Software Evolution and Aspect-Oriented Programming

Belgian Symposium and Contact Day

Monday, 3 May 2004
Het Pand, Gent

Software Evolution and Aspect-Oriented Programming

Co-organised by

FWO Scientific Research Network on *Foundations of Sotware Evolution*

http://prog.vub.ac.be/FFSE

IWT GBOU Project on *Architectural Resources for the Restructuring and Integration of Business Applications* (ARRIBA)

http://progwww.vub.ac.be/

Today's Schedule

Morning

8:30 Opening and registration
9:00 Welcome and introduction
Tom Mens
Theo D'Hondt
9:45 Rule-based approaches to aspect-oriented programming
Maja D'Hondt

10:15 Coffee break
10:45 Using traits as aspects
Roel Wuyts
11:15 Conceptual code mining
Kim Mens
11:45 AOP in a graph-rewriting context
Alon Amsel
12:15 Lunch

Afternoon

14:00 Dynamic evolution of web services with AOP
Wim Vanderperren
14:30 Aspects in network-transparent distributed programming
Peter Van Roy
15:00 Aspectual contracts to manage invasive access
Bart De Win
15:30 Coffee break
16:00 Cross-cutting concerns in industrial C applications
Tom Tourwé
16:30 Aspect-oriented evolution of legacy software
Kris De Schutter
17:00 Conclusion and future plans
17:30 Reception

Software Evolution and Aspect-Oriented Programming

Tom Mens

Université de Mons-Hainaut
Service de Génie Logiciel
staff.umh.ac.be/Mens.Tom/

Separation of concerns

- Concern
  - Something the programmer should care about
  - "Those interests which pertain to the system development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders. Concerns can be logical or physiscal concepts, but they may also include system considerations such as performance, reliability, security, distribution, and evolvability."
    - *Recommended Practice for Architectural Description of Software-Intensive Systems*, IEEE Std 1471-2000. September 2000

Separation of concerns

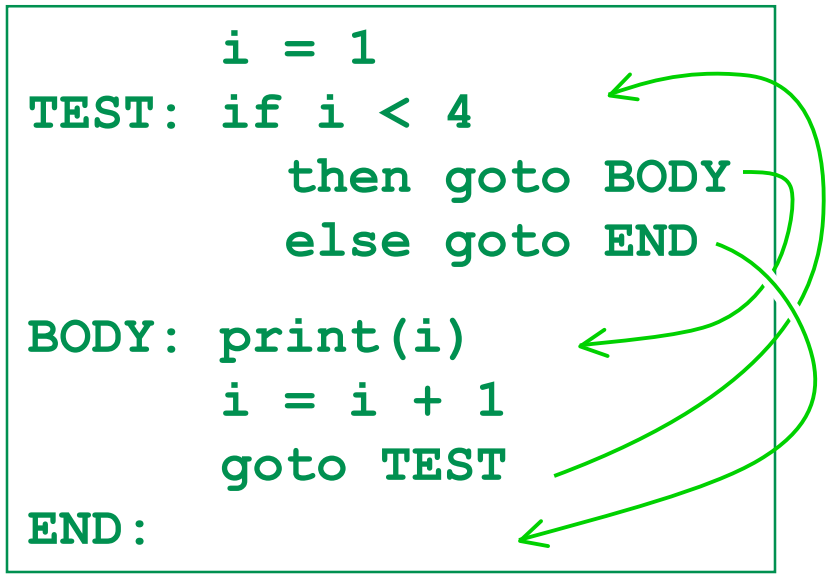
- Separation of concerns
  - Arguably the most important principle in software engineering
    - E. W. Dijkstra, *A Discipline of Programming*, Prentice Hall, 1976
  - Different concerns should be separated in the software
    - to cope with complexity
    - to achieve the required quality factors
      - adaptability, maintainability, extendibility, reusability, ...
    - to facilitate localising and fixing defects
    - to respond easier to market changes

Separation of concerns

- Implemented in different ways in different programming paradigms
  - Procedural programming
    - Separation of behaviour (procedures) and data (abstract data types)
    - Store different concerns in different procedures
  - Modular programming
    - Store different concerns in different modules
  - Object-oriented programming
    - Store different concerns in different objects/classes

