HUMAN-COMPUTER INTERACTION

THIRD EDITION

DIX
FINLAY
ABOWD
BEALE

chapter 6

HCI in the software process

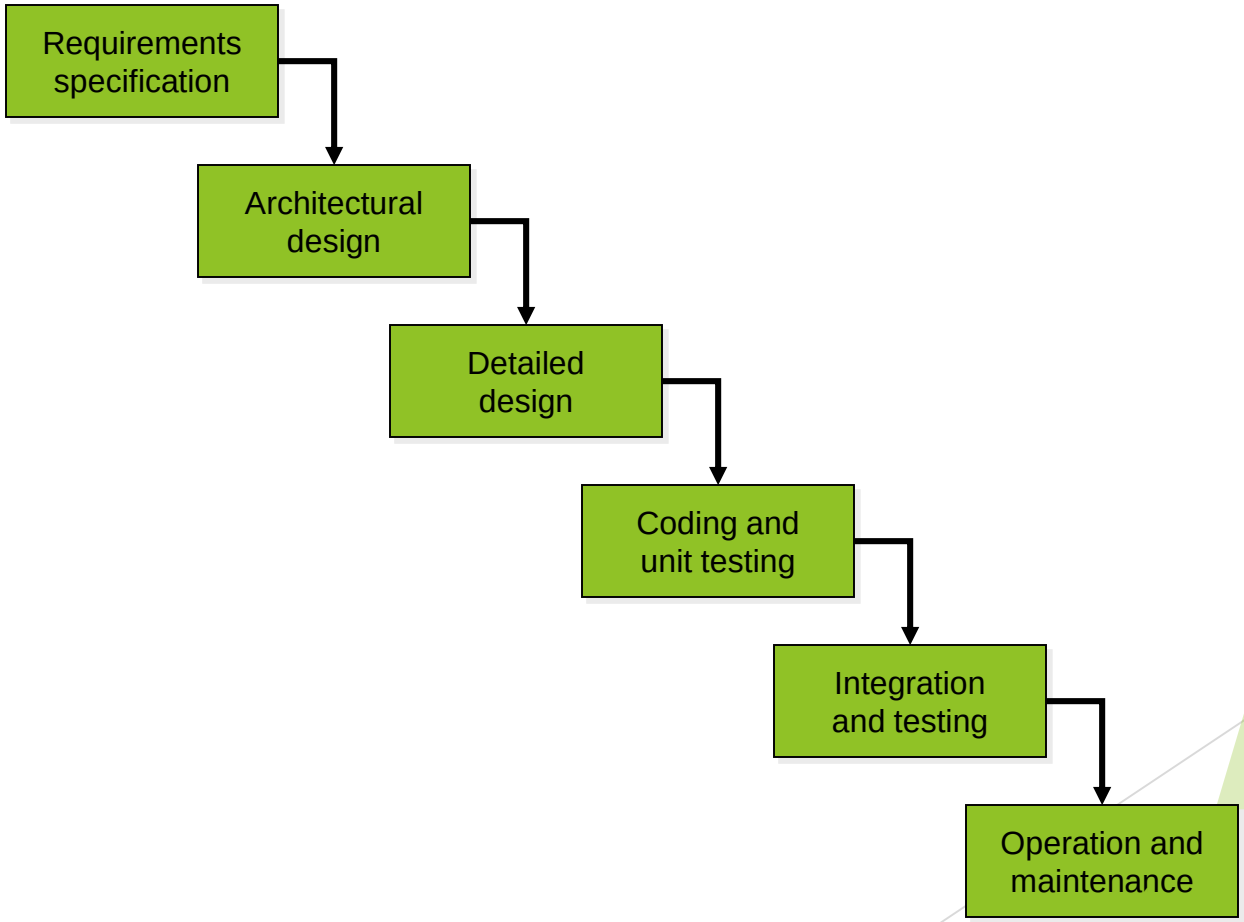
HCI in the software process

- Software engineering and the design process for interactive systems
- Usability engineering
- Iterative design and prototyping
- Design rationale

the software lifecycle

- Software engineering is the discipline for understanding the software design process, or life cycle
- Designing for usability occurs at all stages of the life cycle, not as a single isolated activity

The waterfall model



Activities in the life cycle

Requirements specification

designer and customer try capture what the system is expected to provide can be expressed in natural language or more precise languages, such as a task analysis would provide

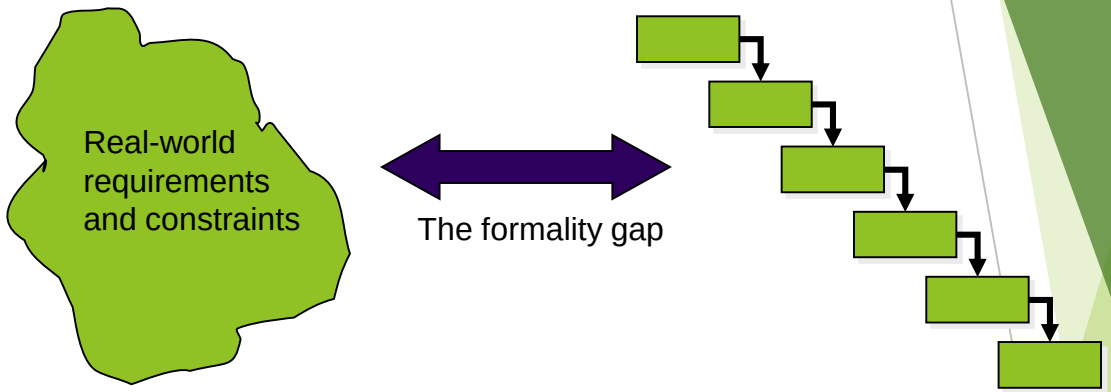
Architectural design

high-level description of how the system will provide the services required factor system into major components of the system and how they are interrelated needs to satisfy both functional and nonfunctional requirements

Detailed design

refinement of architectural components and interrelations to identify modules to be implemented separately the refinement is governed by the nonfunctional requirements

Verification and validation



Verification

designing the product right

Validation

designing the right product

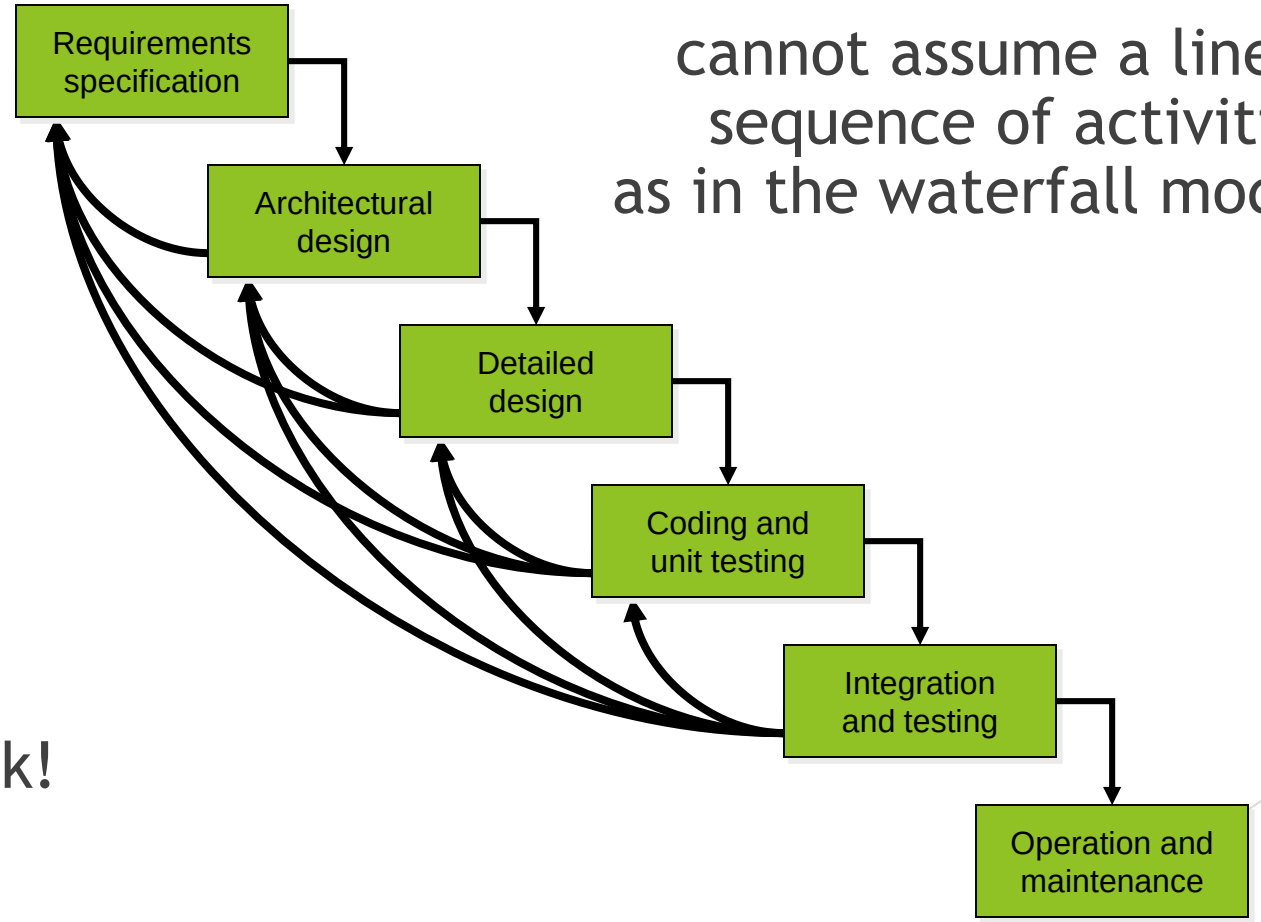
The formality gap

validation will always rely to some extent on subjective means of proof

Management and contractual issues

design in commercial and legal contexts

The life cycle for interactive systems



cannot assume a linear sequence of activities as in the waterfall model

lots of feedback!

Usability engineering

The ultimate test of usability based on measurement of user experience

Usability engineering demands that specific usability measures be made explicit as requirements

Usability specification

- usability attribute/principle
- measuring concept
- measuring method
- now level/ worst case/ planned level/ best case

Problems

- usability specification requires level of detail that may not be
- possible early in design satisfying a usability specification
- does not necessarily satisfy usability

part of a usability specification for a VCR

Attribute: Backward recoverability

Measuring concept:	Undo an erroneous programming sequence
Measuring method:	Number of explicit user actions to undo current program
Now level:	No current product allows such an undo
Worst case:	As many actions as it takes to program-in mistake
Planned level:	A maximum of two explicit user actions
Best case:	One explicit cancel action

ISO usability standard 9241

adopts traditional usability categories:

- effectiveness
  - can you achieve what you want to?
- efficiency
  - can you do it without wasting effort?
- satisfaction
  - do you enjoy the process?

some metrics from ISO 9241

Usability objective	Effectiveness measures	Efficiency measures	Satisfaction measures
Suitability for the task	Percentage of goals achieved	Time to complete a task	Rating scale for satisfaction
Appropriate for trained users	Number of power features used	Relative efficiency compared with an expert user	Rating scale for satisfaction with power features
Learnability	Percentage of functions learned	Time to learn criterion	Rating scale for ease of learning
Error tolerance	Percentage of errors corrected successfully	Time spent on correcting errors	Rating scale for error handling

Iterative design and prototyping

- Iterative design overcomes inherent problems of incomplete requirements
- Prototypes
  - simulate or animate some features of intended system
  - different types of prototypes
    - throw-away
    - incremental
    - evolutionary
- Management issues
  - time
  - planning
  - non-functional features
  - contracts

Techniques for prototyping

Storyboards

- need not be computer-based
- can be animated

Limited functionality simulations

- some part of system functionality provided by designers
- tools like HyperCard are common for these
- Wizard of Oz technique

Warning about iterative design

- design inertia – early bad decisions stay bad
- diagnosing real usability problems in prototypes….
  - …. and not just the symptoms

Design rationale

Design rationale is information that explains why a computer system is the way it is.

