Interface Implementation

HCI Lecture 11

David Aspinall

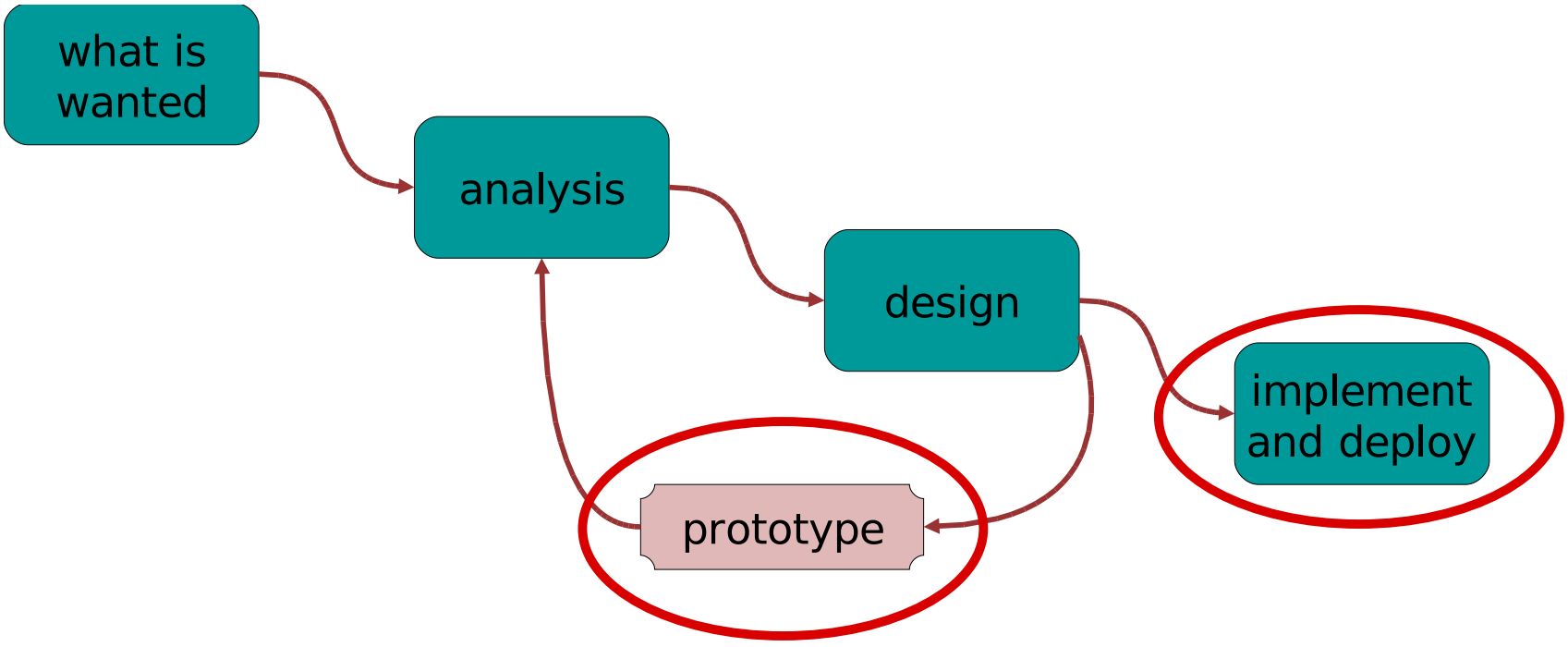
Informatics, University of Edinburgh

26th October 2007

Outline

Outline

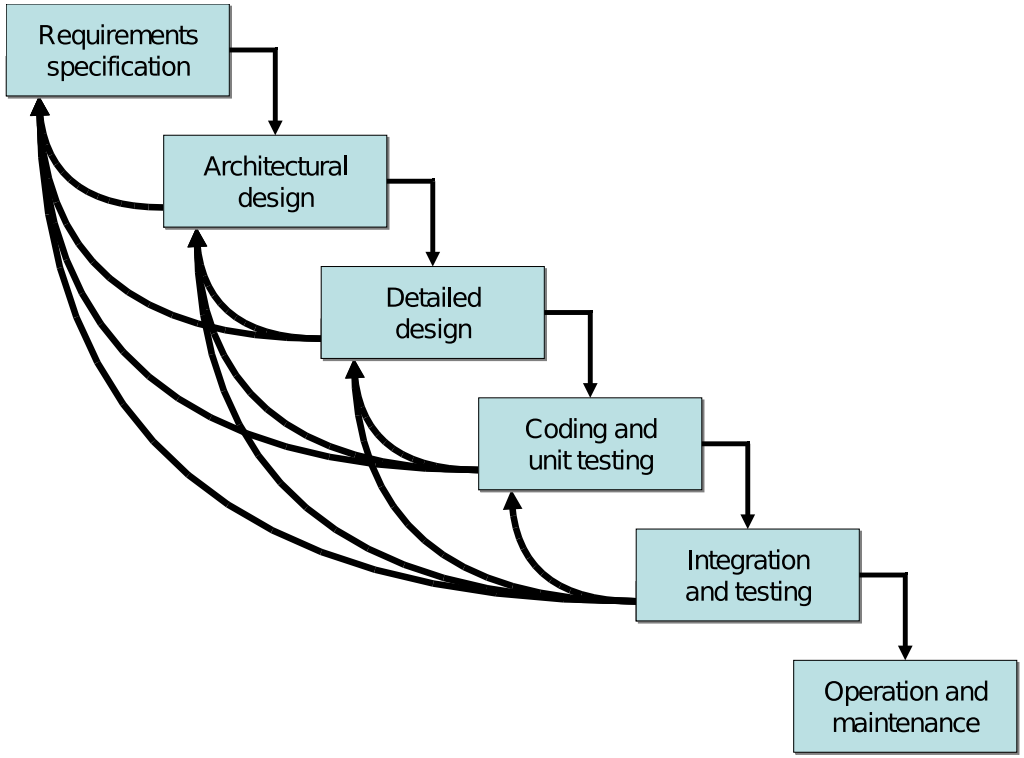
Focus on Implementation



- HCI in the software process:
  Usability Engineering and **Design Rationale**
- Programming interfaces:
  Implementation Support