Separation of concerns
Procedural programming

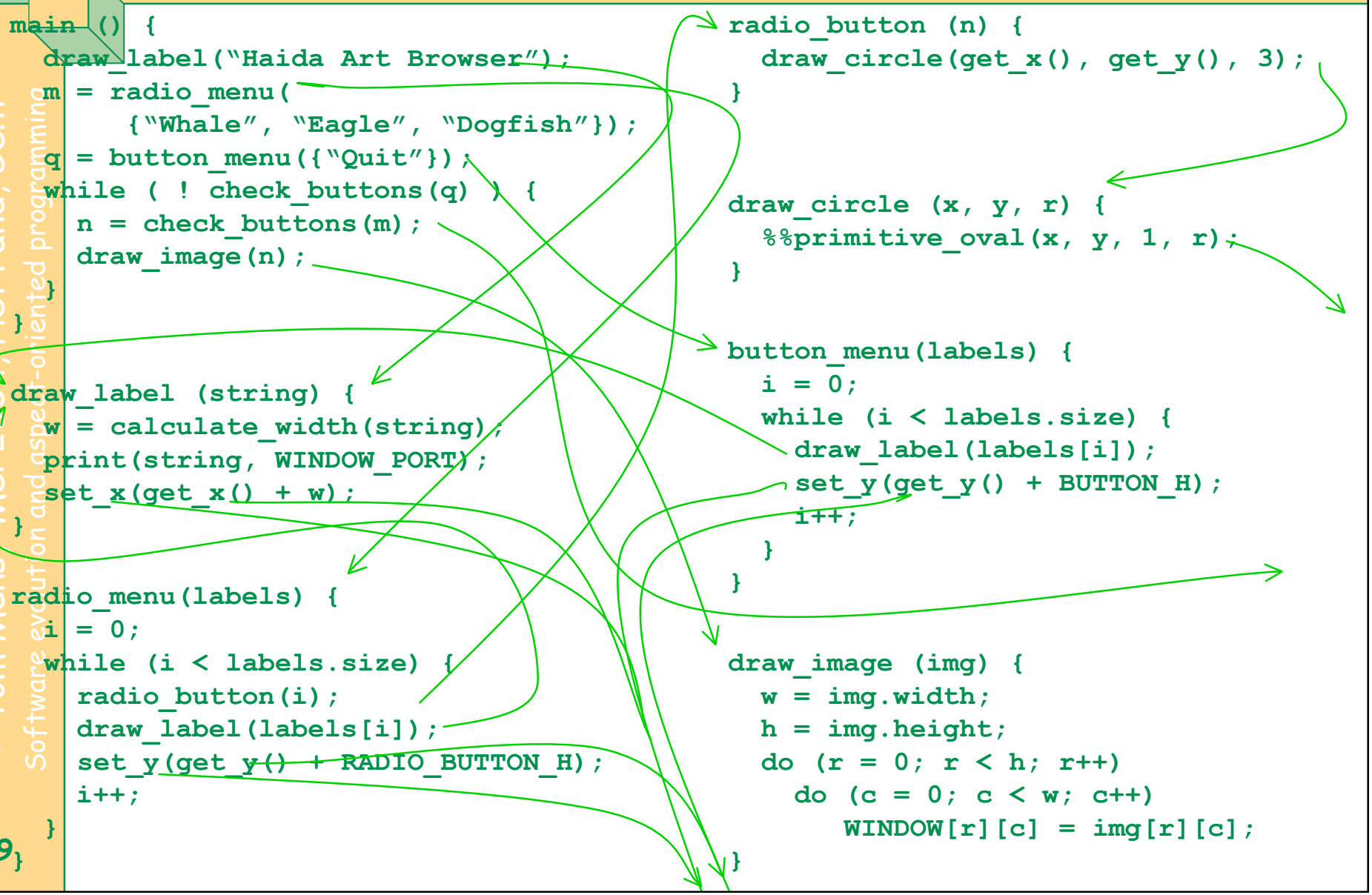
- Tangling from explicit gotos
- Recognized common control structures
  - capture in more explicit form
- Resulting code
  - more clear, easier to write, maintain, debug etc.



```
        i = 1
TEST:   if i < 4
          then goto BODY
          else goto END

BODY:   print(i)
        i = i + 1
        goto TEST
END:
```

```
i = 1
while (i < 4) {
  print(i)
  i = i + 1
}
```

