following a step-by-step recipe is an example of the low-level detailed approach.

b. High-level directing principles

Principles focus on problems common to many different systems.
Principles are more abstract than rules - they don’t explicitly mention what the system is.
principles impose fewer constraints than design rules do.
Problem: Principles don’t help with the details of interface development.

Dix’s principles to support usability:
1. Learnability
   = The ease with which new users can begin effective interaction.
   Predictability: The user can determine the effect of future action based on past history.
   (Operation visibility: The availability of the next operation is shown to the user).
   Synthesisability: The user can assess the effect of past operations on the current state.
   (Honesty: The interface tells the user how the operation changed the internal system state).
   Familiarity: The user’s real-world / computer knowledge can be applied to the new system.
   (Guessability, affordance: The object’s appearance is familiar with its behaviour).
   Generalisability: The user can extend knowledge of specific interaction across applications.
   Consistency: Similar situations have similar input-output behaviour.
   Maps use a set of generally accepted markings and notations. This demonstrates the principle of consistency.

2. Flexibility
   = The ways in which the user and system exchange information.
   Dialogue initiative: The user is free from artificial input constraints imposed by the system.
   (System pre-emptiveness: The system initiates all dialogue. The user responds to requests).
   (User pre-emptiveness: The user is entirely free to initiate any action towards the system).
   Multi-threading: The system can support more than one task at a time.
   (Concurrent multi-threading: Simultaneous support for separate tasks).
   (Interleaved multi-threading: Temporal overlap between separate tasks).
   (Multi-modality: Separate communication channels form a single input / output expression).
   Task migratability: The control of execution of a task can be passed on.
   Substitutivity: Equivalent values of input & output can be arbitrarily substituted for each other.
   (Representation multiplicity: State information is rendered in different formats / modes).
   (Equal opportunity: The user has the choice of what is input and what is output).
   Customisability: The user interface can be modified by the user or the system.
   (Adaptability: User-initiated modification to adjust the form of input & output).
   (Adaptivity: System-initiated modification to customise the user interface automatically).

3. Robustness
   = The level of support the user is given.
   Observability: Users can observe the effects of their input from the display (E.g. your PIN)
   (Browsability: The user can explore the internal system state via the limited interface view).
   (Static defaults: Defined within the system / acquired at initialisation).
   (Dynamic defaults: Evolve during the interactive session).
   (Reachability: The user can navigate through the observable system).
   (Persistence: The duration of the effect of a communication).
   Recovarability: The user can take corrective action once an error has been recognised.
   (Backward recovery: Undo the previous interaction to return to a prior state).
   (Forward recovery: Accept the current state and negotiate from that state to the desired one).
   (Commensurate effort: The effort of undoing an effect is proportionate to the effort of doing it).
   Responsiveness: How the user perceives the rate of communication with the system.
   Task conformance: The system should support all tasks the user wants to perform.
   (Task completeness: The coverage of all tasks of interest).
   (Task adequacy: The user’s understanding of the tasks).

Schneiderman’s principles for user-centred design:
1. Recognise diversity
   * Usage profiles:
     Understand profiles (age, gender, education, culture, motivation, goals, personality…)
     Approaches to accommodating different usage classes:
     → Permit a level-structured approach to learning. (Novices are taught the minimum and progress to higher levels; Experts make rapid progress).
     → Permit user control of the density of feedback. (Novices want more informative feedback; Experts want less distracting feedback).
     * Task profiles:
Perform a formal task analysis.
Task frequencies are important for shaping a menu tree.
High-level actions \(\rightarrow\) middle-level actions \(\rightarrow\) atomic actions.
* Interaction styles:
Choose from: command / natural language, form fill-in, menu selection, direct manipulation.
**Dialogue style, forms and menus are examples of interaction styles.**

2. **Use the eight golden rules of interface design**
   
i) **Consistency** (in terminology, help screens, colour, layout, fonts…)
   ii) **Feedback** (the response should be more substantial for infrequent / major actions)
   iii) **Easy reversal of actions** (encourages exploration of unfamiliar options)
   iv) **Error prevention & handling** (design so that users can’t make serious errors)
   v) **Support internal locus of control** (let users feel they’re in charge of the system)
   vi) **Shortcuts for frequent users** (abbreviations, special keys, hidden commands, macros…)
   vii) **Dialogues that yield closure** (give users a sense of relief so they can carry on…)
   viii) **Reduce short-term memory load** (displays should be kept simple)

3. **Prevent errors**
   * Correct matching pairs (When you type ‘(‘ the system says that ’)’ is outstanding)
   * Complete sequences (Use stored info so the user can trigger the sequence with one action)
   * Correct commands (Display complete alternatives as the user starts typing)

**c. Limitations of guidelines**
Guidelines help you identify good and bad options for your interface.
They restrict the range of techniques you can use (and still conform to a certain style).
They are hard to apply because you can’t just use rules - user characteristics are NB too!

2. **Standards**
Standards concern prescribed ways of doing things to achieve consistency across products.
Standardisation in interface design would provide for:
* A common terminology (so you can compare different systems)
* Maintainability & evolvability (so that additional facilities can be added)
* A common identity (if they all have the same ‘look & feel’ they’re easily recognisable)
* Reduction in training (standard command keys make it easy to transfer knowledge)
* Health & safety (users won’t be stressed / surprised by unexpected system behaviour)

Standards for interactive system design are set by national / international bodies (like ISO / BSI / SABS) to ensure compliance with a set of design rules by a large community.

<table>
<thead>
<tr>
<th>Underlying theory</th>
<th>Standards are based on an understanding of physiology / ergonomics.</th>
<th>Standards are based on theories from psychology / cognitive science (= vaguely worded).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change</td>
<td>More difficult / costly to change, so requirement changes don’t occur that frequently.</td>
<td>Easier to change, because software is designed to be very flexible.</td>
</tr>
</tbody>
</table>

Since standards are relatively stable, they are more suitable for hardware than software.
Standards institutions therefore have hardware standards in place prior to any for software.

The international draft standard ISO9241, entitled Ergonomic requirements includes:
**Hardware issues:**
* Requirements for visual display
* Keyboard layout
* Workstation layout
* Environment
* Display with reflections
* Display colours
* Non-keyboard input devices
**Software issues:**
* Presentation of information
* User guidance
* General dialogue principles
* Command dialogues
* Form-filling dialogues
* Menu dialogues
* Direct manipulation dialogues

**ISO definition of usability:**
Usability = the effectiveness, efficiency and satisfaction with which users achieve goals.