Benefits of design rationale

- communication throughout life cycle
- reuse of design knowledge across products
- enforces design discipline
- presents arguments for design trade-offs
- organizes potentially large design space
- capturing contextual information

Design rationale (cont'd)

Types of DR:

- Process-oriented
  - preserves order of deliberation and decision-making
- Structure-oriented
  - emphasizes post hoc structuring of considered design alternatives

- Two examples:
  - Issue-based information system (IBIS)
  - Design space analysis

Issue-based information system (IBIS)

- basis for much of design rationale research
- process-oriented
- main elements:

  issues

  – hierarchical structure with one ‘root’ issue

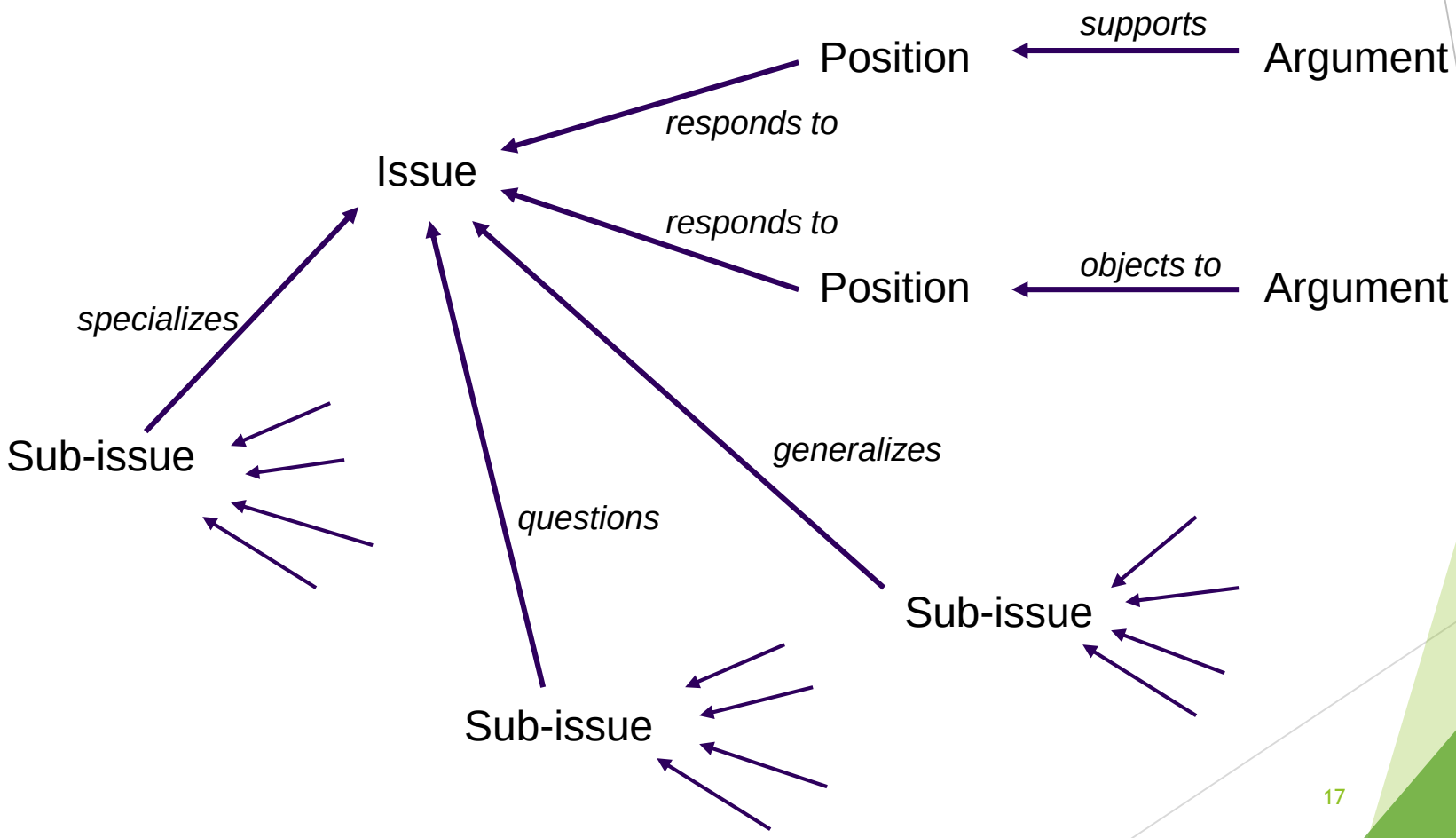
  positions

  – potential resolutions of an issue

  arguments

  – modify the relationship between positions and issues
- gIBIS is a graphical version

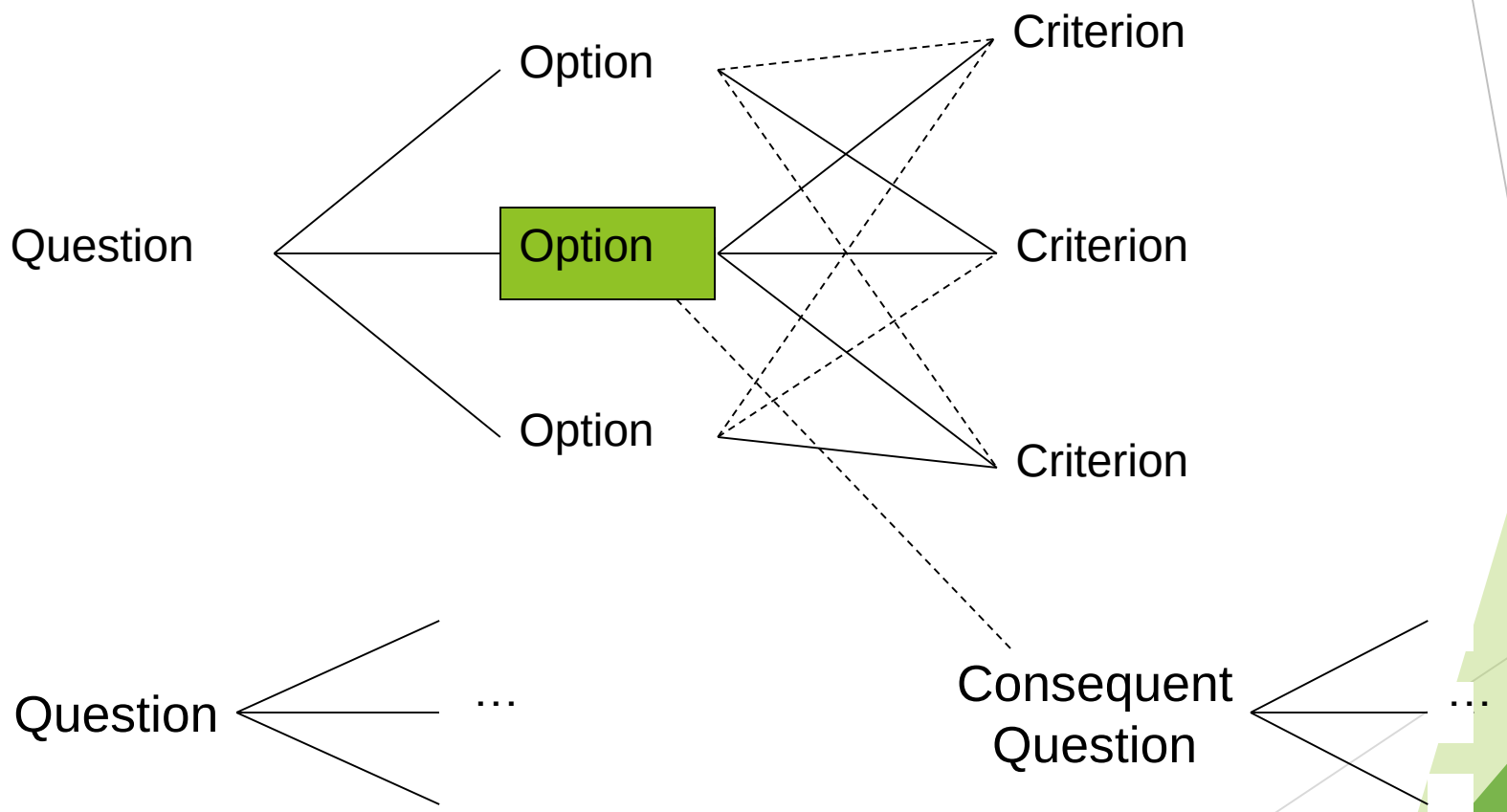
structure of gIBIS



Design space analysis

- structure-oriented
- QOC – hierarchical structure:
  - questions (and sub-questions)
    - represent major issues of a design
  - options
    - provide alternative solutions to the question
  - criteria
    - the means to assess the options in order to make a choice
- DRL – similar to QOC with a larger language and more formal semantics

the QOC notation



Psychological design rationale

- to support task-artefact cycle in which user tasks are affected by the systems they use
- aims to make explicit consequences of design for users
- designers identify tasks system will support
- scenarios are suggested to test task
- users are observed on system
- psychological claims of system made explicit
- negative aspects of design can be used to improve next iteration of design

Summary

The software engineering life cycle

- distinct activities and the consequences for interactive system design

Usability engineering

- making usability measurements explicit as requirements

Iterative design and prototyping

- limited functionality simulations and animations

Design rationale

- recording design knowledge
- process vs. structure

HUMAN-COMPUTER INTERACTION

THIRD EDITION

DIX
FINLAY
ABOWD
BEALE

chapter 7

design rules

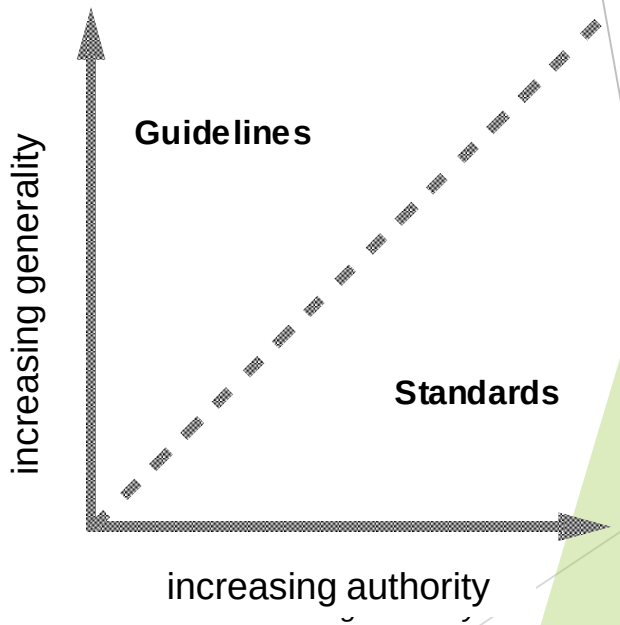
design rules

Designing for maximum usability
– the goal of interaction design

- Principles of usability
  - general understanding
- Standards and guidelines
  - direction for design
- Design patterns
  - capture and reuse design knowledge

types of design rules

- principles
  - abstract design rules
  - low authority
  - high generality
- standards
  - specific design rules
  - high authority
  - limited application
- guidelines
  - lower authority
  - more general application



Principles to support usability

Learnability

the ease with which new users can begin effective interaction and achieve maximal performance

Flexibility

the multiplicity of ways the user and system exchange information

Robustness

the level of support provided the user in determining successful achievement and assessment of goal-directed behaviour

Principles of learnability

Predictability

- determining effect of future actions based on past interaction history
- operation visibility

Synthesizability

- assessing the effect of past actions
- immediate vs. eventual honesty

Principles of learnability (ctd)

Familiarity

- how prior knowledge applies to new system
- guessability; affordance

Generalizability

- extending specific interaction knowledge to new situations

Consistency

- likeness in input/output behaviour arising from similar situations or task objectives

Principles of flexibility

Dialogue initiative

- freedom from system imposed constraints on input dialogue
- system vs. user pre-emptiveness

Multithreading

- ability of system to support user interaction for more than one task at a time
- concurrent vs. interleaving; multimodality

Task migratability

- passing responsibility for task execution between user and system

Principles of flexibility (ctd)

Substitutivity

- allowing equivalent values of input and output to be substituted for each other
- representation multiplicity; equal opportunity

Customizability

- modifiability of the user interface by user (adaptability) or system (adaptivity)

Principles of robustness

Observability

- ability of user to evaluate the internal state of the system from its perceivable representation
- browsability; defaults; reachability; persistence; operation visibility

Recoverability

- ability of user to take corrective action once an error has been recognized
- reachability; forward/backward recovery; commensurate effort

Principles of robustness (ctd)

Responsiveness

- how the user perceives the rate of communication with the system
- Stability

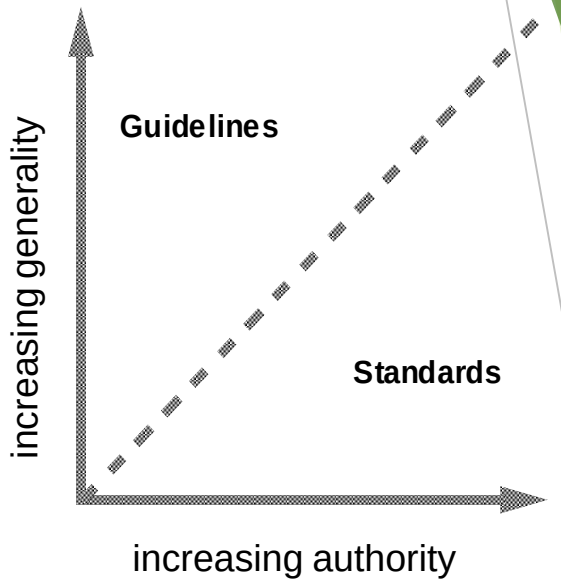
Task conformance

- degree to which system services support all of the user's tasks
- task completeness; task adequacy

Using design rules

Design rules

- suggest how to increase usability
- differ in generality and authority



Standards

- set by national or international bodies to ensure compliance by a large community of designers standards require sound underlying theory and slowly changing technology
- hardware standards more common than software high authority and low level of detail
- ISO 9241 defines usability as effectiveness, efficiency and satisfaction with which users accomplish tasks

Guidelines

- more suggestive and general
- many textbooks and reports full of guidelines
- abstract guidelines (principles) applicable during early life cycle activities
- detailed guidelines (style guides) applicable during later life cycle activities
- understanding justification for guidelines aids in resolving conflicts

Golden rules and heuristics

- “Broad brush” design rules
- Useful check list for good design
- Better design using these than using nothing!
- Different collections e.g.
  - Nielsen’s 10 Heuristics (see Chapter 9)
  - Shneiderman’s 8 Golden Rules
  - Norman’s 7 Principles

Shneiderman's 8 Golden Rules

1. *Strive for consistency*
2. *Enable frequent users to use shortcuts*
3. *Offer informative feedback*
4. *Design dialogs to yield closure*
5. *Offer error prevention and simple error handling*
6. *Permit easy reversal of actions*
7. *Support internal locus of control*
8. *Reduce short-term memory load*

Norman's 7 Principles

1. *Use both knowledge in the world and knowledge in the head.*
2. *Simplify the structure of tasks.*
3. *Make things visible: bridge the gulfs of Execution and Evaluation.*
4. *Get the mappings right.*
5. *Exploit the power of constraints, both natural and artificial.*
6. *Design for error.*
7. *When all else fails, standardize.*

HCI design patterns

- An approach to reusing knowledge about successful design solutions
- Originated in architecture: Alexander
- A pattern is an invariant solution to a recurrent problem within a specific context.
- Examples
  - Light on Two Sides of Every Room (architecture)
  - Go back to a safe place (HCI)
- Patterns do not exist in isolation but are linked to other patterns in *languages* which enable complete designs to be generated

HCI design patterns (cont.)