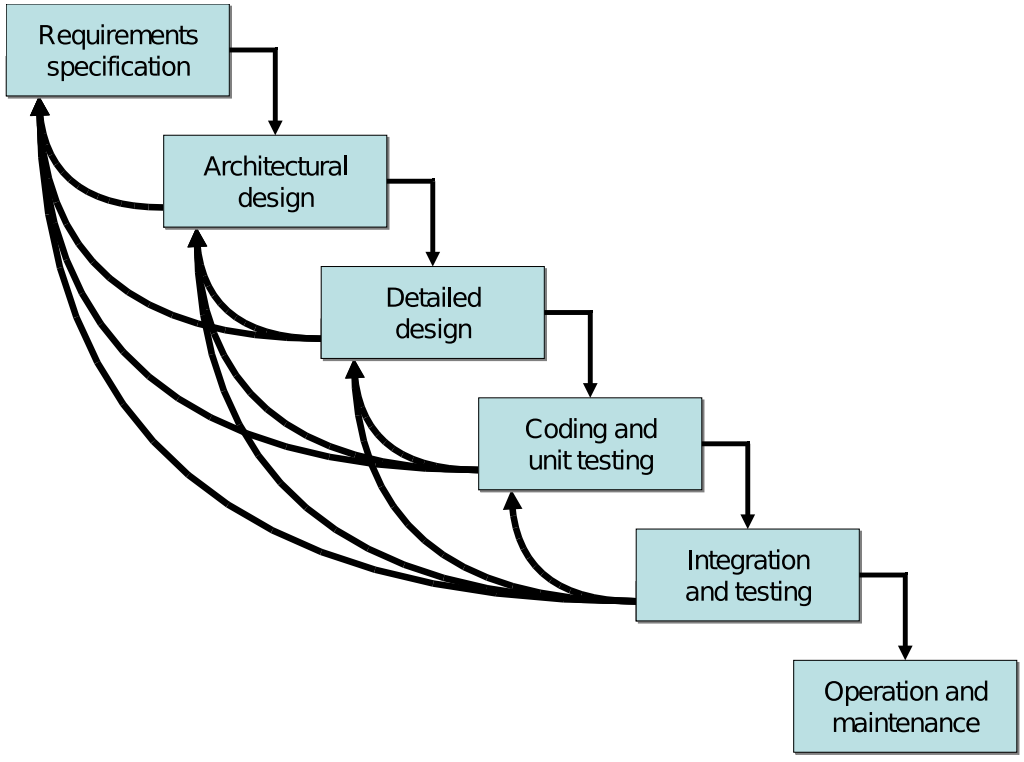
Outline

HCI in Software Engineering



waterfall model with feedback

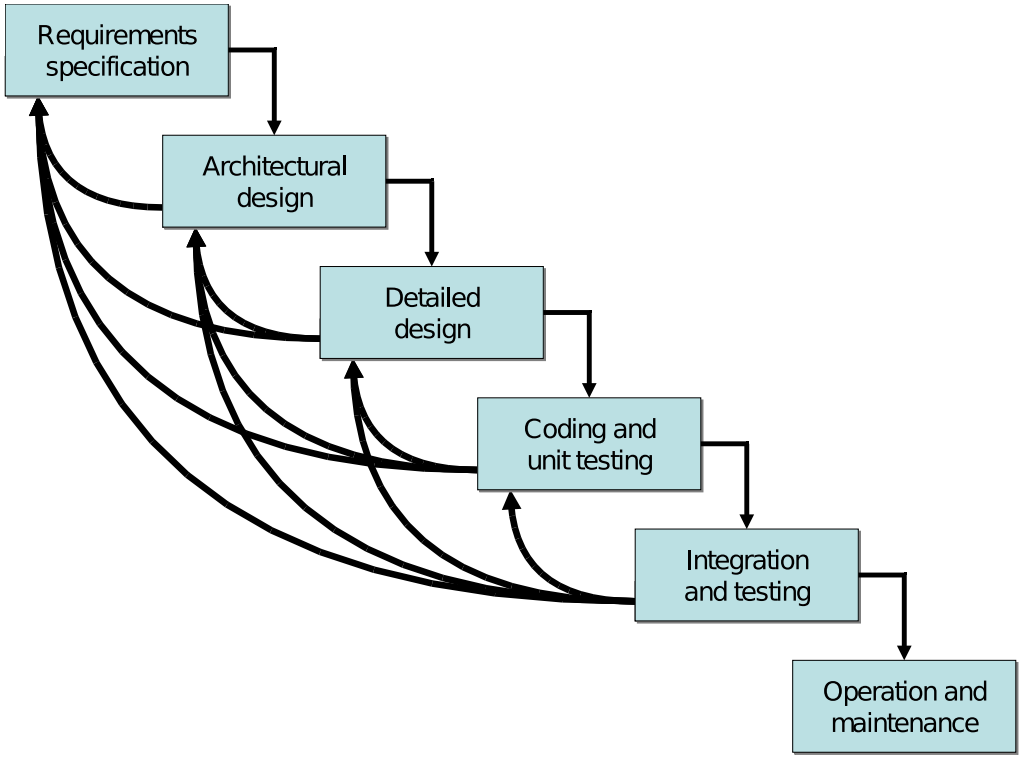
HCI in Software Engineering



waterfall model with feedback

- traditional process models require modification. . .

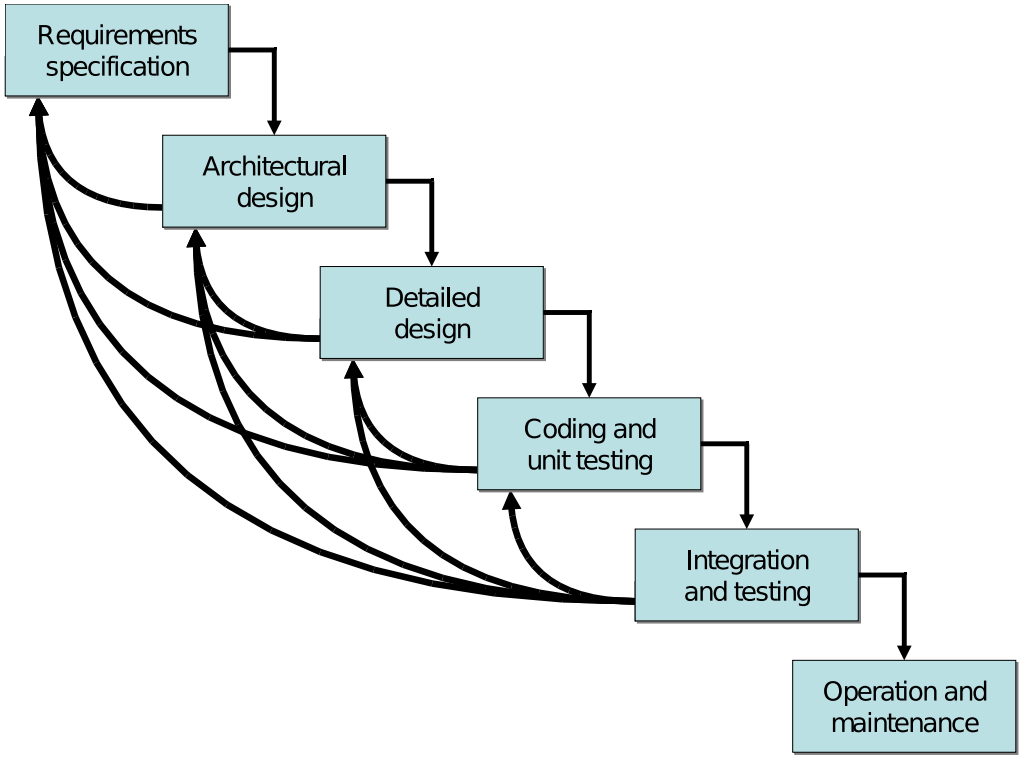
HCI in Software Engineering



waterfall model with feedback

- traditional process models require modification. . .
- user participation
  - during design
  - during evaluation