**3 components of usability:**
Effectiveness = accuracy and completeness with which users achieve specified goals.
Efficiency = resources expended in relation to accuracy & completeness of goals achieved.
Satisfaction = the comfort & acceptability of the work system to its users.

The strength of a standard lies in its ability to force large communities to abide by it. The authority of a standard can only be determined from its use in practice. Some software products become standards long before any formal standards are published.

1.3. HCI and requirements elicitation

1. Gathering information
Requirements elicitation = techniques used to establish objectives for an interface. Don’t lose support of your users by asking questions that are ill-informed. Focus groups and questionnaires can be used to gather initial evidence. After becoming familiar with the domain, designers can use direct techniques (interviews…)

2. Requirements elicitation techniques
Results of requirements gathering:
* A representation of the problems of the current system
* A representation of the requirements of the new system
3 categories of system requirements:
* Functional requirements (specify what the system must do)
* Data requirements (focus on structure as opposed to processing - UML)
* Usability requirements (specify the acceptable level of user performance)

To determine usability requirements there are 3 main types of analysis:
--- Task analysis (to determine task structure, ease of learning… of the users)
--- User analysis (to determine the scope of the user population)
--- Environment analysis (to determine aspects of the physical & user support environments)

Document analysis & questionnaires don't involve personal contact between designer & user

a. Interviews
Advantages:
* Low cost (Few resources are needed to set them up)
* Flexible (They can be structured or unstructured)
* Participatory (They encourage user participation during interface development)

<table>
<thead>
<tr>
<th>INTERVIEWS</th>
<th>Structured interview</th>
<th>Semi-structured interview</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ease of interviewing</td>
<td>Easier</td>
<td>Harder</td>
</tr>
<tr>
<td>Scope for picking up other issues</td>
<td>Less</td>
<td>More</td>
</tr>
</tbody>
</table>

Semi-structured interviews are good for determining users’ understanding of interfaces. A checklist of questions can be formulated to guide the investigation of users’ tasks.

Things to consider when preparing for an interview:
* Purpose (What information do you expect to obtain?)
* Target group (Establish criteria for selecting individuals / groups)
* Nature of the interview (Decide if the interview is to be formal / relaxed)
* Recording & analysis of the interview (Video and audio tapes are commonly used)

b. Rankine charts
Get a team of users to discuss the existing system and observe their discussion. Problem: The most dominant people get their opinions accepted. Rankine charts address this problem.

*The lines give you an idea of everyone’s contribution, so you can use one-to-one interviews afterwards to make sure everyone’s views are represented.*

Advantage: You don’t need to speak until you have an idea of the views of the users.
Disadvantage: It can be difficult to control large meetings and get people to talk freely.
c. Questionnaires
They are more cost effective if you need focussed answers to specific questions. You can also access views of a large and geographically distributed user population.

Disadvantages:
* You don’t know how much effort the users put into the questions
* High failure rate from unreturned forms sent to key individuals
You should spend a lot of time preparing, because questionnaires can’t be recalled.

Considerations for questionnaires:
* Speak the users’ language
  Avoid terms that the subjects won’t understand - Perform a trial run.
* Decide on anonymous / open surveys
  Sometimes you must declare anonymity and state the intended use of the questionnaire.
* Select appropriate questions (see p 28 of the study guide!)
Open questions: Flexible, and can be used to expand on short answer questions.
Closed questions: Usually have a rating scale, and are easy to analyse.

E.g. If you want to find out how long clerks currently take to access the stock in hand.

Simple checklist:
Ranked questionnaire: Difficult to analyse, and inconclusive results.
Multipoint rating scale:
Lickert scale: A multipoint rating scale where there are degrees of agreement in between.
Semantic differential: Bi-polar adjectives at the ends of the scale.

d. Analysing data for questionnaires and interviews
The responses obtained must be converted into numerical values for statistical analysis.

Discuss the Hawthorne effect.

Hawthorne effect: People behave differently if watched, so you can get false results.
These requirements elicitation techniques provide qualitative rather than empirical results.
If you communicate the results to the people they can help interpret surprising results.

3. Specification and initial design
a. Questions, options and criteria (QOC)
QOC diagrams help to structure development decisions (about style, format…)
They are valuable for communicating the justification behind design decisions.

Questions: Represent the agreed characterisation of the issues raised.
Options: Options that provide alternative solutions - box the most favourable one.
Criteria: Determine the most favourable option - favourable = solid line, negative = dashed.
Key to effective QOC: Decide on the right questions and correct criteria to judge the options.
Initial questions must be general, but specific enough to clearly identify a range of options.
Use general criteria too, like guidelines and principles.

Advantages of QOC:
* It can be used to represent the many different forms of decisions that have to be made.
* Designers are not forced to learn complex notations.
* Diagrams help represent options explicitly.
* Diagrams can be used in conjunction with interviews, where users can provide more info.
* Users are made aware of the various constraints that affect development.

1.4. Task analysis

Task analysis = analysing the way people perform their jobs.
Main tools used: observation, interviews, documentation.

One purpose of task analysis: To help produce training manuals & documentation.
Task analysis in itself is not a form of requirements capture as it refers to the existing system, not the planned system. It does, however, make a contribution to the requirements statement.
How task analysis of an existing system can assist:
* The analyst can ask which existing objects / tasks should be in the new system
* Presenting the existing state can help the client decide on new features
* Predictions can be made about how the new system will interact with existing procedures
Task analysis techniques:
* HTA (Hierarchical task analysis) - identifies tasks and subtasks
  - It focuses on what actually happens and not on what should happen
* CTA (Cognitive task analysis) - describes aspects of cognitive characteristics of users’ tasks
  - It focuses on representing the (procedural / task) knowledge that the user has
2. Interaction styles

2.1. How to select a dialogue style

Characteristics to consider:
- **The users’ tasks**
  - Text-based interfaces are good for high-volume data entry tasks.
  - Graphical presentation facilities are good for complex data analysis, like spreadsheets.
- **The users’ characteristics**
  - Menu & graphical based user interfaces are better suited to novice users.
  - Textual interfaces are better suited to expert users.
  - If you pitch your interface at a novice’s level, be prepared for growth of an expert population.
  - If the flow of people through an office is high, you’ll still have to support a novice population.
- **The system’s characteristics**
  - The system facilities provided by a customer can constrain your choice of dialogue style.
- **The available input & output devices**
  - You need speakers & a microphone for audio interaction.
  - You need a high-resolution display for graphical interaction.