But… still tangled



Group functionality…

```
main () {
  draw_label("Haida Art Browser");
  m = radio_menu(
        {"Whale", "Eagle", "Dogfish"});
  q = button_menu({"Quit"});
  while ( ! check_buttons(q) ) {
    n = check_buttons(m);
    draw_image(n);
  }
}

draw_label (string) {
  w = calculate_width(string);
  print(string, WINDOW_PORT);
  set_x(get_x() + w);
}

radio_menu(labels) {
  i = 0;
  while (i < labels.size) {
    radio_button(i);
    draw_label(labels[i]);
    set_y(get_y() + RADIO_BUTTON_H);
    i++;
  }
}

radio_button (n) {
  draw_circle(get_x(), get_y(), 3);
}

draw_circle (x, y, r) {
  %%primitive_oval(x, y, 1, r);
}

button_menu(labels) {
  i = 0;
  while (i < labels.size) {
    draw_label(labels[i]);
    set_y(get_y() + BUTTON_H);
    i++;
  }
}

draw_image (img) {
  w = img.width;
  h = img.height;
  do (r = 0; r < h; r++)
    do (c = 0; c < w; c++)
      WINDOW[r][c] = img[r][c];
}
```

Separation of concerns
Modular programming

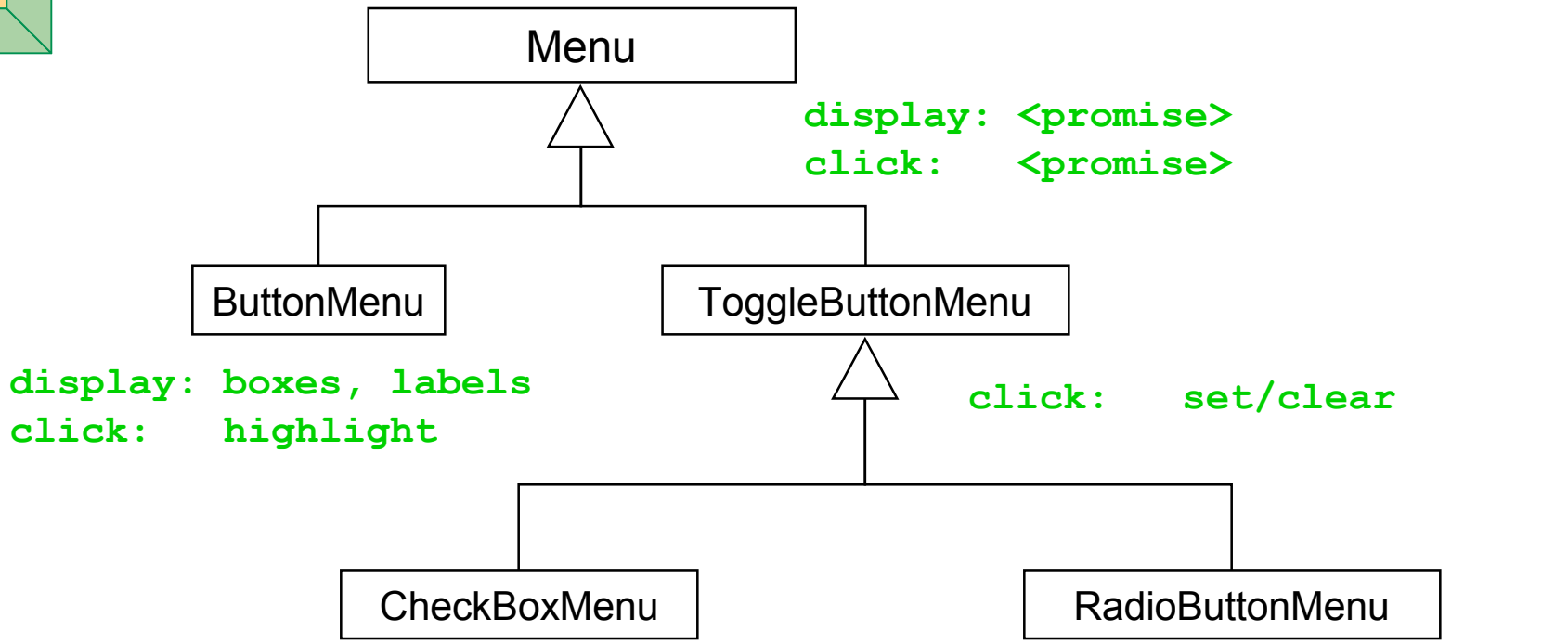
```
main () {
  draw_label("Haida Art Browser");
  m = radio_menu(
       {"Whale", "Eagle", "Dogfish"});
  q = button_menu({"Quit"});
  while ( ! check_buttons(q) ) {
    n = check_buttons(m);
    draw_image(n);
  }
}
```

```
draw_image (img) {
  w = img.width;
  h = img.height;
  do (r = 0; r < h; r++)
    do (c = 0; c < w; c++)
       WINDOW[r][c] = img[r][c];
}

draw_label (string) {
  w = calculate_width(string);
  print(string, WINDOW_PORT);
  set_x(get_x() + w);
}

draw_circle (x, y, r) {
  %%primitive_oval(x, y, 1, r);
}
```