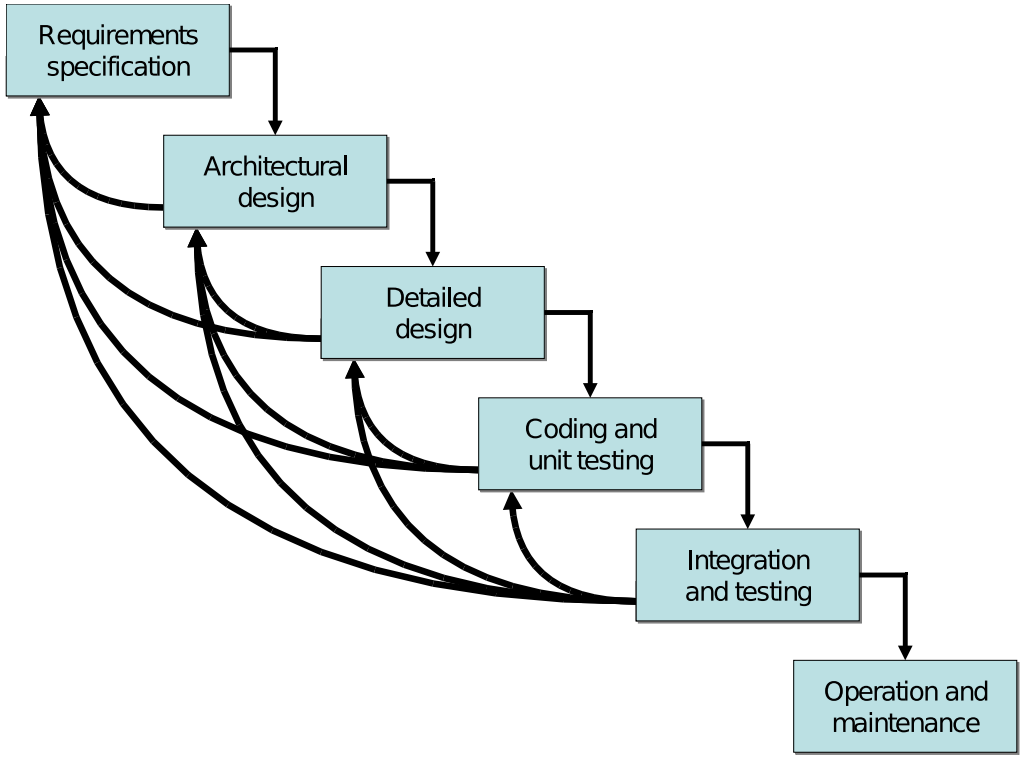
HCI in Software Engineering



waterfall model with feedback

- traditional process models require modification. . .
- user participation
  - during design
  - during evaluation
- usability evaluation

HCI in Software Engineering



waterfall model with feedback

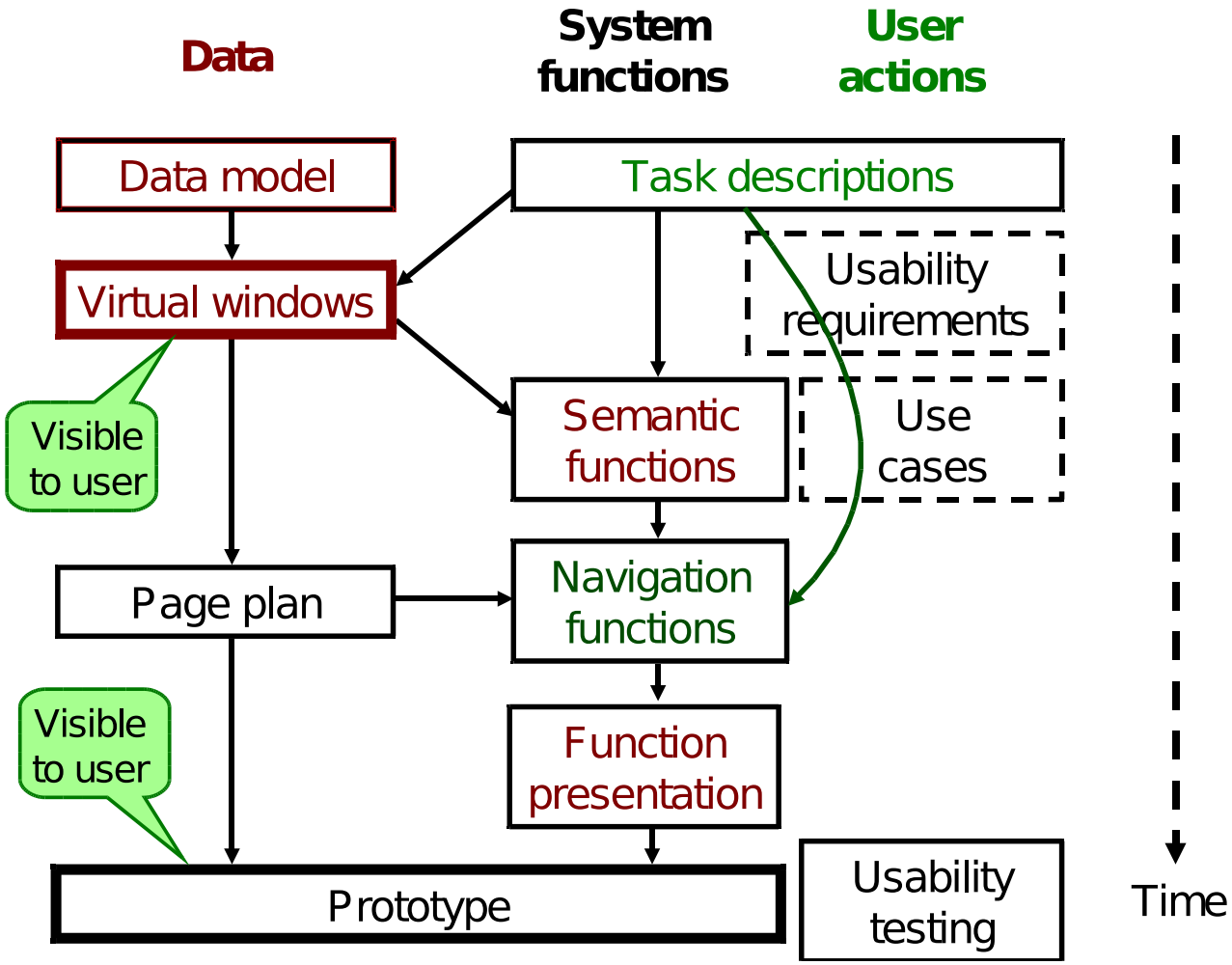
- traditional process models require modification. . .
- user participation
  - during design
  - during evaluation
- usability evaluation
- should design interface early, not as a bolt-on!
- ⟶ UI in process

HCI-oriented processes

- Lauesen and Harning (2001) describe a process called *Virtual Windows* which connects tasks, data models and UI design.