2.2. Text based interfaces

1. **Design aims for command languages**
   - *Consistency* (The meaning of one command should be the same in each context)
   - *Compactness* (The set of different symbols should be as small as possible)
   - *Ease of reading & writing*
   - *Speed in learning* (The shorter the command, the longer it will take to learn)
   - *Low error frequencies* (If there are high error rates you may lose speed improvements)

2. **Programming textual interfaces**

Name two advantages of text-based interfaces.

One of the biggest advantages of text-based command languages is their scripting facility.

Icons may be good for representing folders, but are less easy for representing processes.

3. **Forming command languages**

Set conventions for combining individual commands:
- **Simple commands:**
  - In simple systems you can associate a unique name with each action.
  - E.g. Each command can be associated with a single letter.
- **Simple command plus object:**
  - An operation is followed by one or more objects that are affected by that operation.
  - E.g. PRINT FILE1
- **Command plus option and object:**
  - Special options can indicate subtle changes in the meaning of the operation.
  - E.g. PRINT -C3 FILE1 could produce 3 copies of a file.
  - Advantage: Novices can use the simple commands and experts can learn advanced options.
  - Disadvantage: High number of errors for infrequently used options.
- **Hierarchical commands:**
  - Users type a command and are then presented with a list of further options.
  - E.g. CREATE could lead you to be prompted for the filename etc.
  - Disadvantage: Traversing the hierarchy structure can slow you down.

4. **Consistency**

Find out what systems your users have been exposed to so they can transfer their skills.

Internal inconsistencies e.g: ‘RETURN to continue’ and ‘ENTER to exit’.

The formation of the language should also be consistent:
- *Command names followed by objects e.g: Display File (used in command languages)*
- *Objects followed by command names e.g: File | Display (used in graphical interfaces)*

5. **Naming rules**

For a command to be easy to read, it must bear some relationship to the action performed.
For a command to be easy to write, users must be able to read it and understand it. Conventions to shorten command names:
* Fixed truncation - shortens a command to a fixed number of letters: DEL, INS, COP, MOV
* Minimum truncation - shortens a command to a variable number of letters: INS, MOVE
* Vowel drop - removes all vowels except leading letters: DLT, INSRT, CPY, MV
* Phonetics - makes abbreviations sound like the command: X (to exit), DLT
* User aliasing - allows users to associate names with commands: merge t-prev to merge all the day’s transactions with those of the month to date.

6. Textual output
Graphical interfaces also have textual information. Textual information must be presented in terms that the user will recognise.

7. Design aims for textual displays
* Consistency of layout: If you re-position important text it will lead to distraction & error.
* Minimal memory load on user: Users shouldn’t have to remember information of another screen. Tasks should be arranged so that completion occurs as soon as possible.
* Input compatibility: Users shouldn’t be expected to perform complex manipulations between the information presented and the commands that they have to enter into the system.
* Low error rates: If you use bizarre colours and odd fonts, it will lead to higher error rates.

8. Font selection
Typography = the style & appearance of printed matter.
Serif fonts - e.g. Times New Roman, include curls etc (good for continuous text).
Sans serif fonts - e.g. Arial, appear plain and stark (good for headings & diagrams).
Fonts are stored in one of two ways:
* Stored as bitmap images
  Advantage: Relatively little computation is involved in translating a document into bitmaps.
  Disadvantage: Quality is lost as the bitmaps are enlarged to different sizes. (Staircase effect)
* Stored as a number of mathematical functions
  Advantage: Quality is preserved as the scale of the character is increased.
  Disadvantage: If you use an obscure font, the user might not have the required functions.

9. Point size
One point = 1/72 of an inch.

10. Tasteful design and layout
Changes in styles indicate additional information, such as emphasis.
Avoid using more than 4 different point sizes & more than 3 fonts in one document.
Some issues to consider:
* Logical sequencing: Users should meet items of text in the correct order of processing.
* Grouping: Similar items of information should be grouped to avoid long visual searches.
* Spaciousness: The denser the display, the harder to access information.
* Simplicity: If displays are cluttered & too bright, users will think the system is complex.

2.3. Forms
When you make a hotel reservation online, you enter your details using a form-filling interface.
A Web-based online questionnaire is an example of a form-based interface.
Advantages:
* Provide a structure for text-based interfaces, so you don’t have to guess input sequences
* Unconstrained values - you are free to choose the text to enter in a field
* Possible to check for errors (E.g. numeric data in a name field)
* Related queries: Interaction progresses by gradually filling in the required information
* Order independence: Users can first fill in the values that they are sure about.
Disadvantages:
* Keyboard dependent
* Relatively few data values: If there are only a few data values, menus are better
* Ordered data entry: If users must enter data in a certain order, use command-line prompts
1. Design aims for form filling
   * Easy navigation: It must be easy to move between forms & fields (use tab / arrow keys)
   * Consistent labelling: Stick to one label throughout the system
   * Clear format: Put clear spaces between fields, and make it clear which areas are for input
   * Easy correction: Error messages should state as much information about the problem as possible, and provide overtyping facilities to correct input, giving a completion signal when the data has been successfully processed.

2. Appearance and composition
   1. Field panel
      Prompts the user and indicates the appropriate input formats.
      If space is limited, provide help facilities to reiterate the on-screen prompts.
      Provide information about the maximum size of input for each field.
      Provide feedback if input has been truncated.
      Group data fields according to:
         * Task issues: Frequently edited fields can be grouped at the top to save users time
         * Common subjects: Put name & address information together, so users don’t have to search
         Within each field, use left justification to ensure input starts in the same column on the screen.
         Leave enough space between groups so users can identify boundaries between subjects.
   2. Context panel
      Provides users with info about the purpose of a particular form.
      Screens should have names so that users know what they were doing if they’re interrupted.
      If a form has been reached as part of a long dialogue, users must know how they got there.
      You can concatenate the names of the screens visited to reach that point.
   3. Control panel
      Includes information about how to progress / navigate between and within a form.
      Users should know how to leave the current screen and exit without saving anything.
      Users should know how correct the information that they have entered in particular fields.

3. Behaviour
   * Simple
      Little error checking is performed on the field information.
      Users enter in data and press a button to confirm completion.
      Advantage: Flexibility - the user is in control of the data entered.
   * Advanced
      Many fields have default values.
      Inference capabilities mean the system can use info from previous fields to automatically calculate values for other fields.
      Danger: The defaults might be incorrectly left in, with deduced values incorrect.
      Disadvantage: The system has more control over form filling, which users may not like.

4. Navigation
   Standard methods for navigating forms = Tab & arrow keys, and using a mouse.
   Forms should indicate what a user should do once he has completed it / wants to quit.
   All information required on a specific form should be available on that form.
   Help should be provided to assist with navigation, as well as with the meaning of each field.