- Characteristics of patterns
  - capture design practice not theory
  - capture the essential common properties of good examples of design
  - represent design knowledge at varying levels: social, organisational, conceptual, detailed
  - embody values and can express what is humane in interface design
  - are intuitive and readable and can therefore be used for communication between all stakeholders
  - a pattern language should be generative and assist in the development of complete designs.

Summary

Principles for usability

- repeatable design for usability relies on maximizing benefit of one good design by abstracting out the general properties which can direct purposeful design
- The success of designing for usability requires both creative insight (new paradigms) and purposeful principled practice

Using design rules

- standards and guidelines to direct design activity

HUMAN-COMPUTER INTERACTION THIRD EDITION

DIX FINLAY ABOWD BEALE

chapter 8

implementation support

Implementation support

- programming tools
  - levels of services for programmers
- windowing systems
  - core support for separate and simultaneous user-system activity
- programming the application and control of dialogue
- interaction toolkits
  - bring programming closer to level of user perception
- user interface management systems
  - controls relationship between presentation and functionality

Introduction

How does HCI affect of the programmer?

Advances in coding have elevated programming

hardware specific
→ interaction-technique specific

Layers of development tools

- windowing systems
- interaction toolkits
- user interface management systems

Elements of windowing systems

Device independence

programming the abstract terminal device drivers

image models for output and (partially) input

- pixels
- PostScript (MacOS X, NextStep)
- Graphical Kernel System (GKS)
- Programmers' Hierarchical Interface to Graphics (PHIGS)

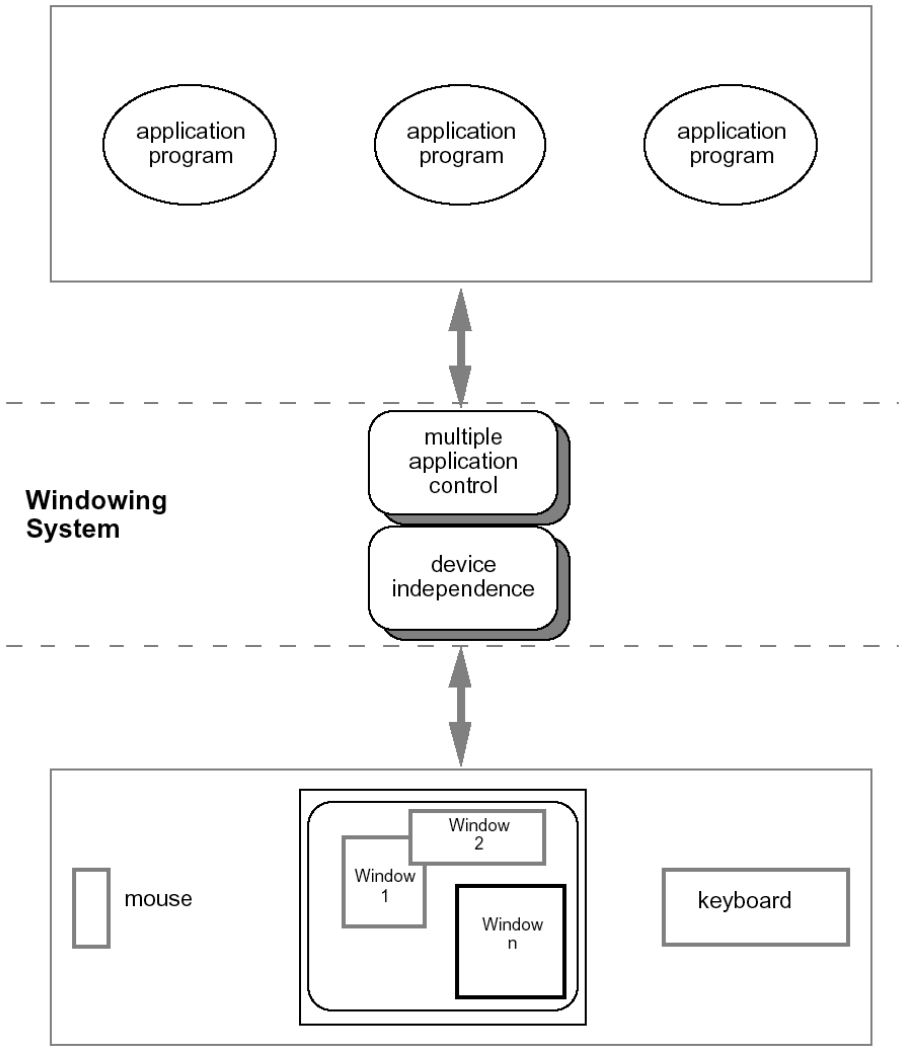
Resource sharing

achieving simultaneity of user tasks

window system supports independent processes

isolation of individual applications

roles of a windowing system



Architectures of windowing systems

three possible software architectures

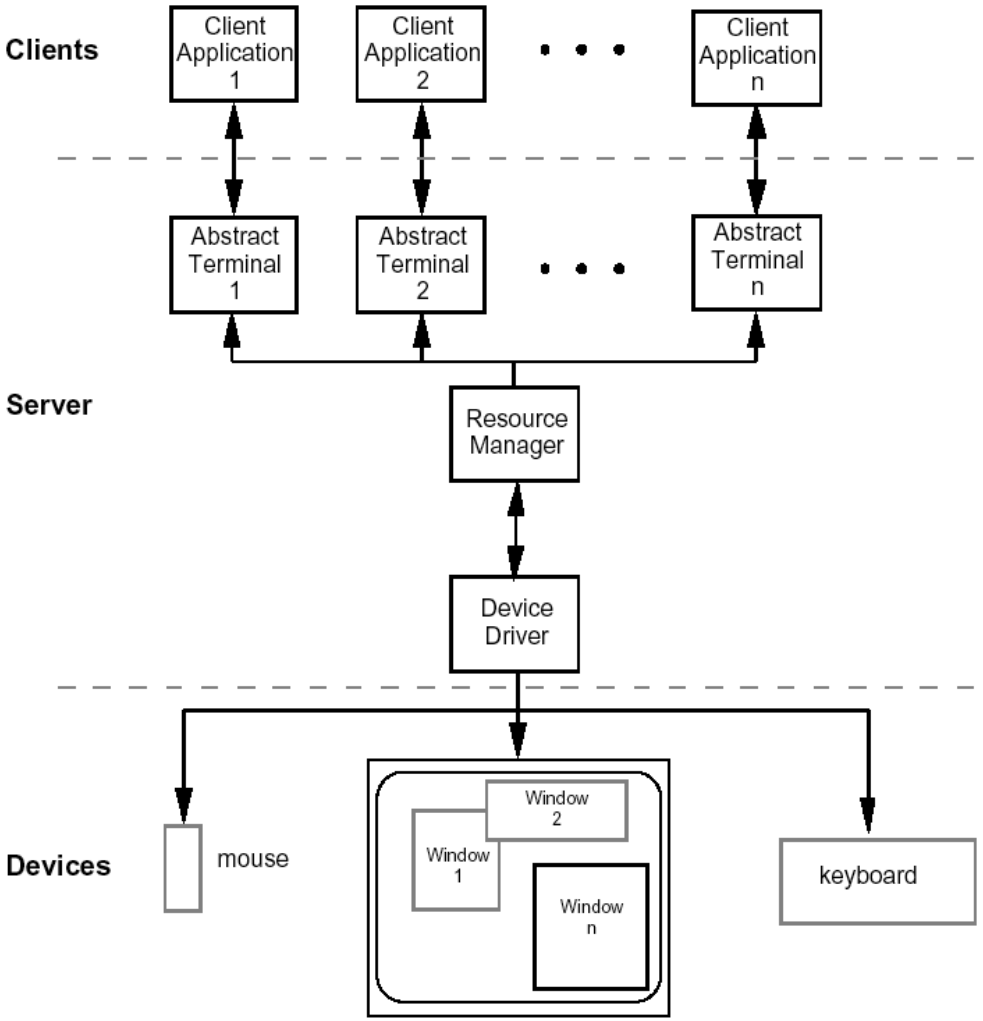
- all assume device driver is separate
- differ in how multiple application management is implemented

1. each application manages all processes
   - everyone worries about synchronization
   - reduces portability of applications

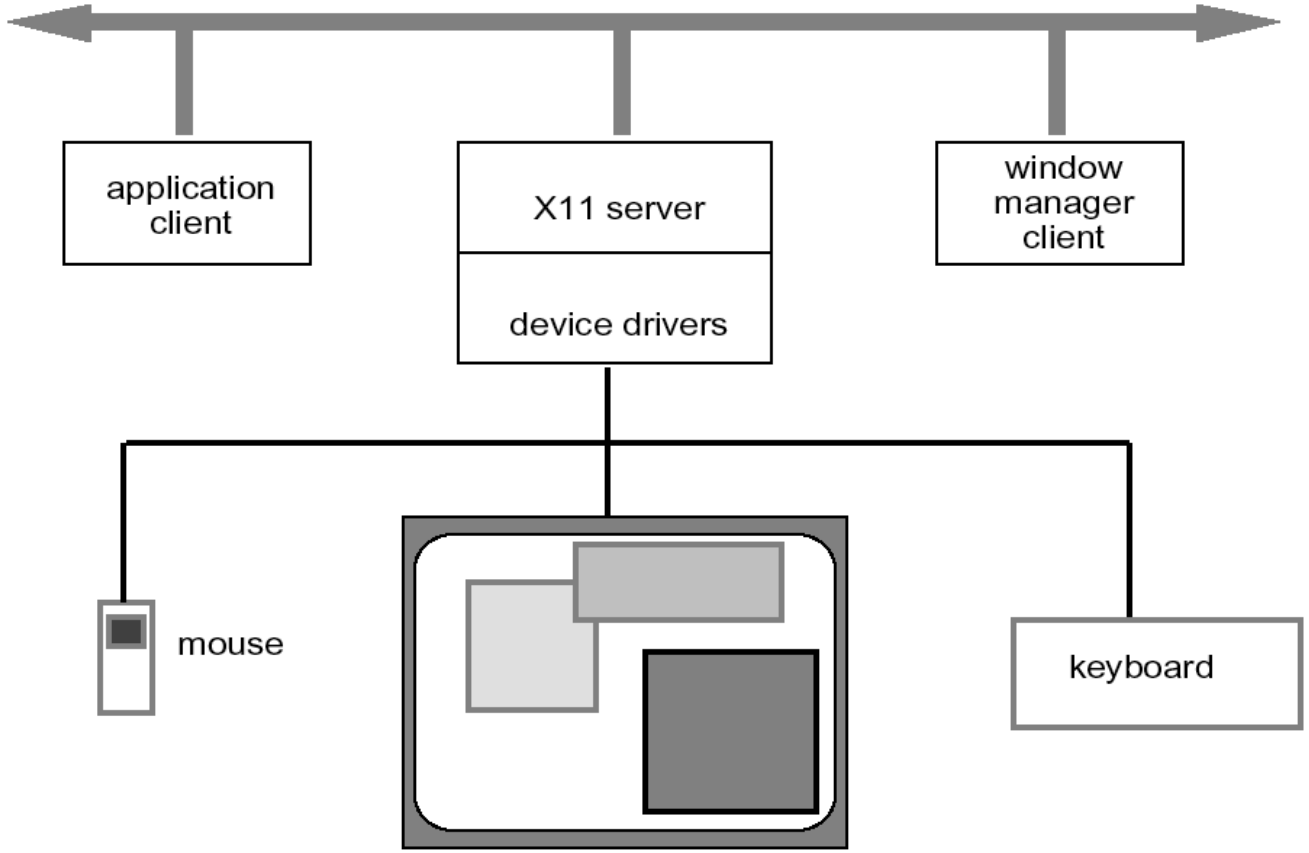
2. management role within kernel of operating system
   - applications tied to operating system