HCI-oriented processes

- Lauesen and Harning (2001) describe a process called *Virtual Windows* which connects tasks, data models and UI design.



Outline

Usability Engineering

- Ultimate usability test: measure user experience

Usability Engineering

- Ultimate usability test: measure user experience
- Usability measures made explicit as requirements
- Usability specification:
  - usability attribute/principle
  - measuring concept
  - measuring method
  - now level/ worst case/ planned level/ best case
- Cf. ISO 9241 metrics in Lecture 10

Usability Engineering

- Ultimate usability test: measure user experience
- Usability measures made explicit as requirements
- Usability specification:
  - usability attribute/principle
  - measuring concept
  - measuring method
  - now level/ worst case/ planned level/ best case
- Cf. ISO 9241 metrics in Lecture 10
- Problems:
  - usability spec requires level of detail that may not be possible early in design
  - satisfying a usability specification does not necessarily satisfy usability

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

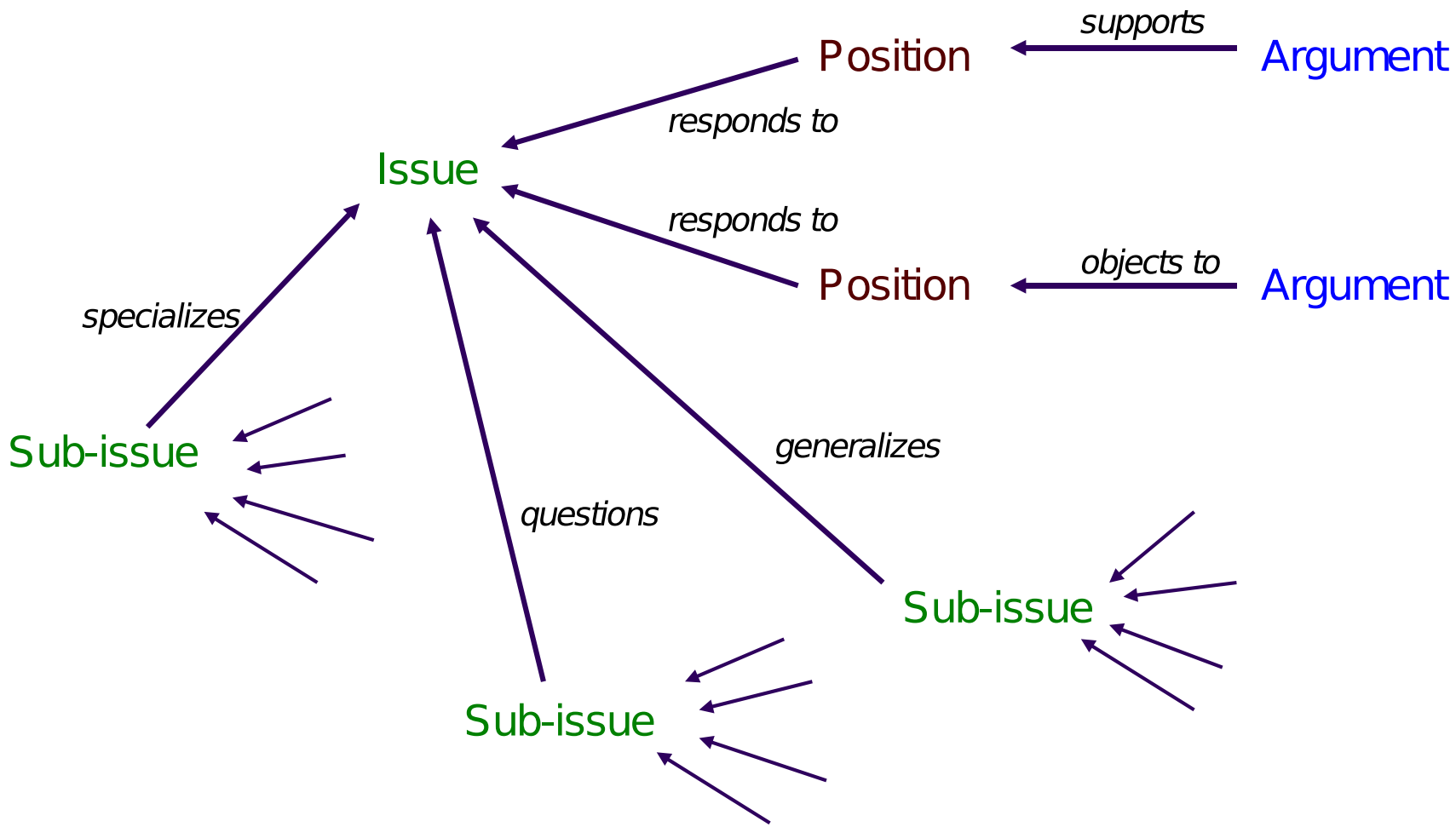
Outline

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
- Types of DR:
  - **Process-oriented** preserves order of deliberation and decision-making
  - **Structure-oriented** emphasizes post hoc structuring of considered design alternatives
- Examples:
  - Issue-based information system (IBIS)
  - Design space analysis
  - Psychological design rationale