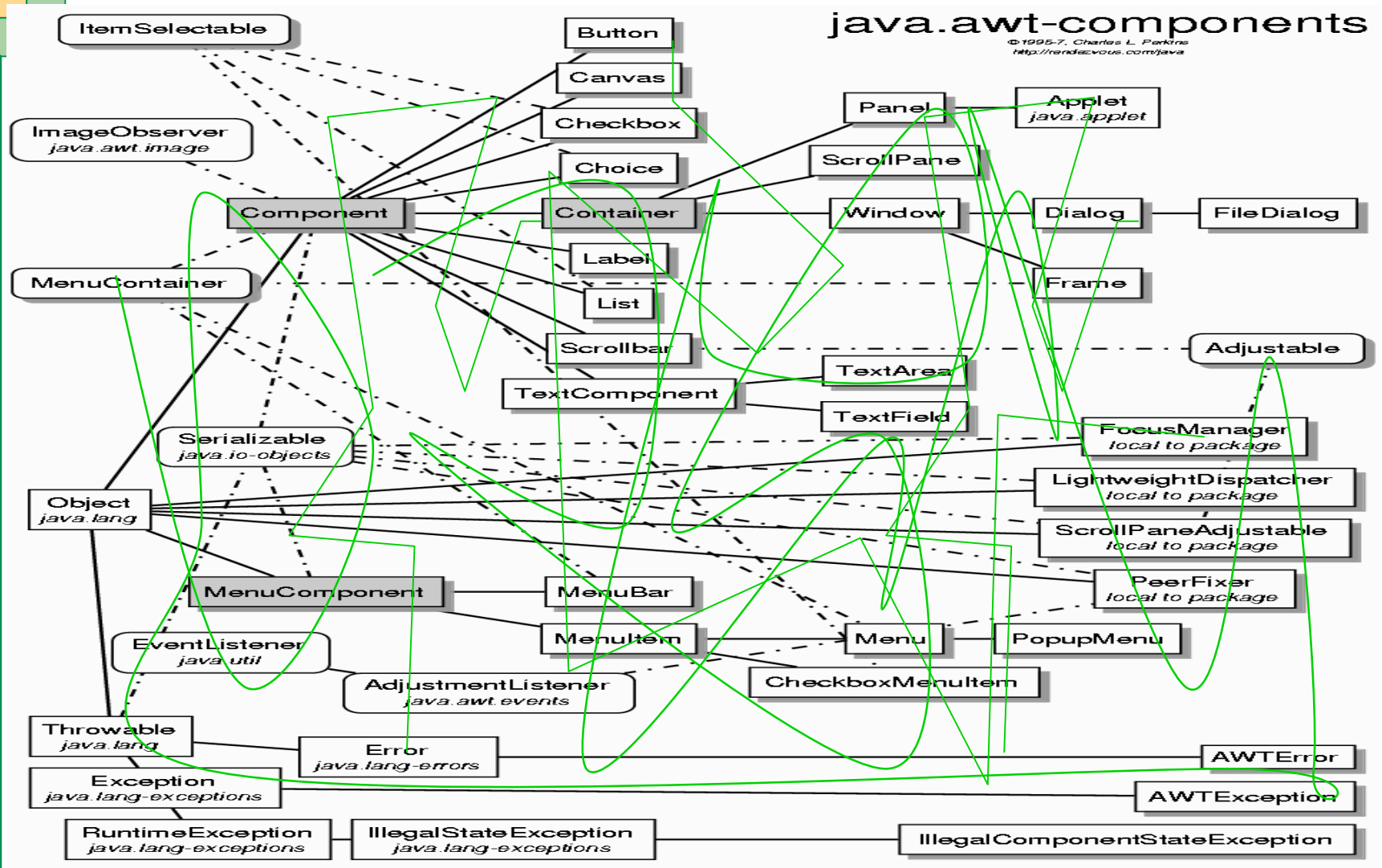
```
radio_menu(labels) {
  i = 0;
  while (i < labels.size) {
    radio_button(i);
    draw_label(labels[i]);
    set_y(get_y() + RADIO_BUTTON_H);
    i++;
  }
}

radio_button (n) {
  draw_circle(get_x(), get_y(), 3);
}

button_menu(labels) {
  i = 0;
  while (i < labels.size) {
    draw_label(labels[i]);
    set_y(get_y() + BUTTON_H);
    i++;
  }
}
```