3. management role as separate application
   maximum portability

The client-server architecture



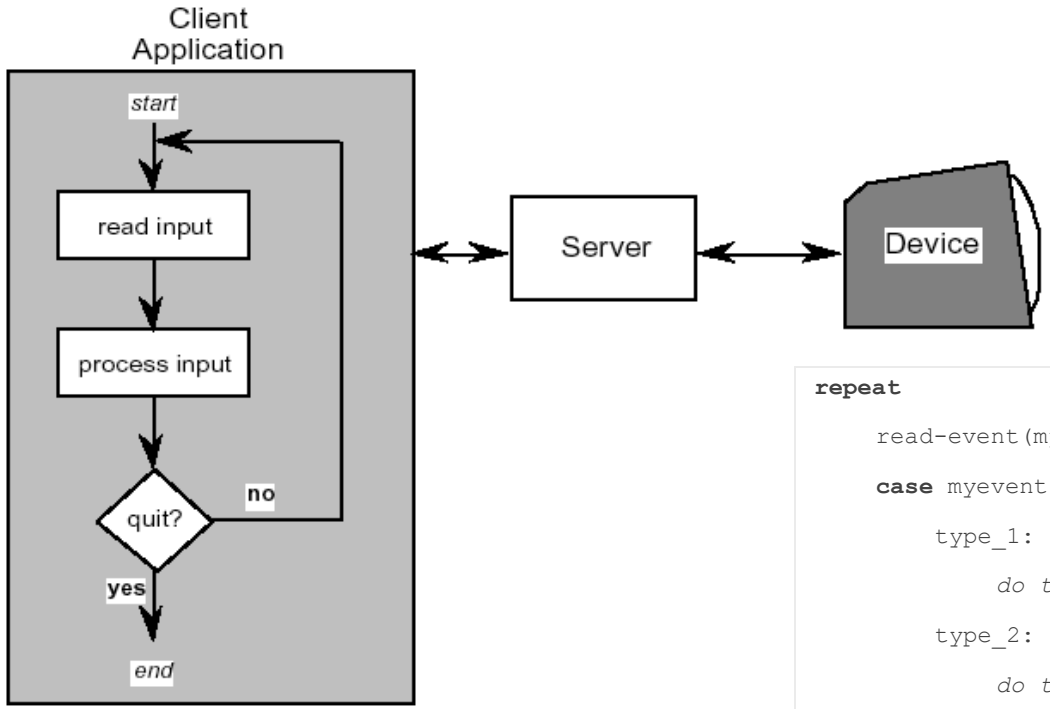
X Windows architecture



X Windows architecture (ctd)

- pixel imaging model with some pointing mechanism
- X protocol defines server-client communication
- separate window manager client enforces policies for input/output:
  - how to change input focus
  - tiled vs. overlapping windows
  - inter-client data transfer

Programming the application - 1
read-evaluation loop



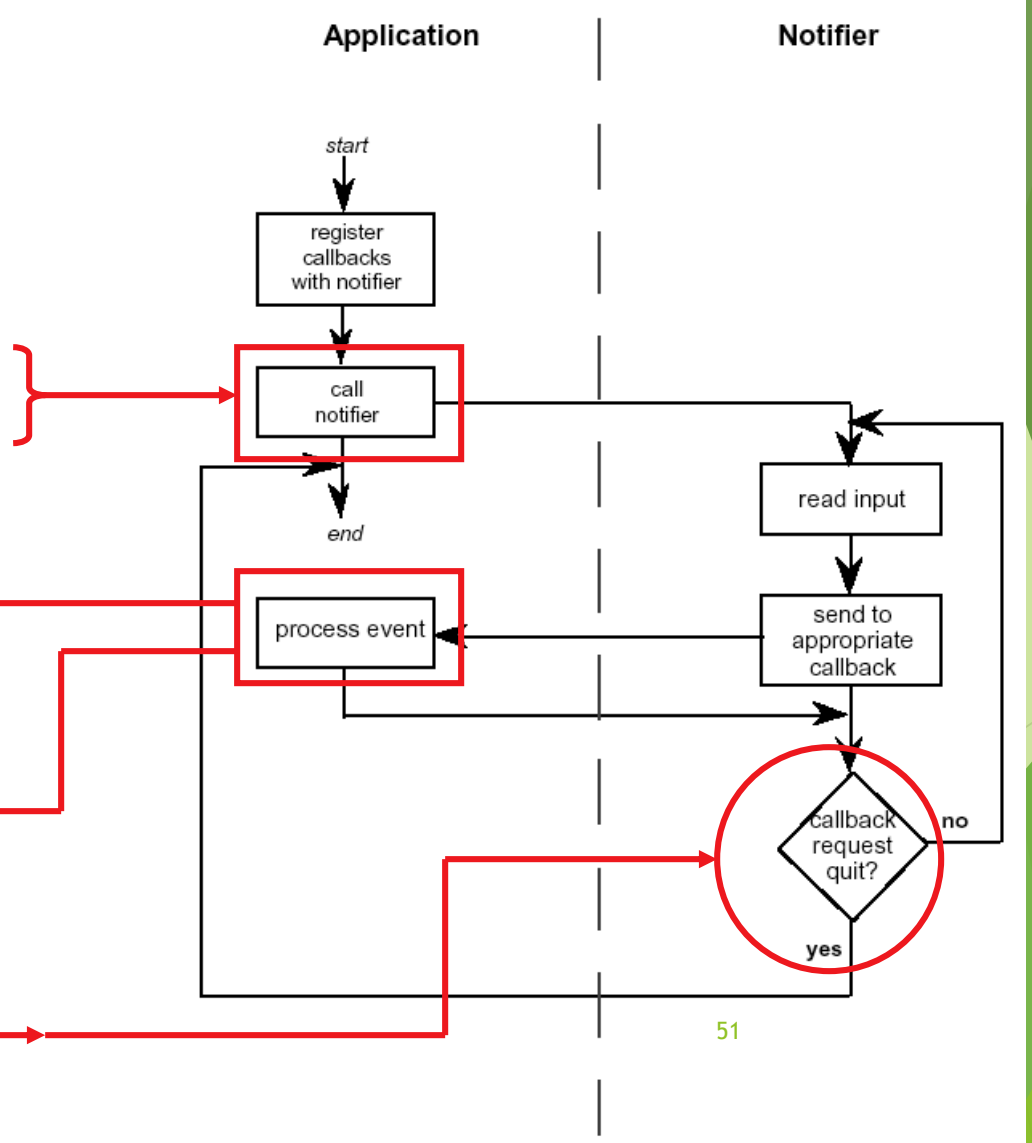

```
repeat
    read-event(myevent)
    case myevent.type
        type_1:
            do type_1 processing
        type_2:
            do type_2 processing
        ...
        type_n:
            do type_n processing
    end case
end repeat
```

Programming the application - 1
notification-based

```
void main(String[] args) {
    Menu menu = new Menu();
    menu.setOption("Save");
    menu.setOption("Quit");
    menu.setAction("Save",mySave)
    menu.setAction("Quit",myQuit)
        ...
}

int mySave(Event e) {
    //  save the current file
}

int myQuit(Event e) {
    //  close down
}
```



going with the grain

- system style affects the interfaces
  - modal dialogue box
    - easy with event-loop (just have extra read-event loop)
    - hard with notification (need lots of mode flags)
  - non-modal dialogue box
    - hard with event-loop (very complicated main loop)
    - easy with notification (just add extra handler)

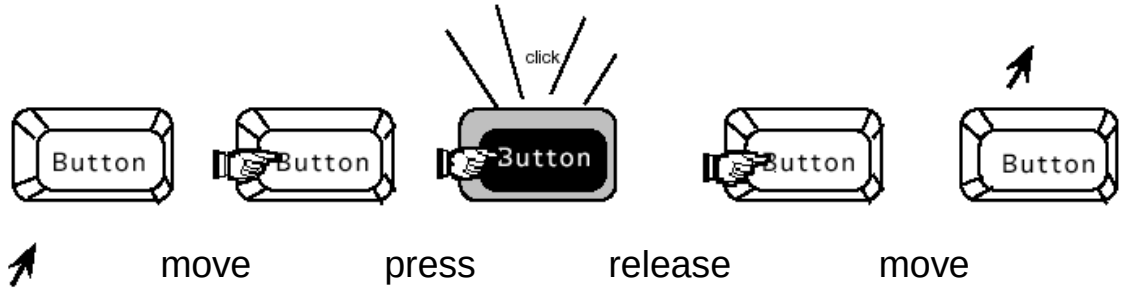
beware!

if you don't explicitly design it will just happen
implementation should not drive design

Using toolkits

Interaction objects

- input and output intrinsically linked



Toolkits provide this level of abstraction

- programming with interaction objects (or
- techniques, widgets, gadgets)
- promote consistency and generalizability
- through similar look and feel
- amenable to object-oriented programming

interfaces in Java

- Java toolkit – AWT (abstract windowing toolkit)
- Java classes for buttons, menus, etc.
- Notification based;
  - AWT 1.0 – need to subclass basic widgets
  - AWT 1.1 and beyond -- callback objects
- Swing toolkit
  - built on top of AWT – higher level features
  - uses MVC architecture (see later)

User Interface Management Systems (UIMS)

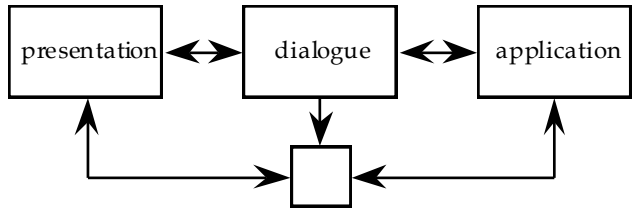
- UIMS add another level above toolkits
  - toolkits too difficult for non-programmers
- concerns of UIMS
  - conceptual architecture
  - implementation techniques
  - support infrastructure
- non-UIMS terms:
  - UI development system (UIDS)
  - UI development environment (UIDE)
    - e.g. Visual Basic

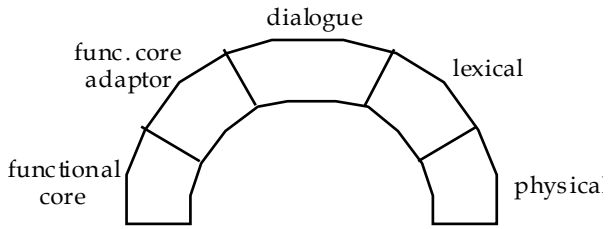
UIMS as conceptual architecture

- *separation* between application semantics and presentation
- improves:
  - portability – runs on different systems
  - reusability – components reused cutting costs
  - multiple interfaces – accessing same functionality
  - customizability – by designer and user

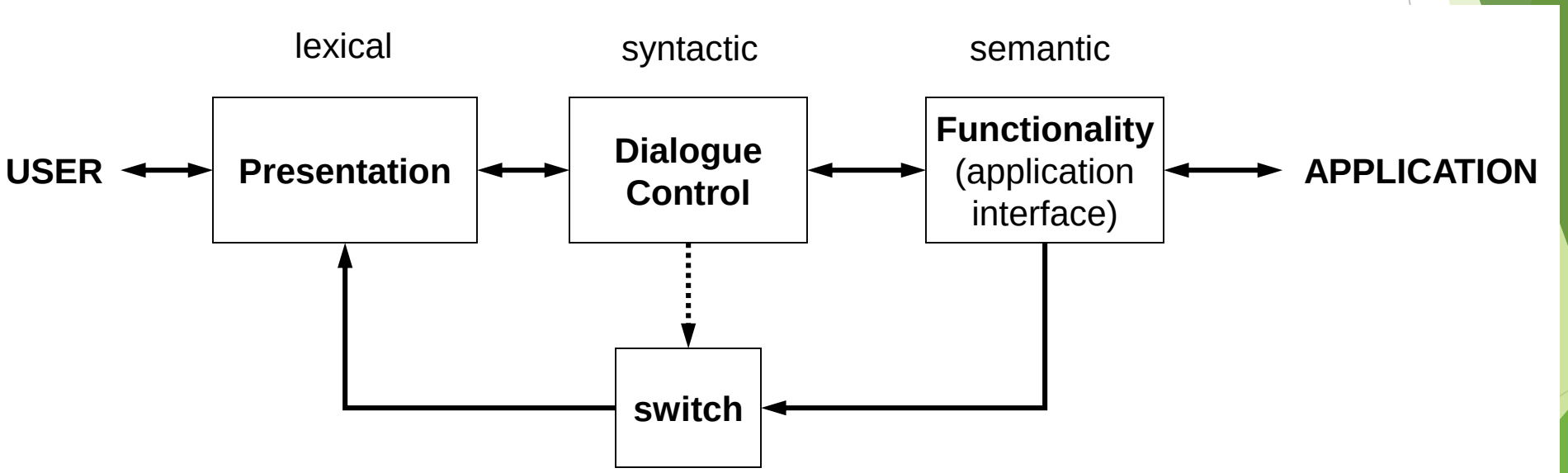
UIMS tradition – interface layers / logical components