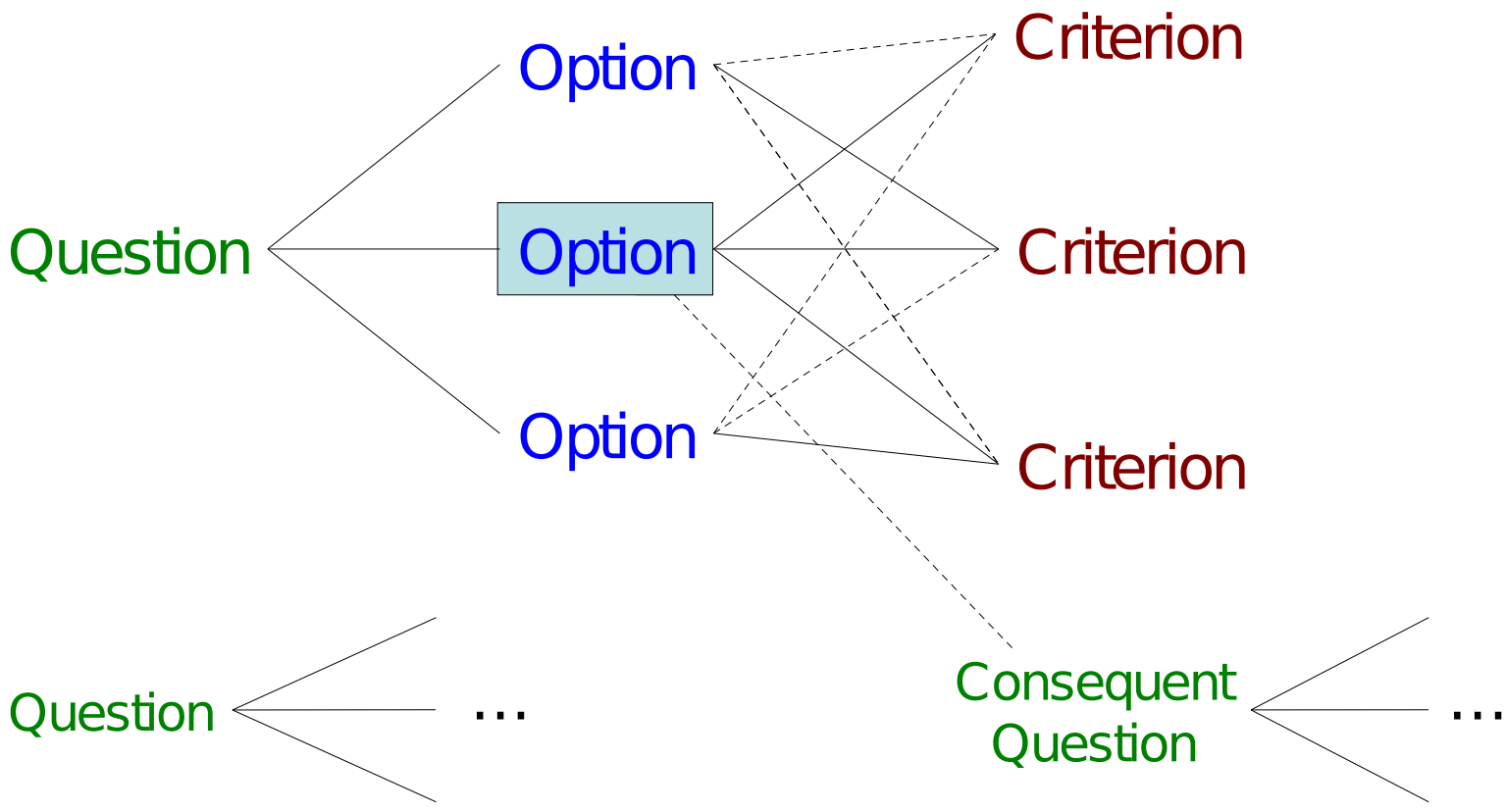
Issue-based Information System

- Process-oriented; main elements are:
  - *issues*: hierarchical structure with root
  - *positions*: potential resolutions
  - *arguments*: modify relationship between above



Design Space Analysis

- Structure-oriented: **QOC** hierarchy
  - *questions*: major issues of a design
  - *options*: alternative answers
  - *criteria*: means to assess options



Psychological Design Rationale

- Supports the **task-artefact** cycle in which user tasks are affected by the systems they use

Psychological Design Rationale

- Supports the **task-artefact** cycle in which user tasks are affected by the systems they use
- Consequences of design for users made explicit
- Method:
  - designers identify tasks system will support
  - scenarios are suggested to test task
  - users are observed on system

Psychological Design Rationale

- Supports the **task-artefact** cycle in which user tasks are affected by the systems they use
- Consequences of design for users made explicit
- Method:
  - 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 used to improve next iteration

Outline

Programming the Interface

- How does HCI affect the programmer?
- Advances in coding have elevated programming
  - hardware specific $\Longrightarrow$ interaction-technique specific

Programming the Interface

- How does HCI affect the programmer?
- Advances in coding have elevated programming
  - hardware specific $\Longrightarrow$ interaction-technique specific
- Layers of development tools
  - windowing systems
  - interaction toolkits
  - user interface management systems (UIMS)

Programming the Interface

- How does HCI affect the programmer?
- Advances in coding have elevated programming
  - hardware specific $\Longrightarrow$ interaction-technique specific
- Layers of development tools
  - windowing systems
  - interaction toolkits
  - user interface management systems (UIMS)
- Application architectures
  - Model-View-Controller (MVC)
  - Presentation-Abstraction-Control (PAC)

Programming the Interface

- How does HCI affect the programmer?
- Advances in coding have elevated programming
  - hardware specific $\Longrightarrow$ interaction-technique specific
- Layers of development tools
  - windowing systems
  - interaction toolkits
  - user interface management systems (UIMS)
- Application architectures
  - Model-View-Controller (MVC)
  - Presentation-Abstraction-Control (PAC)
- Body of programming techniques
  - concurrency management

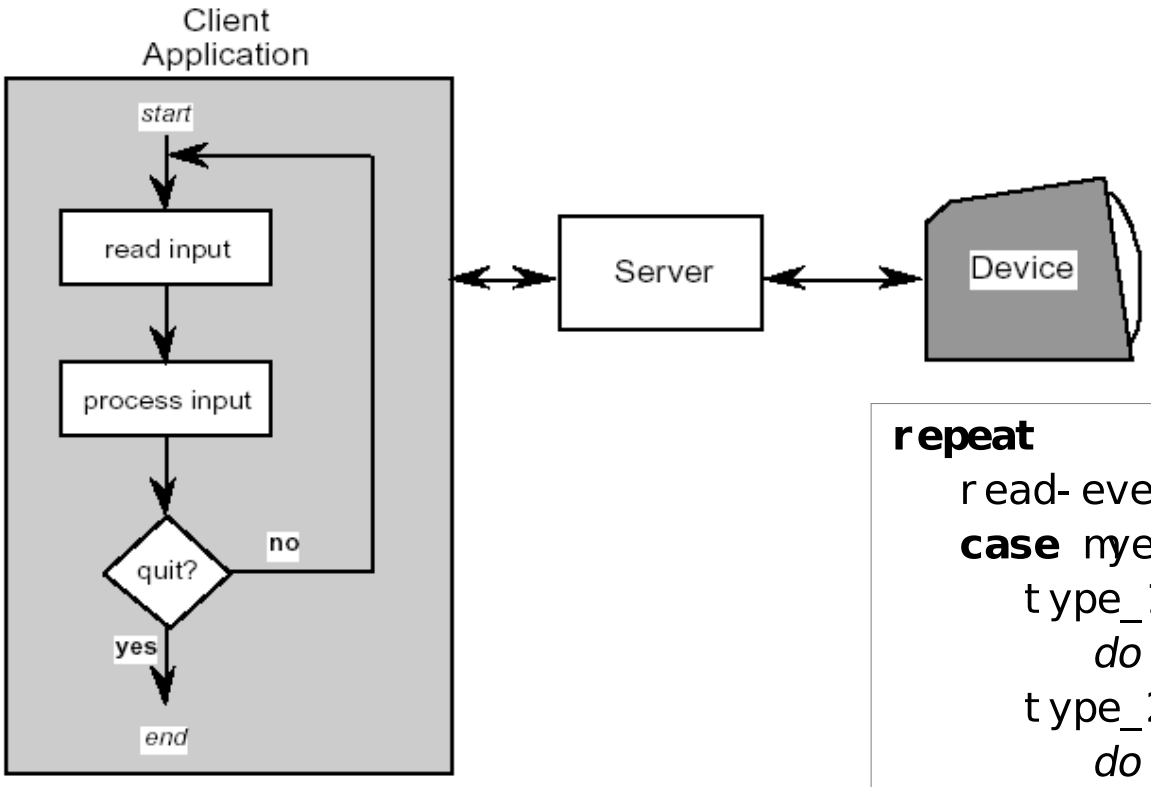
Windowing systems

- Role: mediate between devices and applications
  - “multiplex” I/O devices to allow multiple applications
  - *device indepedence* on top of *imaging model*