2.4. Menus
   Menus offer the user a more constrained choice from a limited number of options.
   Name two advantages of menu-based interfaces.
   Advantage:
   Menus provide prompts that support short and long-term memory. (Users don’t have to remember commands because they can find them in the menu options).
   Strengths of menu-based interaction:
      * Contexts where users are semi-skilled: Menus introduce available commands
      * Contexts where the system constrains interaction: Greyed out items prevent errors
      * Contexts with high keystroke error rates: Quick access to commands with a mouse
      * Educational systems: Menus don’t force you to memorise rules / have a complex structure
   Weaknesses that affect menu-driven interaction:
* **Time critical tasks**: It’s difficult to navigate through several menus under time pressure
* **Poor communication links**: Limited communication bandwidth → efficiency constraints
* **Unpredictable environments**: It’s dangerous / frustrating when critical items are greyed out

1. **Design aims for menus**
   * **Clear structure**: Users must understand the relationship between menus & sub-menus
   * **Clear labelling of categories**: Menu categories must be understood and well labelled
   * **Simple selection procedure**: Frequently used commands should be easy to select
   * **Appropriate breadth & depth**: More breadth → more searching for individual items; More depth → more navigation through sub-menus.

2. **Appearance and composition**
   Similar rules as those for forms (Control panel, context, and options).
   Users must decode names to identify commands. (If finite screen space, use abbreviations)
   Individual items have to be grouped under labels, like ‘Insert’, that reflect common operations.
   Items are also grouped within a menu to support navigation.
   Destructive options, like ‘delete’ shouldn’t be placed at the top of a menu.
   Use a consistent layout, so users know where to find the title of a menu.

3. **Behaviour**
   Menus can support single selections / provide multiple check boxes.
   Context sensitive menus change over time (options can be greyed out at times).
   Menus can have scrolling mechanisms if there are many entries. (= Broad structure).
   **Some menus can be customised** (like ‘Favourites’ and ‘Bookmark’ options in IE & Netscape).
   Advantage of customisation: Maximum support for users.
   Disadvantage of customisation: Consistency is ruined, and confusion can occur.
   Designers should consider ways for users to exit from menu structures.
   Mouse-driven interfaces: Release the button outside the menu region.
   Text-based interfaces: Choose the ‘Cancel’ option.

3 common types of menu-based interaction styles: Drop-down, scrolling, and text-based menus.

2.5. **Graphical and direct manipulation interfaces**
   GUI = any interactive system that uses pictures / images to communicate information.
   Includes: Keyboard systems that only use graphics to present data, and walk-up & use systems where users only interact by selecting portions of a graphical image.
   **Direct manipulation** = interfaces with these characteristics:
   * Both the objects and actions are visible to the user
   * The actions are rapid, reversible and incremental
   * The actions are applied directly to the objects, rather than using command language syntax
   E.g. Touch screen technology to select multi-media information using a stand-alone device.
   **Direct manipulation has a narrower definition** than that for Graphical User Interfaces.
   Some CAD systems use command languages to alter the graphics (= graphical systems)
   Graphical displays can support novices, but not if they use complex command languages.
   **Direct manipulation is better for novices because actions are visible and easily accessed**.
   Some direct manipulation interfaces don’t support using graphics (like spreadsheets).
   Advantages of graphical interaction:
   * **Visibility**: Graphical displays can be used to represent complex relationships in data sets
   * **Impact & animation**: Images are more appealing than text
   * **Cross-cultural communication**: Interfaces are borrowing symbols from road signs etc...
   Disadvantages of graphical user interfaces:
   * **Cluttered displays**: Too many symbols & colours make it hard to extract information
   * **Ambiguity**: Users have to know the meaning of the image (Bridge the gulf of evaluation)
   * **Imprecision**: Some graphics can’t convey enough information without textual annotation
   * **Slow speed**: Network delays may delay the presentation of results

Name the four design aims for graphical user interfaces and describe each with one sentence.
1. **Design aims for graphics and direct manipulation**
* **Clear semantics:**
   Images don’t have to resemble the actual objects the user interacts with.
   If these abstract images are used, users will have to learn to interpret the symbols correctly.
   For infrequent users, it is better to use icons with a more direct mapping to the actual things.
* **Unambiguity:**
   Check that the symbols don’t have another meaning for the users.
   Don’t overload symbols with many different meanings.
   E.g. If ‘+’ and ‘-’ alter channels, volume, and time, depending on the mode, it’s confusing.
* **Task fit:**
   Graphical user interfaces should ‘fit’ with the overall task structure.
   Animated images can distract users from other tasks.
* **Natural interaction:**
   If the primary means of input is textual, then graphical interaction won’t be beneficial.
   If you’re dealing with shapes, directly manipulating them is better than using text commands.

2. **Appearance and composition**
   There must be a natural mapping between the images on the display and the user’s tasks.
   A broken down metaphor: Dragging a disk icon into a rubbish bin to eject it.

   Four types of commonly used icons in interactive systems:
   * Those that exactly resemble the **intended objects** (like people’s photos)
   * Those that resemble typical **objects** (like the rubbish bin)
   * Those that symbolise an **abstract (general) property** (like up & down arrows)
   * Those that represent an **arbitrary sign** (like the ‘do not tumble dry’ sign)

   Reasons why designers should avoid clutter:
   * **Perceptual loading:** It can be difficult to take in and understand many different objects
   * **Semantic loading:** The more objects, the more meanings you need to figure out
   * **Searching:** The more objects, the harder it is to find the ones you really need
   * **Manipulation:** The more objects, the smaller the size, which makes manipulation harder

   Find out from users what symbols mean to them.
   Context plays an important role in determining the meaning of icons, so you might have to develop an entire graphical interface. (Pencil & paper prototypes are cheap).
   Present alternative options so that users don’t simply agree with the first version they see.

   Dangers:
   * Users may prefer the prototype to the implementation
   * You might not be able to implement the system that you prototype
   Graphical / direct manipulation interfaces can restrict your user group.
   Blind users can use screen readers for textual interfaces, but not for direct manipulation.
   If you use colour, you may make it difficult for colour-blind users.
   Use colour sparingly to provide redundant information.

3. **Behaviour**
   Ways of interacting with graphical systems:
   * **Selecting:** Selections must be able to be passed through several layers of objects
   * **Positioning:** Note the feedback during positioning (like shadow / carried object…)
   * **Quantifying:** It can be necessary to input exact quantities, like numbers for a chart
   * **Orienting:** It must be possible to place objects in various orientations to other objects
   * **Specifying:** It must be possible to specify attributes, like creation dates…
   * **Entering text:** It is rare to have purely graphical user interfaces
   * **Stretching:** It is natural for users to graphically stretch objects
   * **Sketching:** Graphics systems appear poor compared to free-hand drawing. Curved lines are represented by bitmaps or splines (maths functions). Both make it hard to draw good curves.