- linguistic: lexical/syntactic/semantic
- Seeheim:
- Arch/Slinky





Seeheim model



conceptual vs. implementation

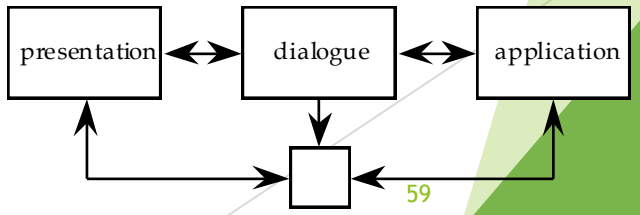
Seeheim

- arose out of implementation experience
- but principal contribution is conceptual
- concepts part of ‘normal’ UI language

… because of Seeheim …
… we think differently!

e.g. the lower box, the switch

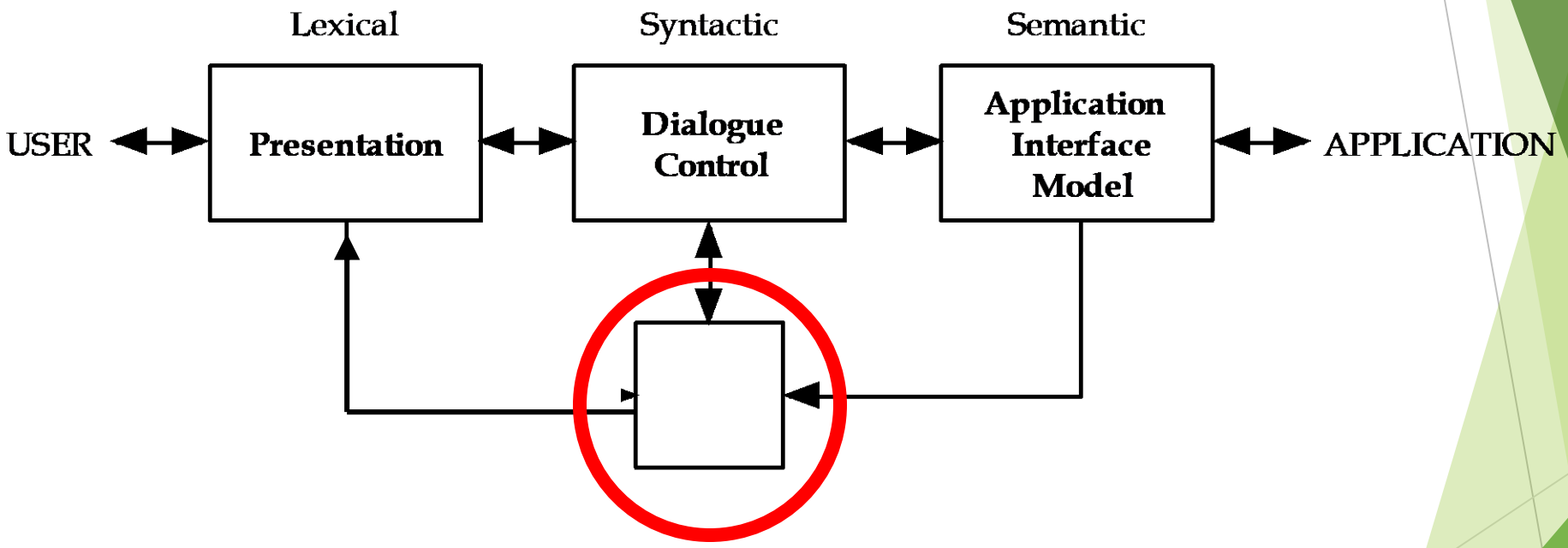
- needed for implementation
- but not conceptual



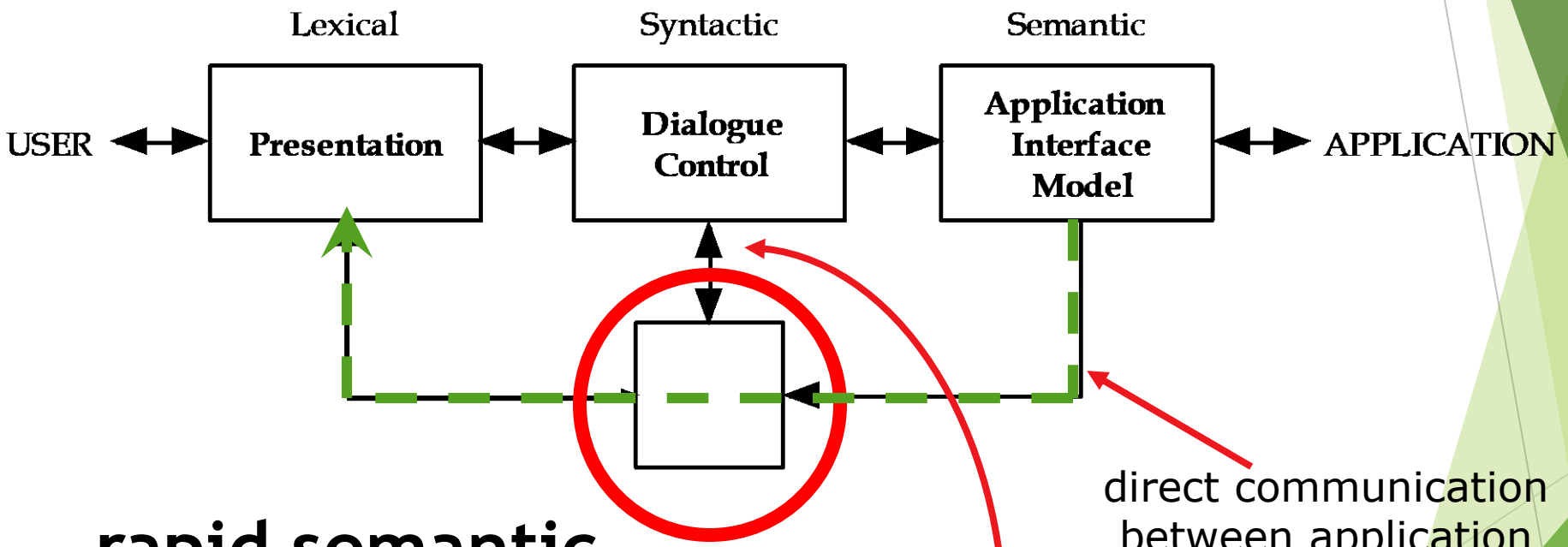
semantic feedback

- different kinds of feedback:
  - lexical – movement of mouse
  - syntactic – menu highlights
  - semantic – sum of numbers changes
- semantic feedback often slower
  - use rapid lexical/syntactic feedback
- but may need rapid semantic feedback
  - freehand drawing
  - highlight trash can or folder when file dragged

what's this?


Lexical
Syntactic
Semantic
USER
Presentation
Dialogue Control
Application Interface Model
APPLICATION

the bypass/switch

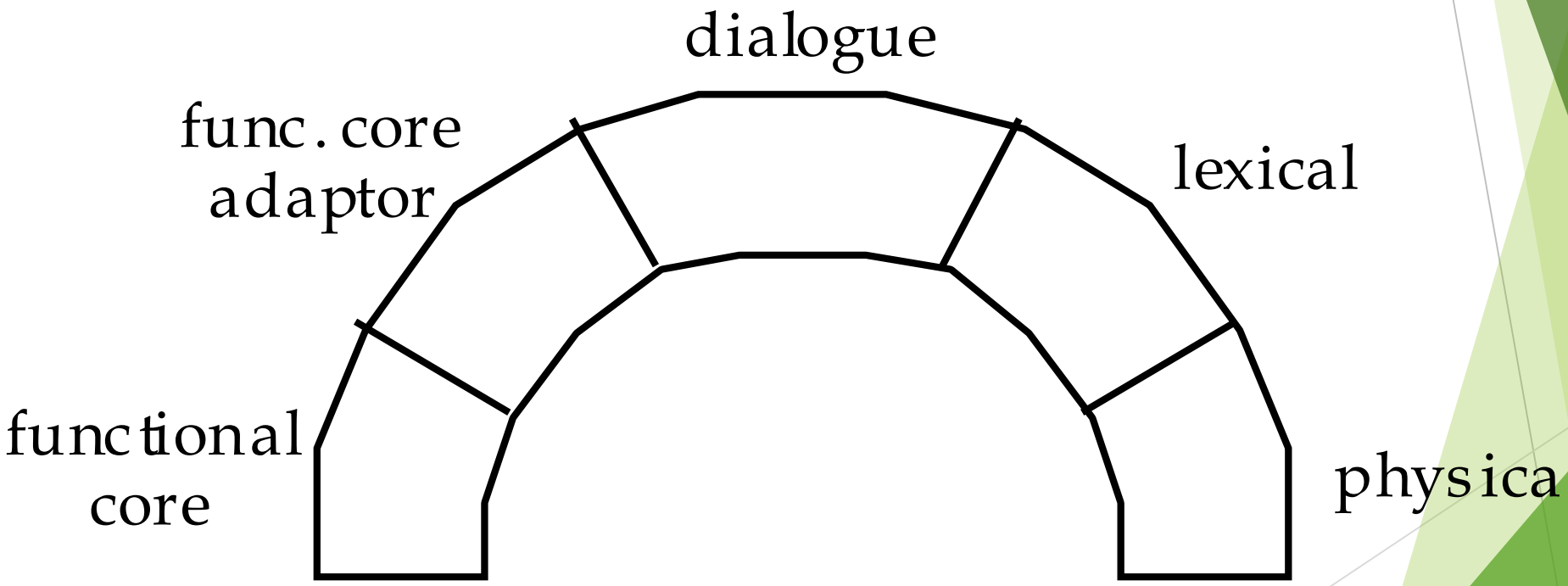


rapid semantic feedback

direct communication between application and presentation

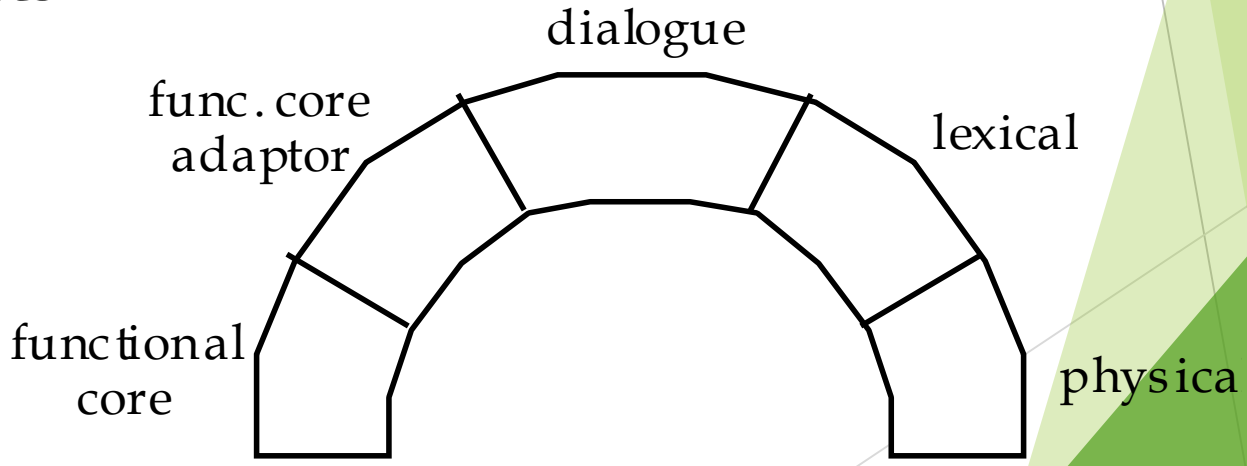
but regulated by dialogue control

more layers!