Windowing systems

- Role: mediate between devices and applications
  - “multiplex” I/O devices to allow multiple applications
  - *device indepedence* on top of *imaging model*
- Three possible software architectures:
  - each application manages all processes
    - everyone worries about synchronization
    - reduces portability of applications
  - management role within kernel of operating system
    - applications tied to operating system
  - management role as separate application
    - maximum portability
    - client-server, e.g. X Windows

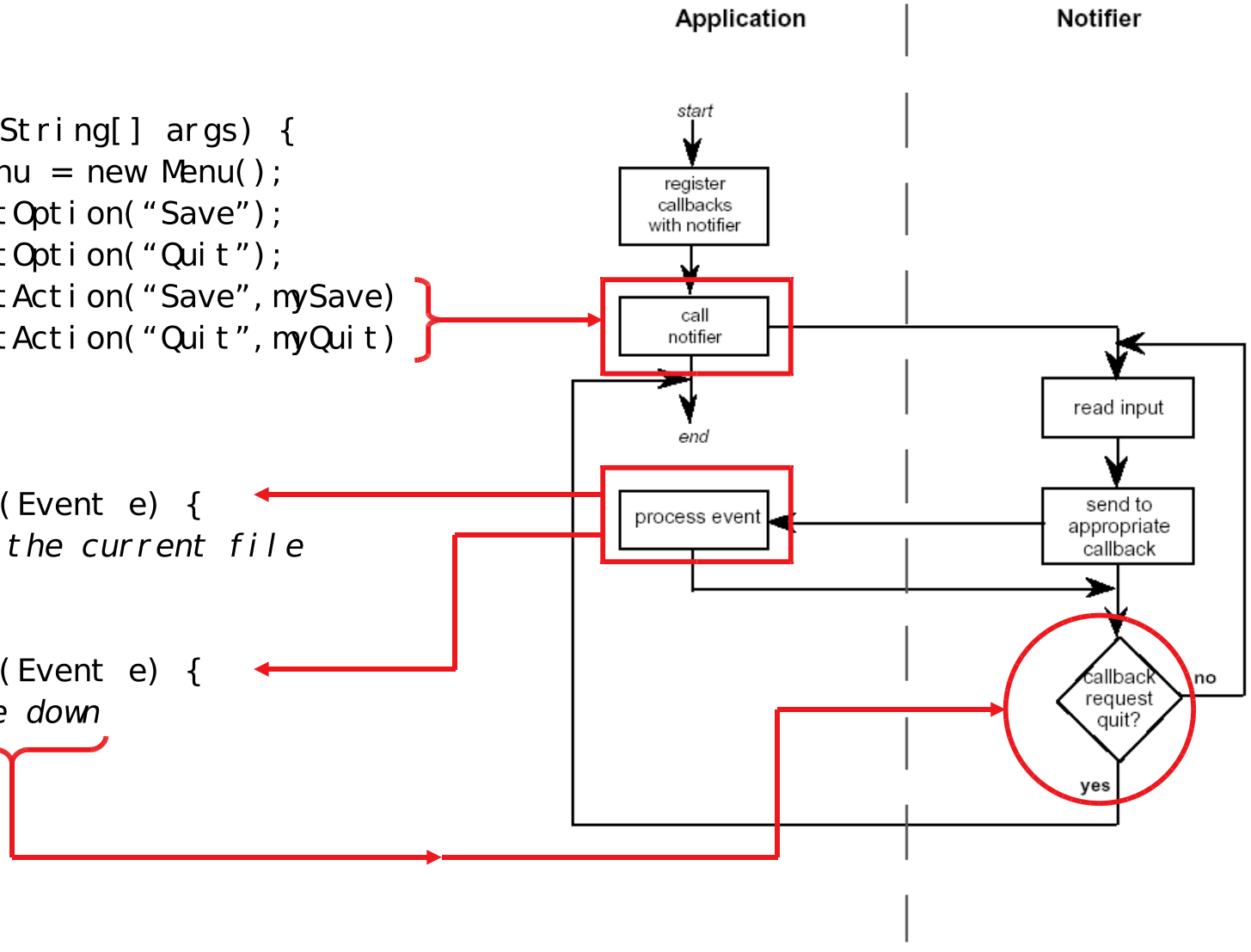
Application architecture: read-eval




```
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
```

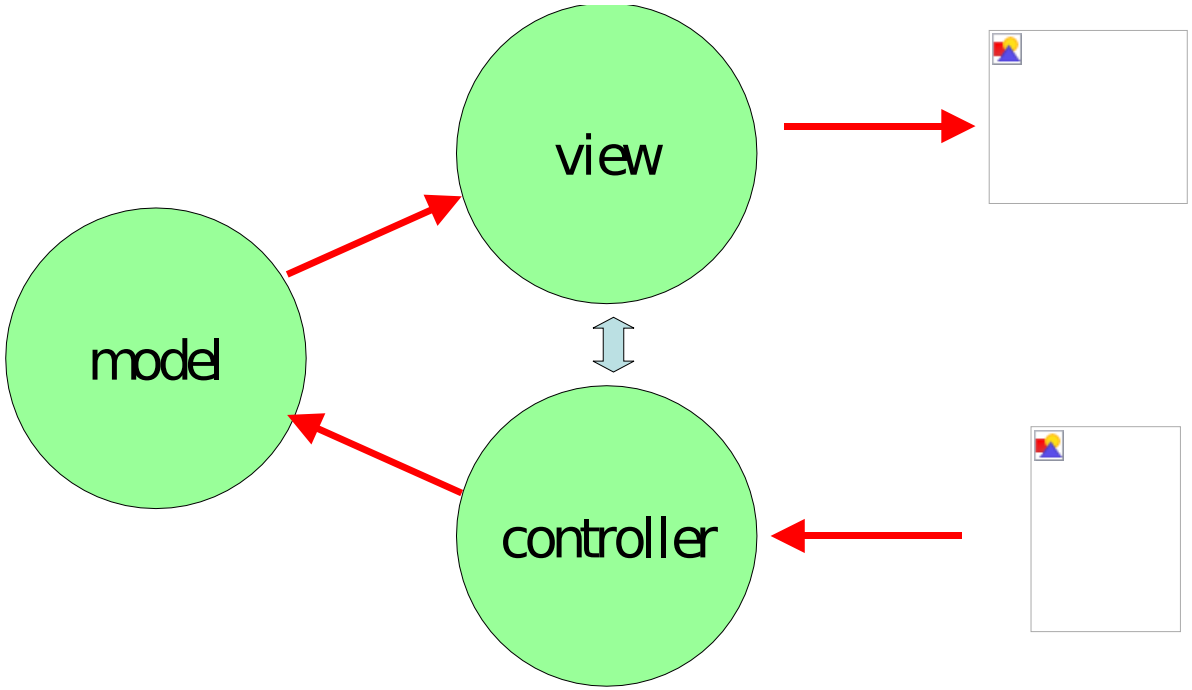
Application architecture: notification based

```
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
}
```