3. **Evaluation**

3.1. **Why bother evaluate**
   Evaluation provides **benefits for:**
   * **Designers:** To judge the adequacy of their designs
   To provide evidence that can be used for marketing purposes
   To help convince clients that the product meets their needs
* **Clients**: To make informed decisions about the software they pay for. Evaluation tests can be set as milestones for the development process.

* **Users**: To voice their opinions and preferences, and make them feel part of development. A secondary objective of evaluation is to make users part of the development process.

Things to consider when evaluating:
* **Characteristics of the users**: Age, gender, psychological & physical characteristics...
* **Types of user activities**: From tightly specified tasks to activities decided by the users.
* **Environment of the study**: From a laboratory situation to a natural setting.
* **Nature of the artefact**: From a series of drawings to a prototype / fully developed system.

Evaluation techniques might be used to investigate:
* **The users and their tasks**:
  - Find out whether systems actually support user tasks in the manner predicted.
  - Find out if the interface provides the relevant information that the user needs.
  - Find out if the dialogue style is appropriate for the level of expertise.
  - Find out how long it will take users to learn how to operate the system.
  - Get evidence about likely errors and their associated frequencies.
  - Establish levels of user satisfaction with potential interfaces.
* **The system and interaction devices**:
  - Find out if users can successfully operate system hardware.
    (Delays in refresh rate can cause problems, so test systems on low-end machines!)
* **The working environment and supporting documentation**:
  - Determine whether the users’ environment will cause problems.
    (Changes discovered at this point will be hard to fix).

Evaluation, like software testing, gets worse the longer you leave it. Problems uncovered during evaluation can be corrected through training and documentation.

### 3.2. When to evaluate

1. **Formative and summative evaluation**
   **Formative evaluation**:
   * During design, designers need to check that their ideas are what users really want.
   * Evaluation meshes closely with design and guides the design by providing feedback.
   * **Formative evaluation helps form the decisions that must be made during development.**
   
   It’s exploratory.
   
   If formative evaluation is to guide development, it must be conducted at regular intervals.
   
   This implies that low cost techniques should be used (like pencil & paper prototypes).
   
   Formative evaluation identifies difficulties users have when starting to operate new systems.
   
   Interface design is an iterative task as designers get closer to the final delivery of the system.

   **Summative Evaluation**:
   * It takes place at the end of the design cycle.
   * It helps developers and clients to make the final judgements about the finished system.
   * It focuses on one or two major issues, instead of being exploratory.
   
   **Summative evaluation demonstrates that people can use the system in their working setting.**
   
   This involves acceptance testing (which is trivial if sufficient formative evaluation was done).

2. **Evaluation in the life cycle**
   * During the early design stages, evaluation is done to:
     * Predict the usability of a product
     * Check the design team’s understanding of user requirements
   
   Later in the design process the focus shifts to:
   * Identifying user difficulties, so the product can be fine-tuned to meet their needs
   * Improving an upgrade of the product

3.3. **How to evaluate**

The main difference between the various approaches to interface evaluation is the degree to which designers must constrain the subject’s working environment.
1. Scenario-based evaluation

Scenarios = sample traces of interaction.

Scenario-based evaluation forces designers to identify key tasks with statistical techniques in the requirements' elicitation stage.

As design progresses, these tasks form a casebook containing standard tests.

The users are shown what it would be like to complete these tests using each interface.

They are asked to comment on the proposed design in an informal way (sketches / mock-ups)

Advantages of scenarios:

- Different design options can be evaluated against a common test suite
- Users are in a good position to provide feedback about the use of the system for NB tasks
- Direct comparisons can be made between the alternative designs
- They help identify and test hypotheses early in the development cycle

Disadvantages:

- It can focus designers’ attention upon a small selection of tasks
- Some functionality may stay untested, while users become too familiar with a few examples
- It is difficult to derive empirical data from the use of scenario-based techniques

2. Experimental techniques

Experimental techniques constrain the subject's working environment more than the others.

There is an attempt to introduce the empirical techniques of scientific disciplines.

It is therefore important to identify a hypothesis / argument to be tested.

The next step is to devise an appropriate experimental model.

This will involve focusing in upon a small portion of the final interface.

Subjects will be asked to perform simple tasks that can be observed over time.

To avoid outside influences, tests will be conducted under lab conditions (no phones, faxes...)

(Laboratory-based evaluation techniques are useful in final stages of summative evaluation).

The experimenter must not directly interact with the user in case they bias the results.

The intention is to derive measurable observations that can be statistically analysed.

For this approach to be successful, it requires skills in HCI development / psychology.

Appropriate to demonstrate that measurably less errors are made with an improved system

than the old one.

Disadvantages:

- By excluding distractions, designers could create a false environment. (The results obtained in a lab setting may not be useful during real-world interaction).
- By testing limited hypotheses, designers may miss more important problems that are not affected by the more constrained issues which they do examine.
- These techniques are not useful if designers only require formative evaluation for half-formed hypotheses.
- Users feel constrained by the rules of testing.

3. Co-operative evaluation techniques

Co-operative evaluation techniques are useful during the formative stages of design.

They are less hypothesis-driven and a good for getting feedback on partial implementations.

The experimenter sits with the user while they work through a series of tasks.

This can occur in the working context or in a quiet room.

Designers can use paper prototyping or partial implementations of the final interface.

The experimenter is free to talk to the user, but should not distract too much.

If the user needs help, the designer should offer it and take note of it.

Main point of the exercise: Subjects should voice their thoughts as they work with the system.

Vocalisations are encouraged, recorded and analysed in a rigorous manner.

This low cost technique is very good for providing rough and ready feedback.

Users feel directly involved in development.

Disadvantage:

- Provides qualitative feedback and not measurable results of empirical science.
- Co-operative evaluation is bad if designers are unaware of political pressures that might bias a user's responses.
4. Observational techniques

Observational techniques of ethno-methodology are so focussed on the tasks of daily life that it is difficult to establish any hypothesis at all.

E.g. Watching users and recording their problems with photocopy machines.
Ethno-methodology requires a neutral observer to enter the users' working lives unobtrusively. They should go in without any hypotheses and simply record what they see.

Ethnographic research is research in a natural setting as opposed to laboratory techniques. It is used to investigate working practices and customs.