dialogue
func. core
adaptor
lexical
functional
core
physica

Arch/Slinky

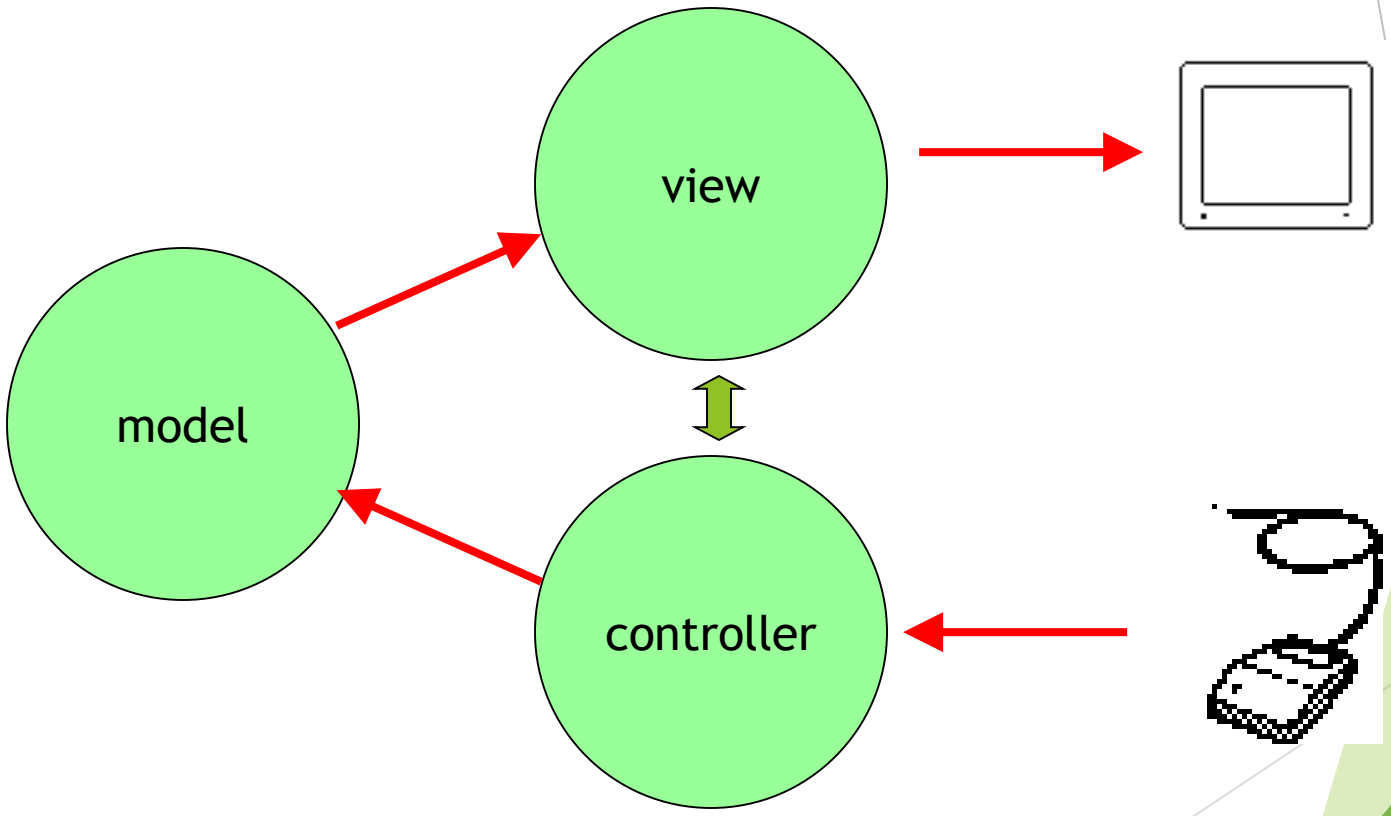
- more layers! – distinguishes lexical/physical
- like a ‘slinky’ spring different layers may be thicker (more important) in different systems
- or in different components



monolithic vs. components

- Seeheim has big components
- often easier to use smaller ones
  - esp. if using object-oriented toolkits
- Smalltalk used MVC – model–view–controller
  - model – internal logical state of component
  - view – how it is rendered on screen
  - controller – processes user input

MVC
model - view - controller



MVC issues

- MVC is largely pipeline model:

  input → control → model → view → output

- but in graphical interface
  - input only has meaning in relation to output

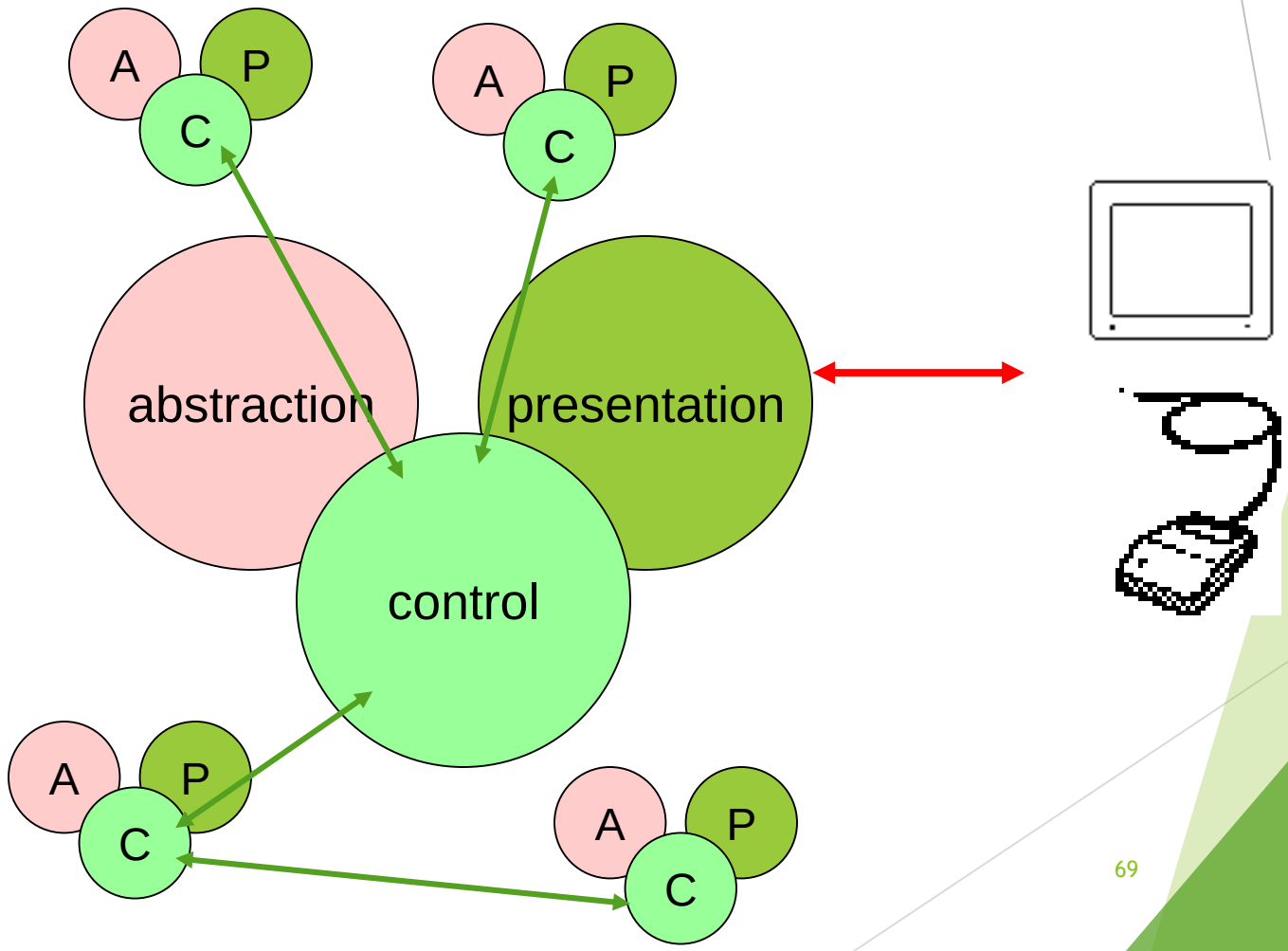
  e.g. mouse click
  - need to know *what* was clicked
  - controller has to decide what to do with click
  - but view knows what is shown where!
- in practice controller ‘talks’ to view
  - separation not complete

PAC model

- PAC model closer to Seeheim
  - abstraction – logical state of component
  - presentation – manages input and output
  - control – mediates between them
- manages hierarchy and multiple views
  - control part of PAC objects communicate
- PAC cleaner in many ways …
  but MVC used more in practice

  (e.g. Java Swing)

PAC
presentation - abstraction - control



Implementation of UIMS

- Techniques for dialogue controller
  - menu networks
  - grammar notations
  - declarative languages
  - graphical specification
  - state transition diagrams
  - event languages
  - constraints
  - for most of these see chapter 16
- N.B. constraints
  - instead of what *happens* say what should be *true*
  - used in groupware as well as single user interfaces

    (ALV - abstraction–link–view)

see chapter 16 for more details on several of these

graphical specification

- what it is
  - draw components on screen
  - set actions with script or links to program
- in use
  - with raw programming most popular technique
  - e.g. Visual Basic, Dreamweaver, Flash
- local vs. global
  - hard to ‘see’ the paths through system
  - focus on what can be seen on one screen

The drift of dialogue control

- internal control

  (e.g., read-evaluation loop)

- external control

  (independent of application semantics or presentation)

- presentation control

  (e.g., graphical specification)

Summary

Levels of programming support tools

- Windowing systems
  - device independence
  - multiple tasks
- Paradigms for programming the application
  - read-evaluation loop
  - notification-based
- Toolkits
  - programming interaction objects
- UIMS
  - conceptual architectures for separation
  - techniques for expressing dialogue

HUMAN-COMPUTER INTERACTION THIRD EDITION

DIX FINLAY ABOWD BEALE

chapter 9

evaluation techniques

Evaluation Techniques

- Evaluation
  - tests usability and functionality of system
  - occurs in laboratory, field and/or in collaboration with users
  - evaluates both design and implementation
  - should be considered at all stages in the design life cycle

Goals of Evaluation

- assess extent of system functionality
- assess effect of interface on user
- identify specific problems

Evaluating Designs

Cognitive Walkthrough

Heuristic Evaluation

Review-based evaluation

Cognitive Walkthrough

Proposed by Polson *et al*.

- evaluates design on how well it supports user in learning task
- usually performed by expert in cognitive psychology
- expert ‘walks though’ design to identify potential problems using psychological principles
- forms used to guide analysis

Cognitive Walkthrough (ctd)

- For each task walkthrough considers
  - what impact will interaction have on user?
  - what cognitive processes are required?
  - what learning problems may occur?

- Analysis focuses on goals and knowledge: does the design lead the user to generate the correct goals?

Heuristic Evaluation

- Proposed by Nielsen and Molich.
- usability criteria (heuristics) are identified
- design examined by experts to see if these are violated
- Example heuristics
  - system behaviour is predictable
  - system behaviour is consistent
  - feedback is provided
- Heuristic evaluation `debugs' design.

Review-based evaluation

- Results from the literature used to support or refute parts of design.
- Care needed to ensure results are transferable to new design.
- Model-based evaluation
- Cognitive models used to filter design options
  e.g. GOMS prediction of user performance.
- Design rationale can also provide useful evaluation information

Evaluating through user Participation

Laboratory studies

- Advantages:
  - specialist equipment available
  - uninterrupted environment
- Disadvantages:
  - lack of context
  - difficult to observe several users cooperating
- Appropriate
  - if system location is dangerous or impractical for constrained single user systems to allow controlled manipulation of use

Field Studies

- Advantages:
  - natural environment
  - context retained (though observation may alter it)
  - longitudinal studies possible
- Disadvantages:
  - distractions
  - noise
- Appropriate
  - where context is crucial for longitudinal studies

Evaluating Implementations

Requires an artefact:

simulation, prototype,

full implementation

Experimental evaluation

- controlled evaluation of specific aspects of interactive behaviour
- evaluator chooses hypothesis to be tested
- a number of experimental conditions are considered which differ only in the value of some controlled variable.
- changes in behavioural measure are attributed to different conditions

Experimental factors

- Subjects
  - who – representative, sufficient sample
- Variables
  - things to modify and measure
- Hypothesis
  - what you'd like to show
- Experimental design
  - how you are going to do it

Variables

- independent variable (IV)

  characteristic changed to produce different conditions

  e.g. interface style, number of menu items

- dependent variable (DV)

  characteristics measured in the experiment

  e.g. time taken, number of errors.

Hypothesis

- prediction of outcome
  - framed in terms of IV and DV

    e.g. “error rate will increase as font size decreases”

- null hypothesis:
  - states no difference between conditions
  - aim is to disprove this

    e.g. null hyp. = “no change with font size”

Experimental design

- within groups design
  - each subject performs experiment under each condition.
  - transfer of learning possible
  - less costly and less likely to suffer from user variation.
- between groups design
  - each subject performs under only one condition
  - no transfer of learning
  - more users required
  - variation can bias results.

Analysis of data

- Before you start to do any statistics:
  - look at data
  - save original data
- Choice of statistical technique depends on
  - type of data
  - information required
- Type of data
  - discrete - finite number of values
  - continuous - any value

Analysis - types of test

- parametric
  - assume normal distribution
  - robust
  - powerful
- non-parametric
  - do not assume normal distribution
  - less powerful
  - more reliable
- contingency table
  - classify data by discrete attributes
  - count number of data items in each group

Analysis of data (cont.)

- What information is required?
  - is there a difference?
  - how big is the difference?
  - how accurate is the estimate?

- Parametric and non-parametric tests mainly address first of these

Experimental studies on groups

More difficult than single-user experiments

Problems with:

- subject groups
- choice of task
- data gathering
- analysis

Subject groups

larger number of subjects
$\Rightarrow$ more expensive

longer time to `settle down'
… even more variation!

difficult to timetable

so … often only three or four groups

The task

must encourage cooperation

perhaps involve multiple channels

options:

- creative task — e.g. '*write a short report on …*'
- decision games — e.g. desert survival task
- control task — e.g. ARKola bottling plant

Data gathering

several video cameras
+ direct logging of application

problems:

- synchronisation
- sheer volume!

one solution:

- record from each perspective

Analysis

N.B. vast variation between groups

solutions:

- within groups experiments
- micro-analysis (e.g., gaps in speech)
- anecdotal and qualitative analysis

look at interactions between group and media

controlled experiments may `waste' resources!

Field studies

Experiments dominated by group formation

Field studies more realistic:

- *distributed cognition* ⇒ work studied in context
- real action is *situated action*
- physical and social environment both crucial

Contrast:

- psychology – controlled experiment
- sociology and anthropology – open study and rich data

Observational Methods

Think Aloud

Cooperative evaluation

Protocol analysis

Automated analysis

Post-task walkthroughs

Think Aloud

- user observed performing task
- user asked to describe what he is doing and why, what he thinks is happening etc.