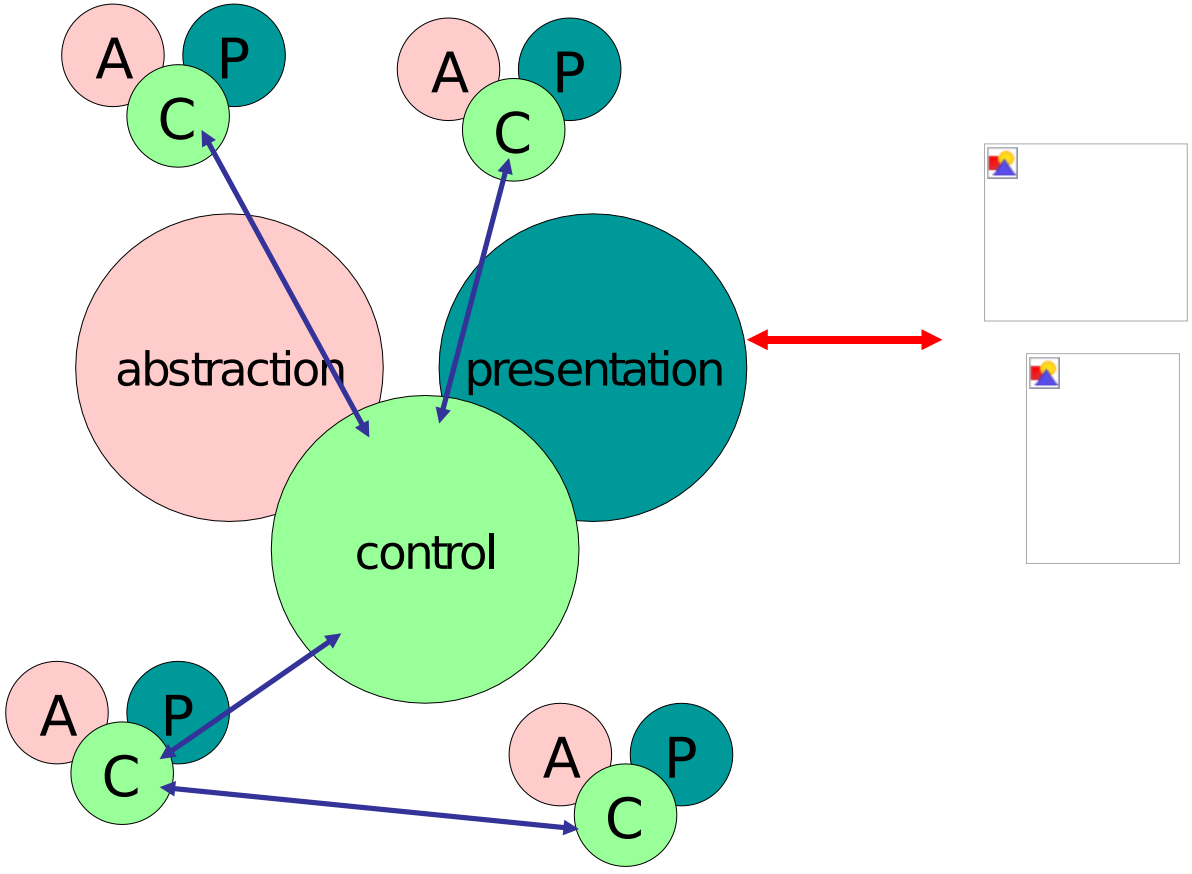
MVC: Model-View-Controller

- MVC highly influential design pattern used in Smalltalk (1980)



PAC: Presentation-Abstraction-Control

- Coutaz (1987) introduced PAC, a generalisation of MVC:



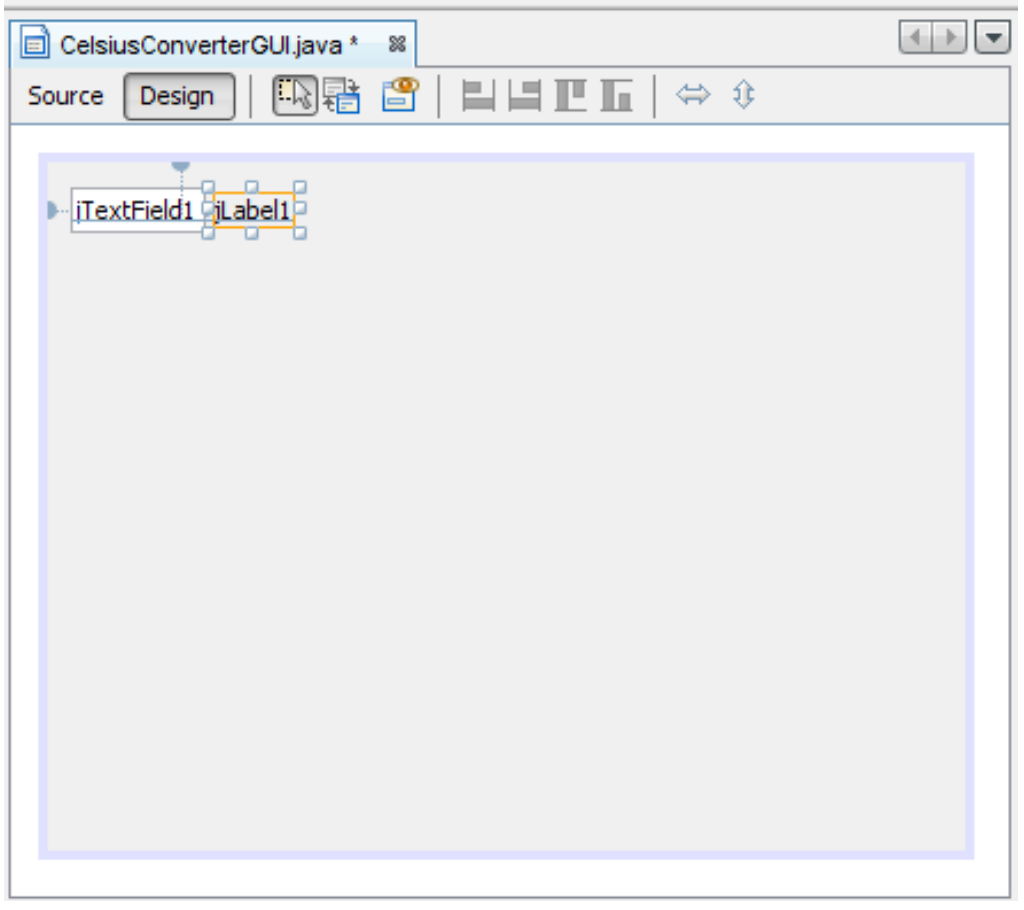
Graphical specification

- Trend in dialogue control:
  - internal control (e.g. read-eval loop)
  - external control (e.g. UIMS)
  - presentation control (e.g. **graphical specification**)

Graphical specification

- Trend in dialogue control:
  - internal control (e.g. read-eval loop)
  - external control (e.g. UIMS)
  - presentation control (e.g. **graphical specification**)

- coder draws components
- sets actions with script or links to program
- Issues: focus on one window, hard to “see” paths through system
- Examples: Visual Basic, Flash, DreamWeaver, **NetBeans Interface Builder**



Multi-threading in practice

Multithreaded GUI toolkits seem to be one of the Failed Dreams [of Computer Science].