Advantage:
* Provides useful feedback during an initial requirements' analysis.

In complex situations, it may be difficult to form hypotheses about users' tasks until designers have a clear understanding of the working problems facing the users.
This technique avoids the problem of alienation / irritation that can be created by the unthinking use of interviews and questionnaires.

Disadvantage:
* It requires a considerable amount of skill.
It is difficult to enter a working context, and observe without affecting users' tasks.

5. Query techniques

**Name and discuss two query techniques used in evaluation.**

a. Interviews
The interviewer must plan, preparing specific questions or making a list of topics to address.
The interview shouldn’t be too structured, so the interviewer can adapt questions to the user.
Structured interviews are easier to conduct and analyse than flexible interviews.
Flexible interviews pick up important details relating to the user’s experience.

Semi-structured interviews are good: based on leading questions, but flexible to investigate.

Advantages of interviews:
* The evaluator can vary the level of questioning to suit the context
* The evaluator can probe the user for more info on relevant issues

b. Questionnaires
Less flexible than interviews.
Advantages:
* A larger number of users can be included because it’s less time-consuming and labour intensive and the results can be analysed more rigorously.

Open-ended and closed questions can be used.
Questionnaires can be completed in fixed sessions, but they can also be administered independent of time and place, and without the presence of an evaluator.

6. Heuristic evaluation
User interface design experts evaluate the user interface according to usability principles.
E.g. ‘There should be a match between the system and the real world’.
E.g. ‘Users must always be aware of the current system status’.

**Process of heuristic evaluation:**
1. Briefing: Experts are told what to do
2. Evaluation: Each expert spends a few hours with the interface, using heuristics to identify problems
3. Debriefing: Experts meet to discuss their evaluations, prioritise problems and suggest solutions.

Advantage:
* There are fewer practical and ethical issues to take into account as users are not involved

Disadvantage:
* Experts often identify problems that aren’t really problems.
Heuristic evaluation should preferably be used along with other techniques and several evaluators should take part.

4. Social aspects

4.1. Computer technology as social devices

1. The Internet
a. What is the Internet?
The growth of websites and applications like IE, Netscape, and Mosaic has encouraged the active participation of new groups of users.

b. Why is the Internet important?
The huge amount of hype regarding the Internet is due to two factors:
* The growth of the WWW
* The increasing use of e-mail
The WWW grew from the National Centre for Supercomputer Applications (NCSA) and from CERN, a European research centre for nuclear physics. The work at NCSA led to the development of the Mosaic Web browser in 1993. Mosaic was free, then Netscape was its commercial successor, and then IE followed.

2. E-mail
   a. Addressing and routing
   If you send a message to someone with the same server, you can drop the location.
   b. Mail groups
   c. Filtering and uncertainty
   You can specify that certain mail be deleted / forwarded to another mailbox.
   **An example of filtering of email messages is forwarding mail from a particular user to a different mailbox.**
   MIME (Multipurpose Internet Mail Extensions) attachments enable users to include graphics, sounds and films.

3. News
   The system broadcasts a single file to all registered news servers on the network. Users can ‘subscribe’ to a newsgroup and read messages as they arrive at the local site.
   This isn’t like email because you don’t specify an account as the recipient of a message.
   a. Updates
   If you post a news article, every site linked to that newsgroup eventually gets a copy of the file
   b. Threads
   Threads are chains of similar messages.
   This offers a mechanism for filtering the mass of articles posted to a group.

4. Netiquette
   * Unwritten rules that govern good behaviour on the Internet*

5. The Web architecture
   a. What is the WWW?
   The WWW is an Internet client-server hypertext distributed information retrieval system which originated from the CERN physics labs in Geneva. Application or client programs, called browsers, translate user requests for info so that data can be transferred from remote servers. On the WWW all objects (documents, menus…) are represented to the user in HTML format.
   The client program, known as a browser (E.g. NN / IE) runs on the user’s computer and provides two basic navigation operations: Follow a link / send a query to a server. The browser includes e-mail handling facilities and can be used to access newsgroups. Most clients & servers also support ‘forms’ which allow the user to enter text / select options.
   b. URLs
   http = the low-level communications mechanism that is used to transport the file.
   c. Supply push
   Web browsers & servers are freely available, which pushes people to explore the Web.
   Modems & dial-up connection contracts included with new machines, provided WWW access.
   Free browsers enabled users of earlier Internet tools, like Gopher & ftp to try the new systems.
   Backwards compatibility — when new tools perform the functions of the earlier ones, as well as executing advanced functionality.
   Backwards compatibility encouraged people to explore HTML.
   E.g. NN & IE could be used to access files that had once been retrieved with older systems.
   d. Demand pull
   There was a pull from the material that began to appear on the Internet. (Shopping, games…)
   People not part of the WWW community are seen as losing out, which pulls more users to join
   e. Interface design
   The expansion of the WWW is entirely due to the standardised and successful interface design techniques that were incorporated into the browsers.
6. Java
The Java platform generates a device-independent byte code when a program is compiled. Each machine then interprets that code into a format it can understand. Each underlying platform must have its own Java interpreter or translator. Because of this, the Java Platform can provide a standard, uniform programming interface to applets and applications on any hardware.

a. Applets
- Programs that require a browser to run.
  The applet tag is embedded in a Web page and names the program to be run. When that page is accessed by a user, the applet automatically downloads from the server and runs on the client machine.

b. Applications
- Programs that do not require a browser to run - (have no built-in downloading mechanisms)
  When an application is called, it runs, so applications are like programs in other languages. Like an applet, an application requires the Java Platform for it to run. The platform can be a separate program / embedded within the OS or in the application itself.

4.2. Computers as communication devices

1. Computers as tools
Users shouldn’t be conscious that they’re using a computer, but focus on what they’re doing. Only when they have problems do they have to worry about finding the right menu or form.
Unless a user understands how a computer network operates, he or she cannot make optimal use of them.

2. Mobile computation

a. Limitations of traditional networks
The following limitations have led manufacturers to explore wireless technology:
  * Expensive to maintain
    A considerable amount of re-cabling is required every time a new node is added to a network. You can use connectors that are snapped on an existing cable / use connector boxes with plugs for users to hook up their devices.
    In each case, the physical characteristics of the wiring impose limits on the network's flexibility
  * Can be insecure
    You can attack a system by hooking onto the cabling and listening to traffic (= eavesdropping)
    Radio-based communication is less vulnerable: you need to find the transmission frequency.
  * May not support all user tasks
    Some systems, like mobile ones, can’t use conventional networks.
    The physical characteristics of wiring makes them inflexible.
  * Not good for consumer electronics
    Consumers don’t want to install a LAN between rooms in their houses.

b. Mobile computing infrastructure

Strengths and weaknesses of cellular technology
Networks for mobile communications have brought about many hand-held & laptop devices. Recent developments in the communications infrastructure enable users of these systems to access local & remote resources without being forced to connect to physical phone lines. These systems are, in turn, posing new challenges for human-computer interaction.