- Advantages
  - simplicity - requires little expertise
  - can provide useful insight
  - can show how system is actually use
- Disadvantages
  - subjective
  - selective
  - act of describing may alter task performance

Cooperative evaluation

- variation on think aloud
- user collaborates in evaluation
- both user and evaluator can ask each other questions throughout

- Additional advantages
  - less constrained and easier to use
  - user is encouraged to criticize system
  - clarification possible

Protocol analysis

- paper and pencil – cheap, limited to writing speed
- audio – good for think aloud, difficult to match with other protocols
- video – accurate and realistic, needs special equipment, obtrusive
- computer logging – automatic and unobtrusive, large amounts of data difficult to analyze
- user notebooks – coarse and subjective, useful insights, good for longitudinal studies

- Mixed use in practice.
- audio/video transcription difficult and requires skill.
- Some automatic support tools available

automated analysis – EVA

- Workplace project
- Post task walkthrough
  - user reacts on action after the event
  - used to fill in intention
- Advantages
  - analyst has time to focus on relevant incidents
  - avoid excessive interruption of task
- Disadvantages
  - lack of freshness
  - may be post-hoc interpretation of events

post-task walkthroughs

- transcript played back to participant for comment
  - immediately → fresh in mind
  - delayed → evaluator has time to identify questions
- useful to identify reasons for actions and alternatives considered
- necessary in cases where think aloud is not possible

Query Techniques

Interviews

Questionnaires

Interviews

- analyst questions user on one-to -one basis usually based on prepared questions
- informal, subjective and relatively cheap

- Advantages
  - can be varied to suit context
  - issues can be explored more fully
  - can elicit user views and identify unanticipated problems
- Disadvantages
  - very subjective
  - time consuming

Questionnaires

- Set of fixed questions given to users
- Advantages
  - quick and reaches large user group
  - can be analyzed more rigorously
- Disadvantages
  - less flexible
  - less probing

Questionnaires (ctd)

- Need careful design
  - what information is required?
  - how are answers to be analyzed?
- Styles of question
  - general
  - open-ended
  - scalar
  - multi-choice
  - ranked

Physiological methods

Eye tracking

Physiological measurement

eye tracking

- head or desk mounted equipment tracks the position of the eye
- eye movement reflects the amount of cognitive processing a display requires
- measurements include
  - fixations: eye maintains stable position. Number and duration indicate level of difficulty with display
  - saccades: rapid eye movement from one point of interest to another
  - scan paths: moving straight to a target with a short fixation at the target is optimal

physiological measurements

- emotional response linked to physical changes
- these may help determine a user's reaction to an interface
- measurements include:
  - heart activity, including blood pressure, volume and pulse.
  - activity of sweat glands: Galvanic Skin Response (GSR)
  - electrical activity in muscle: electromyogram (EMG)
  - electrical activity in brain: electroencephalogram (EEG)
- some difficulty in interpreting these physiological responses - more research needed

Choosing an Evaluation Method

when in process:	design vs. implementation
style of evaluation:	laboratory vs. field
how objective:	subjective vs. objective
type of measures:	qualitative vs. quantitative
level of information:	high level vs. low level
level of interference:	obtrusive vs. unobtrusive
resources available:	time, subjects, equipment, expertise

chapter 15

task models

What is Task Analysis?

Methods to analyse people's jobs:

- what people do
- what things they work with
- what they must know

An Example

- in order to clean the house
  - get the vacuum cleaner out
  - fix the appropriate attachments
  - clean the rooms
  - when the dust bag gets full, empty it
  - put the vacuum cleaner and tools away
- must know about:
  - vacuum cleaners, their attachments, dust bags, cupboards, rooms etc.

Approaches to task analysis

- Task decomposition
  - splitting task into (ordered) subtasks
- Knowledge based techniques
  - what the user knows about the task and how it is organised
- Entity/object based analysis
  - relationships between objects, actions and the people who perform them
- lots of different notations/techniques

general method

- observe
- collect unstructured lists of words and actions
- organize using notation or diagrams

Differences from other techniques

Systems analysis		**vs.**		**Task analysis**
system design	-	focus	-	the user
Cognitive models		**vs.**		**Task analysis**
internal mental state	-	focus	-	external actions
practiced `unit' task	-	focus	-	whole job

Task Decomposition

Aims:

- describe the actions people do
- structure them within task subtask hierarchy
- describe order of subtasks

Variants:

- Hierarchical Task Analysis (HTA)
  - most common
- CTT (CNUCE, Pisa)
  - uses LOTOS temporal operators

Textual HTA description

Hierarchy description ...

0. in order to clean the house
   1. get the vacuum cleaner out
   2. get the appropriate attachment
   3. clean the rooms
      - 3.1. clean the hall
      - 3.2. clean the living rooms
      - 3.3. clean the bedrooms
   4. empty the dust bag
   5. put vacuum cleaner and attachments away

... and plans

Plan 0: do 1 - 2 - 3 - 5 in that order. when the dust bag gets full do 4

Plan 3: do any of 3.1, 3.2 or 3.3 in any order depending on which rooms need cleaning

N.B. only the plans denote order

Generating the hierarchy

1. get list of tasks
2. group tasks into higher level tasks
3. decompose lowest level tasks further

Stopping rules

- How do we know when to stop?
- Is “empty the dust bag” simple enough?
- Purpose: expand only relevant tasks
- Motor actions: lowest sensible level

Tasks as explanation

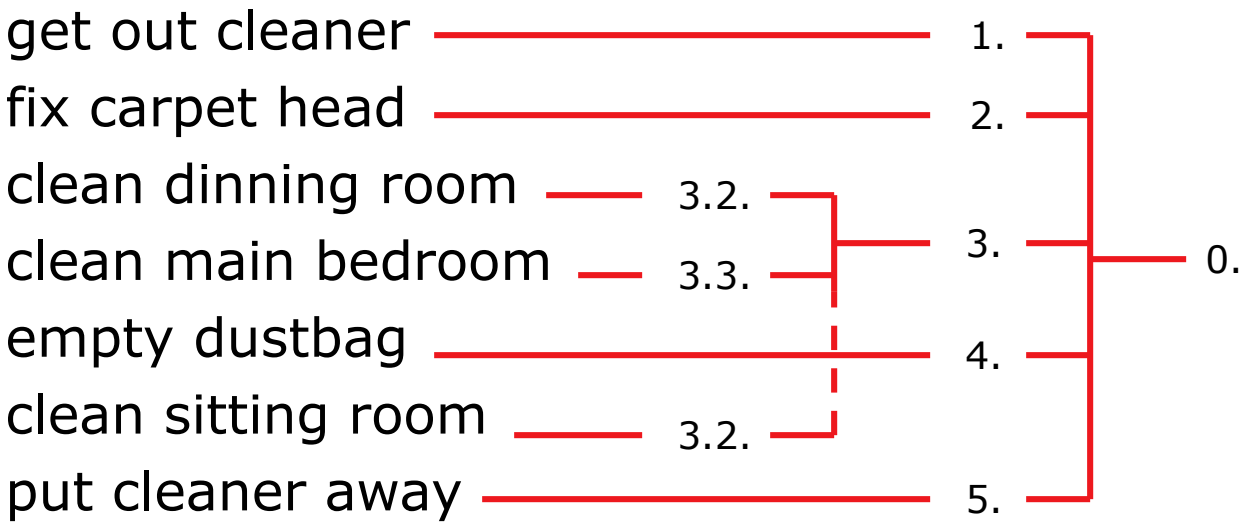
- imagine asking the user the question:
  what are you doing now?
- for the same action the answer may be:

  typing ctrl-B
  making a word bold
  emphasising a word
  editing a document
  writing a letter
  preparing a legal case

HTA as grammar

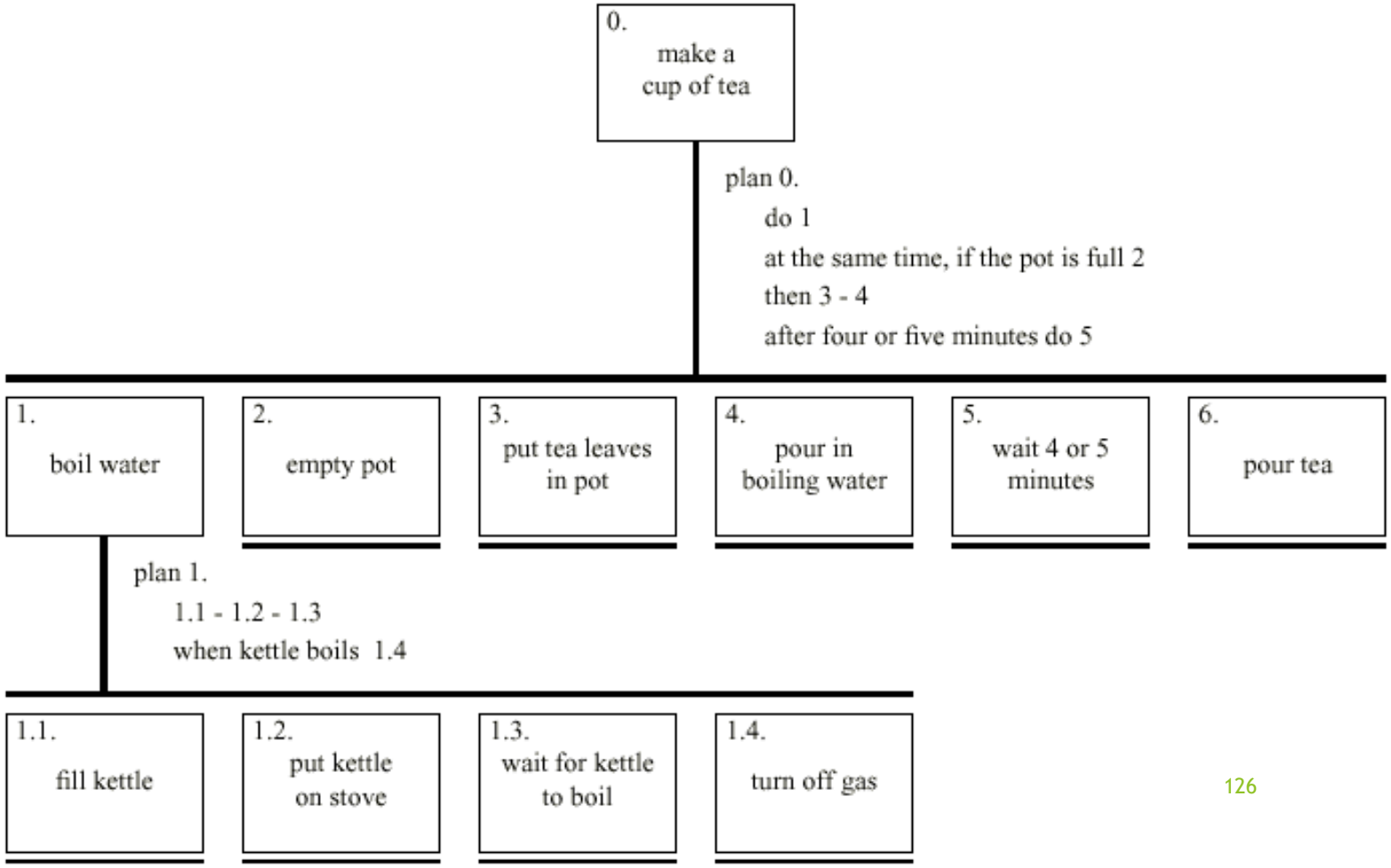
- can parse sentence into letters, nouns, noun phrase, etc.

parse scenario using HTA



0. in order to clean the house
    1. get the vacuum cleaner out
    2. get the appropriate attachment
    3. clean the rooms
        3.1. clean the hall
        3.2. clean the living rooms
        3.3. clean the bedrooms
    4. empty the dust bag
    5. put vacuum cleaner and attachments away