Graham Hamilton, Sun VP

http://weblogs.java.net/blog/kgh/archive/2004/10/

- Multi-threading desirable; yet nearly all GUI toolkits use **single-threaded subsystem**, e.g. an *event dispatch thread* as in Swing. Why?

Multi-threading in practice

> *Multithreaded GUI toolkits seem to be one of the Failed Dreams [of Computer Science].*
>
> *Graham Hamilton, Sun VP*
>
> *http://weblogs.java.net/blog/kgh/archive/2004/10/*

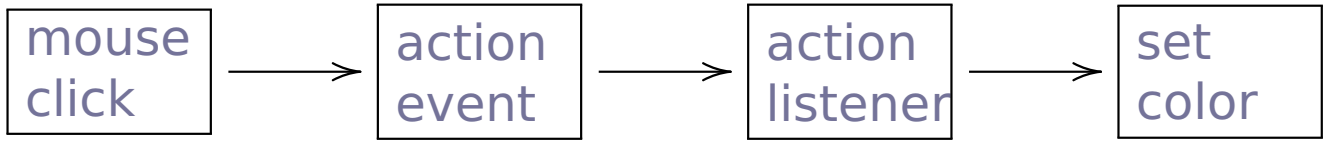
- Multi-threading desirable; yet nearly all GUI toolkits use **single-threaded subsystem**, e.g. an *event dispatch thread* as in Swing. Why?
- GUI components (visual, e.g. `JTable` and data, e.g. `TreeModel`) accessed only from event thread.
- A few exceptions, e.g:
  - adding and removing listeners
  - `SwingUtilities.isEventDispatchThread` to check if current thread is event thread

Example Event Listener

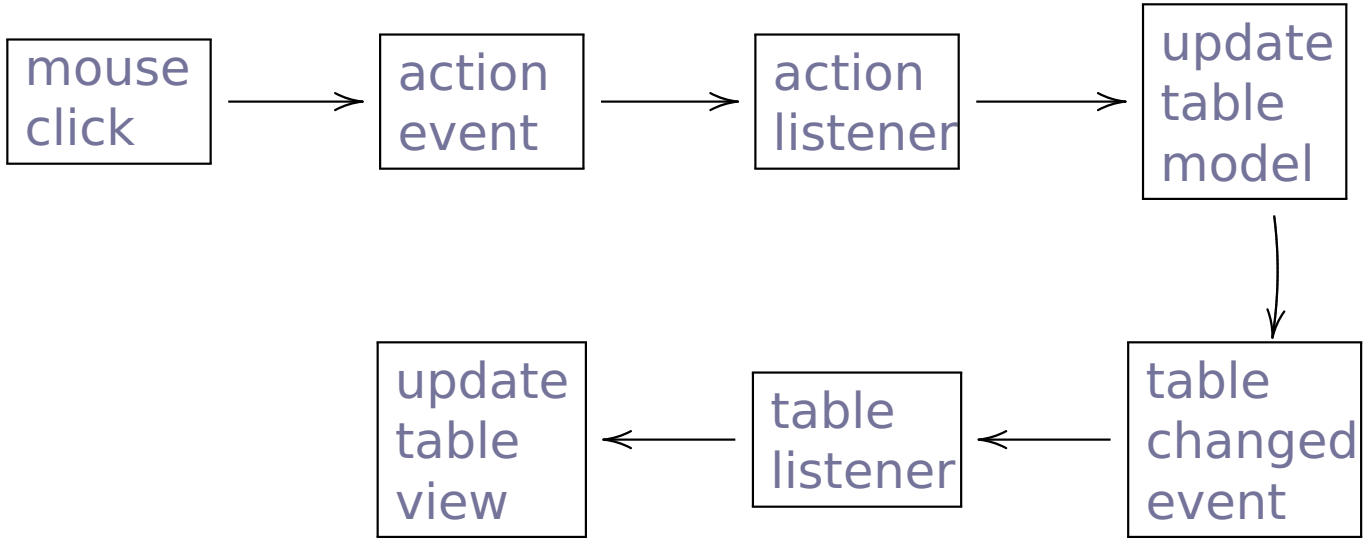
```
import javax.swing.*;   import java.awt.event.*;
import java.awt.Color;  import java.util.Random;

public class ColorButton extends JFrame {
  // A button with a listener to change its color
  public static void main(String[] args) {
      new ColorButton();
  }
  final Random random = new Random();
  final JButton button = new JButton("Change Color");
  ColorButton() {
      button.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
              button.setBackground(new Color(random.
                  nextInt()));
          }
       });
      add(button); pack(); setVisible(true);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
}
```

Flow of Events

- Simple event listener processes events like this:



- To use a view and data model (MVC), this:



- `fireXxx` methods used to indicate model change
- control stays in event thread

Advanced GUI architectures

- Update presentation from GUI thread:
  - `SwingUtilities.invokeLater`, schedules a task for execution in the event thread (callable anywhere)
  - `SwingUtilities.invokeAndWait`, schedules task in event thread and blocks (call from *non*-GUI thread)
- To keep GUI responsive, handle long-running tasks:
  - dispatch separate non-GUI threads to do work (e.g. using thread pool `Executor`)
- Need “thread-hopping”:
  - non-GUI thread queues GUI events to signal progress, completion
  - GUI thread handles cancellation event to kill non-GUI thread
- Managing data models:
  - *shared data model*: synchronisation needed
  - *thread-safe data models*: fine-grained concurrency; versioning
  - *split data model*: presentation-domain and application-domain models

References

Joëlle Coutaz.

PAC, on object oriented model for dialog design.

In *Interact'87*, 1987.

Soren Lauesen and Morten Borup Harning.

Virtual windows: Linking user tasks, datamodels, and interface design.

IEEE Software, pages 67–75, July/August 2001.

See also:

- Dix et al, Chapters 6 and 8.
- Jakob Nielsen's website for more on Usability Engineering: `http://www.useit.com`.
- Java Swing programming resources at `http://java.sun.com/docs/books/tutorial/uiswing/`