Cellular architectures
Radio technology offers the most obvious means of connecting mobile devices.
An area is divided into cells, and each cell has its own transceiver.
If a user moves to another cell, the call is passed between transceivers.
Problems of mobile computer systems:
  * High frequency signals carry more info, but are susceptible to interference & dispersion.
  * Low frequency signals carry less info, but will travel over longer distances.
  * There is signal fade with bad weather
  * There can be unintentional electromagnetic interference
  * There can be interference from other devices using the same channel
  * Signal strength can vary due to movement of the device
What will these systems look like?
First approach:
Future mobile devices could have underlying telecommunications that are invisible to the user (This will be hard to achieve: there will be delays while info is passed from a remote site)
Second approach:
Users could log into a remote transceiver before sending a message.
(This would let users choose whether to incur delays when transferring data between sites)
Strengths and weaknesses of satellite networks
Advantages:
* No multi-path transmission (which occurs when signals ‘bounce’ off objects)
Disadvantages:
* Satellites must filter & correct for atmospheric interference and for noise in space

GeoStationary satellites
They must maintain an orbit of 36 000 km in order to hold their position relative to earth.
This incurs a half-second delay on transmissions, which affects the usability of mobile devices
If an item of information is lost between transmitter and receiver, then several seconds may go by before the missing item can be detected.

Low earth satellites
They avoid this delay, but the user’s device must track a satellite’s movement across the sky.
Both forms of satellite communication suffer from a relatively low bandwidth (8-20Kbps).
This limits the range of tasks that users can perform over these links.

3. Groupware
* Programs that help several people work together on a common product.
E.g. Multi-user conferencing systems, CASE tools, command and control systems on an aircraft, group text-editors, virtual universities…
CSCW = Computer-Supported Co-operative Work.
These applications are particularly difficult to design and build.
Tele-working enables groups of users to remotely login to their place of work.
Tele-working can combine elements of mobile computation and groupware.

a. What is covered by the term?
* Conferencing systems
These systems reduce the problems that occur when several people try to talk at once.
(With videoconferencing systems, there are visual cues about who wants to talk next etc.)
* Multi-user text editors
These systems enable groups of users to simultaneously edit the same document.
Both groups can access the most recent version that the other group is working on.
* CASE tools
These systems help groups of programmers to develop code.
They can help teams to determine where their colleagues are currently concentrating.
* Command and control systems
These applications include computer interfaces to systems on-board aircraft etc…
The various operators must co-operate to preserve the safety of the application.

b. What are the problems?
* Synchronous and asynchronous systems
It may be difficult for users to know exactly who else is using the system.
Synchronous = at the same time; Asynchronous = at different times.
CSCW systems may be synchronous or asynchronous.
If the application is asynchronous then many problems of contention do not arise.
* Contention
Occurs when several users want to gain access to a resource that can’t be shared.
* Interference
This arises when one user frustrates another by getting in their way.
One person might want to move some text while another attempts to edit it.
* Different views
If people are working on different parts of a document, there can be confusion if one user has to tell another about the info on his screen if the other user can’t see it simultaneously.

c. What are the solutions?
Locking mechanisms
These techniques bar access to a shared resource for some period of time.
Problem of fairness: All users should be allocated the resource they require.
Issue of liveness: Any particular user must eventually be granted that resource.
Critical issue: **Granularity** of the lock (How fine are the grains? paragraph / document sized?)

The smaller the lock, the more flexible the system, the greater the complexity of managing it.

**Priority protocols**

You must decide what to do if several users simultaneously try acquiring the locked resource.

One solution: Associate priorities with particular users (which reduces fairness…)

**Voting**

The team takes a vote before anyone gains access to a document, paragraph, or sentence.

Disadvantage: massive overhead, because dozens of requests for votes might be generated.

**Split and independent views**

The screen can be split to show a high level view and one focussed on your own activity.

**Video portholes are one way of providing split-screen views.**

Video portholes enable users to view cameras in many different locations at the same time.

By double clicking on the image from a camera, the user’s can see that room in more detail.

---

### 4.3. Security

The field of computer security covers both the threats to computer applications and the protection mechanisms that preserve their safety.

#### 1. Why bother?

Reasons for security threats:

* **Increasing interconnection of computers**
  
  The increasing value of info transferred is increasing the importance of security.
  
  Companies don’t always keep additional manual records so computer records must be secure.
  
  In these cases, you should have back-up machines and **shadowing systems** to create copies.

* **Increasing technological sophistication of the population**
  
  Commercial and governmental organisations must always stay one step ahead of **hackers**.

#### 2. Different attitudes to digital data

Security can be difficult to achieve because people think that digital data is less important.

You should be more careful with digital security than with conventional systems because it is easier to copy, forge, and search digital information than physical sources.

#### 3. Information is valuable

Factors that contribute to increasing the value of information:

* **Once a secret is lost you can’t get it back**
  
  You can’t assume that someone with access to sensitive information won’t pass it on.

* **Intruders may not leave any footprints**
  
  If your system doesn’t have an efficient accounting system you may be unaware of intruders.

* **Digital data is easily destroyed**
  
  A few subtle changes to particular values might have a huge impact on a firm’s behaviour.
  
  (E.g. changing competitors’ prices so that yours are the cheapest).

#### 4. Who can you trust?

Most security violations within large organisations come from within.

Therefore, many companies have rules of disclosure that specify what you can / can’t reveal.

A particular concern is what repair facilities may be provided for machines with sensitive data.

#### 5. Inferences from information

**Absence of information**

People can make judgements based on the absence of information.

**Network monitoring**

Even if the intruder can’t see the message content, the amount of traffic is revealing.

**Transfer of funds**

If money enters / leaves an account, intruders may guess what transaction led to the payment.

#### 6. Technological threats

**a. Trojan horses**

* A virus that hides within / is designed to look like a legitimate program.

* An insidious piece of code hidden inside a program that offers different functionality.
E.g. While the program is running, another program tries to access your password file.

b. Time bombs
Employees may run programs that do damage when they’re gone.
Less severe than Trojan horses because the system can be left secure in spite of the damage

c. Worms
= Self-replicating programs.
They don’t attach themselves to other programs (like viruses), but copy themselves repeatedly in memory / on a disk drive until no memory / disk space is left.

d. Eavesdropping
Attackers can monitor traffic / measure electromagnetic radiation emitted from displays.
To prevent attackers accessing reconstructed images from a display, you’d need to have shields in the walls, floors and ceilings.