Diagrammatic HTA



Refining the description

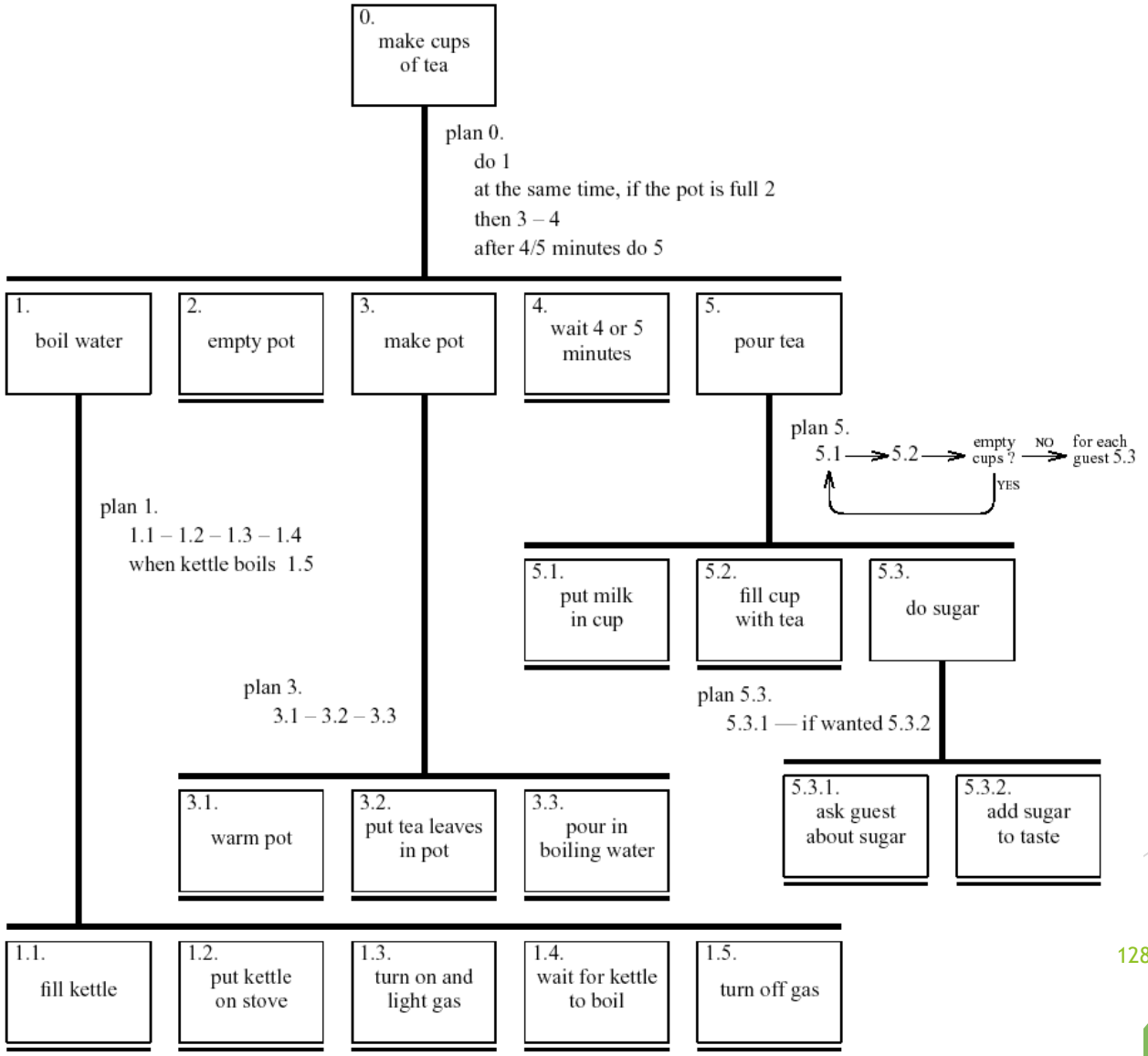
Given initial HTA (textual or diagram)

How to check / improve it?

Some heuristics:

paired actions	e.g., where is `turn on gas'
restructure	e.g., generate task `make pot'
balance	e.g., is `pour tea' simpler than making pot?
generalise	e.g., make one cup ….. or more

Refined HTA for making tea

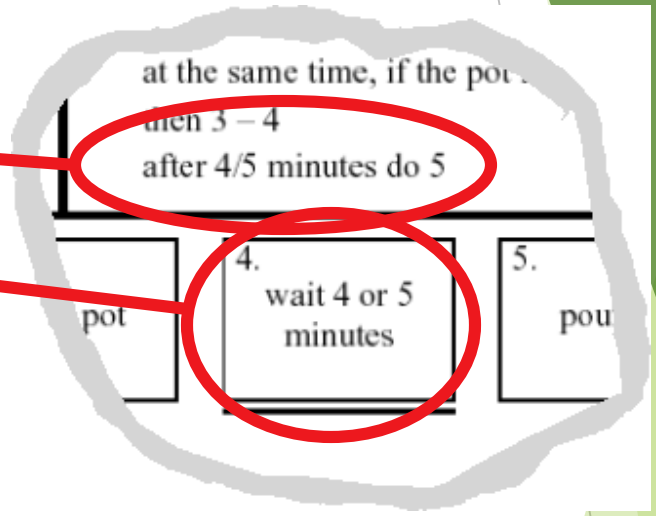


Types of plan

fixed sequence	-	1.1 then 1.2 then 1.3
optional tasks	-	if the pot is full 2
wait for events	-	when kettle boils 1.4
cycles	-	do 5.1 5.2 while there are still empty cups
time-sharing	-	do 1; at the same time ...
discretionary	-	do any of 3.1, 3.2 or 3.3 in any order
mixtures	-	most plans involve several of the above

wait...g ...

- is waiting part of a plan?
  … or a task?
- generally
  - task – if 'busy' wait
    - you are actively waiting
  - plan – if end of delay is the event
    - e.g. "when alarm rings", "when reply arrives"
- in this example …
  - perhaps a little redundant …
  - TA not an exact science



see chapter 19 for more on delays!

Knowledge Based Analyses

Focus on:

Objects – used in task

Actions – performed

+ Taxonomies –
represent levels of abstraction

Knowledge-Based Example …

motor controls

- steering *steering wheel, indicators*
- engine/speed
  - direct *ignition, accelerator, foot brake*
  - gearing *clutch, gear stick*
- lights
  - external *headlights, hazard lights*
  - internal *courtesy light*
- wash/wipe
  - wipers *front wipers, rear wipers*
  - washers *front washers, rear washers*
- heating *temperature control, air direction, fan, rear screen heater*
- parking *hand brake, door lock*
- radio *numerous!*

Task Description Hierarchy

Three types of branch point in taxonomy:

- XOR - normal taxonomy
  object in one and only one branch
- AND - object must be in both
  multiple classifications
- OR - weakest case
  can be in one, many or none

```
wash/wipe AND
 function XOR
     wipe     front wipers, rear wipers
     wash     front washers, rear washers
 position XOR
     front    front wipers, front washers
     rear     rear wipers, rear washers
```

Larger TDH example

```
kitchen item AND
/____shape XOR
/    |____dished  mixing bowl, casserole, saucepan,
/    |            soup bowl, glass
/    |____flat    plate, chopping board, frying pan
/____function OR
     {____preparation    mixing bowl, plate, chopping board
     {____cooking    frying pan, casserole, saucepan
     {____dining XOR
          |____for food   plate, soup bowl, casserole
          |____for drink  glass
```

N.B. '/ | {' used for branch types.

More on TDH

Uniqueness rule:

- can the diagram distinguish all objects?

e.g., plate is:

```
kitchen item/shape(flat)/function{preparation,dining(for food)}/
```

nothing else fits this description

Actions have taxonomy too:

```
kitchen job OR
|____ preparation  beating, mixing
|____ cooking frying, boiling, baking
|____ dining  pouring, eating, drinking
```

Abstraction and cuts

After producing detailed taxonomy
'cut' to yield abstract view

That is, ignore lower level nodes
e.g. cutting above shape and below dining, plate becomes:

```
kitchen item/function{preparation,dining}/
```

This is a term in Knowledge Representation Grammar (KRG)

These can be more complex:

e.g. 'beating in a mixing bowl' becomes:

```
kitchen job(preparation) using a
        kitchen item/function{preparation}/
```

Entity-Relationship Techniques

Focus on objects, actions and their relationships

Similar to OO analysis, but …

- includes non-computer entities
- emphasises domain understanding not implementation

Running example

‘Vera's Veggies’ – a market gardening firm
owner/manager: Vera Bradshaw
employees: Sam Gummage and Tony Peagreen
various tools including a tractor `Fergie‘
two fields and a glasshouse
new computer controlled irrigation system

Objects

Start with list of objects and classify them:

Concrete objects:

simple things: spade, plough, glasshouse

Actors:

human actors: Vera, Sam, Tony, the customers

what about the irrigation controller?

Composite objects:

sets: the team = Vera, Sam, Tony

tuples: tractor may be < Fergie, plough >

Attributes

To the objects add attributes:

Object Pump3 **simple** – irrigation pump

Attributes:

status: on/off/faulty

capacity: 100 litres/minute

N.B. need not be computationally complete

Actions

List actions and associate with each:

- agent – who performs the actions
- patient – which is changed by the action
- instrument – used to perform action

examples:

- Sam (*agent*) planted (*action*) the leeks (*patient*)
- Tony dug the field *with* the spade (*instrument*)

Actions (ctd)

implicit agents – read behind the words

`the field was ploughed' – *by whom?*

indirect agency – the real agent?

`*Vera* programmed the *controller* to irrigate the field'

messages – a special sort of action

`Vera *told* Sam to ... '

rôles – an agent acts in several rôles

Vera as *worker* or as *manager*

example – objects and actions

Object Sam **human actor**

Actions:

S1: drive tractor

S2: dig the carrots

Object Vera **human actor** - the proprietor

Actions: as worker

V1: plant marrow seed

V2: program irrigation controller

Actions: as manager

V3: tell Sam to dig the carrots

Object the men **composite**

Comprises: Sam, Tony

Object glasshouse **simple**

Attribute:

humidity: 0-100%

Object Irrigation Controller **non-human actor**

Actions:

IC1: turn on Pump1

IC2: turn on Pump2

IC3: turn on Pump3

Object Marrow **simple**

Actions:

M1: germinate

M2: grow

Events

… when something happens

- performance of action
  ‘Sam dug the carrots’
- spontaneous events
  ‘the marrow seed germinated’
  ‘the humidity drops below 25%’
- timed events
  ‘at midnight the controller turns on’

Relationships

- object-object
  - social - Sam is subordinate to Vera
  - spatial - pump 3 is in the glasshouse
- action-object
  - agent (listed with object)
  - patient and instrument
- actions and events
  - temporal and causal
    ‘Sam digs the carrots because Vera told him’
- temporal relations
  - use HTA or dialogue notations.
  - show task sequence (normal HTA)
  - show object lifecycle

example – events and relations

Events:

Ev1: humidity drops below 25%

Ev2: midnight

Relations: object-object

location (Pump3, glasshouse)

location (Pump1, Parker's Patch)

Relations: action-object

patient (V3, Sam)

- Vera tells *Sam* to dig

patient (S2, the carrots)

- Sam digs the *carrots* ...

instrument (S2, spade)

- ... *with* the spade

Relations: action-event

before (V1, M1)

- the marrow must be sown *before* it can germinate

triggers (Ev1, IC3)

- *when* humidity drops below 25%, the controller turns on pump 3

causes (V2, IC1)

- the controller turns on the pump *because* Vera programmed it

Sources of Information

Documentation

- N.B. manuals say what is *supposed* to happen
  but, good for key words and prompting interviews

Observation

- formal/informal, laboratory/field (see Chapter 9)

Interviews

- the expert: manager or worker? (ask both!)

Early analysis

Extraction from transcripts

- list nouns (objects) and verbs (actions)
- beware technical language and context:
  `the rain poured' vs. `I poured the tea'

Sorting and classifying

- grouping or arranging words on cards
- ranking objects/actions for task relevance (see ch. 9)
- use commercial outliner

Iterative process:

data sources ⌦ analysis

… but costly, so use cheap sources where available

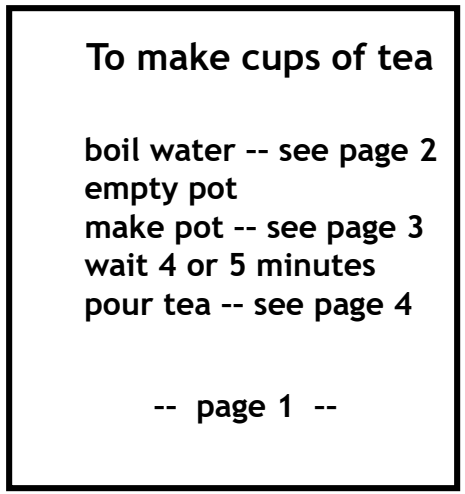
Uses – manuals & documentation

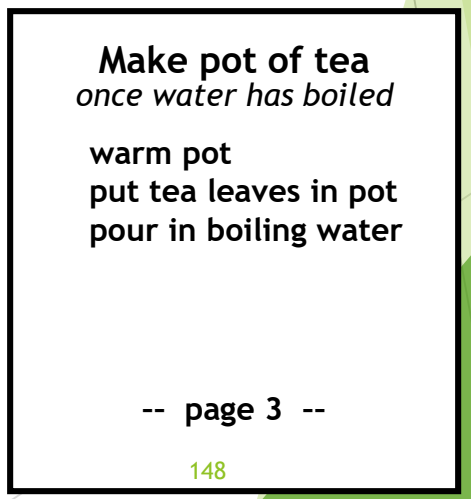
Conceptual Manual

- from knowledge or entity-relations based analysis
- good for open ended tasks

Procedural 'How to do it' Manual

- from HTA description
- good for novices
- assumes all tasks known





Uses – requirements & design

Requirements capture and systems design

- lifts focus from system to use
- suggests candidates for automation
- uncovers user's conceptual model

Detailed interface design

- taxonomies suggest menu layout
- object/action lists suggest interface objects
- task frequency guides default choices
- existing task sequences guide dialogue design

NOTE. task analysis is never complete

- rigid task based design $\Rightarrow$ inflexible system