But: variations on modules are incredibly complex

Separation of concerns
Object-oriented programming



But…


java.awt-components
©1995-7, Charles L. Perkins
http://rendezvous.com/java
ItemSelectable
Button
Canvas
Checkbox
Choice
ImageObserver
java.awt.image
Component
Container
Label
List
Scrollbar
TextComponent
TextArea
TextField
Panel
Applet
java.applet
ScrollPane
Window
Dialog
FileDialog
Frame
MenuContainer
Adjustable
Serializable
java.io-objects
FocusManager
local to package
LightweightDispatcher
local to package
ScrollPaneAdjustable
local to package
PeerFixer
local to package
Object
java.lang
MenuComponent
MenuBar
MenuItem
Menu
PopupMenu
CheckboxMenuItem
EventListener
java.util
AdjustmentListener
java.awt.events
Throwable
java.lang
Error
java.lang-errors
AWTError
Exception
java.lang-exceptions
AWTException
RuntimeException
java.lang-exceptions
IllegalStateException
java.lang-exceptions
IllegalComponentStateException

Croscutting concerns

- Problem
  - Implementation of some concerns remains spread across different objects
- Tyranny of the Dominant Decomposition
  - Tarr et al. *Multi-dimensional separation of concerns*, Proc. ICSE 1999, pp. 107-119, IEEE CS Press
  - There is no ideal way to decompose a problem
  - For any decomposition of the problem …
    - covering the core functionality concerns
  - … some concerns will cross-cut this decomposition
    - non-functional concerns, functional concerns
    - concerns that have been added later on

Crosscutting concerns and software evolution

- Requirements change all the time
  - corresponds to Lehman's 1st law of *Continuing Change*
    - "*An E-type program that is used must be continually adapted else it becomes progressively less satisfactory.*"
- Consequence
  - new concerns need to be added, and existing concerns modified, to make the software compatible again with the changed requirements
  - number of crosscutting concerns will increase over time, according to Lehman's 2nd law of *Increasing Complexity*
    - "*As a program is evolved its complexity increases unless work is done to maintain or reduce it.* "

Crosscutting concerns and aspect-oriented programming

- Aspects
  - an additional abstraction mechanism for (object-oriented) programming languages
  - allow to capture cross-cutting concerns in a localised way
    - reduces complexity

Software evolution and aspect-oriented programming

- Existing software evolution expertise on
  - migration of legacy code
  - reverse and re-engineering
  - refactoring and restructuring
- may be reused to
  - migrate/restructure existing software into aspect-oriented software
    - identify cross-cutting concerns and modularise them into aspects

Software evolution and aspect-oriented programming

Three important research goals

1. automatically identify crosscutting concerns
   based on pattern matching, clone detection, logic reasoning, formal concept analysis, ...
2. restructure legacy programs into aspect-oriented programs
3. deal with evolution of aspect-oriented programs
   co-evolution of base program and aspects