7. Information retrieval systems
Reasons why a huge mass of data doesn’t slow down intruders:
(The following helps intruders retrieve information illegally)
* Feedback mechanisms
These techniques give searchers info about how relevant a document is to a query (E.g. 50%)
* Efficient indexing
A system can allocate key terms to the documents so it doesn’t have to search everything.
* Natural language queries
It is no longer necessary for users to learn complex query languages to use these systems.

8. Protection mechanisms
describe three security protection mechanisms.

a. Encryption
= Keeping the key used to decode a message secret.
* Secret key encryption
An algorithm hides the message. To decode it, you need to know the key.
* Public key encryption
There are two keys - one to encrypt (publicly available) and another to decrypt (private).

b. Digital signatures
= A unique electronic sign on an electronic document that associates it with its owner.
Like public key encryption, digital signatures use two keys, but now the encryption key is private and the key used to decrypt the message is public.
Because only one person can correctly encode the data, it’s like adding a signature.

c. Access control lists
Capability-based approach:
Different users have different privileges (E.g. academic staff can do more than students).
When you log on with a password, all of your privileges are set with the system.
Access control lists:
These associate lists of users with objects in the system.
Only those users on the list can access that resource.
Lock and key method:
Some systems hide their files so that only users who know the exact location can access it.
The location of the files / their path, becomes the key.
The user presenting the key must have the correct status and appear on the resource’s list.

d. Security policies
= A set of assumptions that guide the application of security mechanisms.
You can reduce unauthorised users by introducing locks & keys and changing them often.
Fire-walling = erecting defences around particular areas of the system.
Password access and other checks ensure that even if someone gains access to one area, they can’t access other areas.
Other security policies associate privileges with groups.

4.4. Safety

1. Introduction
Luddism = a strong negative reaction against the introduction of technology.
There is an increasingly subservient relationship between people and computers.
Our everyday safety is almost entirely dependant upon information technology. Modern society is heavily supported by IT - if all computer systems failed then so would our food and power distribution networks so we are all dependant on machines to survive.

2. Risk
Some government authorities, such as the National Statistical Service in the UK, and regulatory organisations such as the British Health and Safety Inspectorate, maintain records of the number of deaths that can be attributed to technological failures. It is almost impossible to derive accurate estimates for computer-related deaths from statistics. This is because computers can’t kill - it is the systems they help control that cause accidents. We must examine the contribution that computers make to safety of the system as a whole. If we want to reduce the risk, we must find the parts of the system with the greatest failure risk.

a. What is risk?
Risk = frequency x cost. (This applies to safety, not security risks)
High frequency & low cost → irritations.
Low frequency & high cost → fatalities
Frequency
= The number of failures that occur within a specified time period
Determining failure frequency is a problem because you can’t test every path in a program.
Cost or utility
Cost can’t be determined solely by the immediate consequences of an incorrect instruction.
A bug might have effects in the subsequent performance of the system as a whole.

3. Hardware failure
a. Transient failures
= Failures that occur periodically and then rectify themselves, making them difficult to detect.
Hardware problems can be hard to identify if they leave few traces and no lasting damage.
b. Intermittent failures
= Failures that appear, disappear and then reappear later.
Just as difficult to detect as transient failures, because testing must coincide with failure.
c. Persistent failures
= Failures that continue for a prolonged period, making them the easiest to diagnose.
They are not necessarily easy to diagnose.

4. Hardware fault detection / resolution
a. Triple multiple redundancy
It involves taking a vote (on the correct output / answer) between several pieces of hardware.
E.g. 3 processors performing the same task simultaneously.
If the vote is not unanimous, a fault has occurred and the system follows the consensus.
You need an even number replicated components (i.e. an odd total) to avoid split decisions.
This increases cost, weight of the device, and amount of heat generated by the circuit.
(Heat is significant, because more faults will be introduced if there isn’t enough cooling).
If all components share the same design flaw they will agree on the same wrong decision.
(You should therefore use different hardware to perform the same function).
b. Signal comparison
For TMR to work, it must be possible to compare the output of different components.
This is difficult if a heterogeneous architecture is used.
E.g. Components from different designers might give their signals at different times.
c. Information redundancy
You can check the integrity of a signal by encoding additional info into the signal.
E.g. By adding a check digit to the end of some numbers that must equal their sum.
d. Watchdog timers
Watchdogs decrement a counter after a preset interval of time.
When the counter reaches zero, the watchdog issues instructions to reset the application.
The monitored application increments the counter; When it crashes, the counter falls to 0.
e. Bus monitoring
Buses transfer data between key components of the computer system.
E.g. A bus runs between the CPU and RAM.
Bus monitoring checks that info being accessed in RAM is in a permissible address.
If not, the CPU is informed of an error.

5. Software failure
   a. Requirements flaws
      Faults can be introduced during requirements capture if important requirements are omitted. E.g. If it was forgotten that a program should compute a result within 6 seconds.
   b. Design flaws
      Requirements elicitation establishes constraints that the software must satisfy, and design partly describes how those constraints are satisfied. Design decisions don’t include details of the code and errors can creep in (like if you don’t specify that a number can have only certain values).
   c. Implementation flaws
      These are bugs, such as dividing by 0.
   d. Testing flaws
      Testing can fail if there are problems in simulating the operating environment of a program.

   e. Why are requirements difficult?
      Functional requirements:
      Include aspects like whether the program will include software to compute particular results.
      Non-functional requirements:
      Refer to qualitative issues, like the provision of a usable interface. It’s easier to gather functional requirements because customers are focused on functions. The ease with which functions can be used is determined by non-functional requirements.
      Why requirements are difficult:
      * They often conflict with each other. (The person paying might not be the main user).
      * Users might want something that the client isn’t willing to pay for.
      * People are often vague about what they want.
      * Requirements change over time, as working practices evolve.

f. Why is testing difficult?
   It is difficult to know what to test with many programs.
   Solution 1 - Formal verification (Appropriate for testing a safety-critical system).
      Formal methods are mathematical techniques for proving program correctness. Prove that a program works for a base case and that it works for all numbers + 1.
   Solution 2 - Structured testing
      Test critical areas of a program, identified during the requirements stage. E.g. Test what happens when variables reach their upper & lower limits. Programmers must document the rationale behind any test they perform.

6. Human failure