As We May Code
The art (and craft) of computer programming in the 21st century

Alan Dix
Lancaster University

www.hcibook.com/alan/papers/PPIG2008-as-we-may-code/

1968

Donald Knuth
The Art of Computer Programming

so what’s changed?
obvious things ...

languages ... what we teach

1968 – Fortran, Algol, Cobol
1978 – Fortran, maybe still Cobol
1988 – Pascal, ADA, but never ever Cobol
1998 – C++, Java, VB, but never ever Cobol
2008 – Java, Java, Java, what’s Cobol?

but how different are the courses?
variables, arrays, loops, functions, ...
Fortran by any other name ...
languages ... what is used

1968 – Fortran, Assembler, Cobol
1978 – mostly Cobol
1988 – VB, C, lots of Cobol
1998 – C++, VB, still lots of Cobol (Y2K!)
2008 – Java, C++,
Javascript, PHP, Python, Perl,
Actionscript, Processing,
not just Fortran in new clothes
... and betcha life still some Cobol

coding technology

15 years ago
emacs & listings

30 years ago
forms & cards

NOW
IDEs & no paper
programmers

C20
specialists

C21
anyone

expertise
strong

mathematics
weak

team / community of practice
organisational unit
global community

what is programming?
**what is programming?**

was

problem solving

Problem: find square root + Primitives: INTEGER, +, *, IF, GOTO, WHILE

FUNCTION f(x)

Solution:

```c
while ( abs(next-last) > 0.001 ) {
    last = next;
    next = ( x + x/last ) / 2;
}
```

**what is programming?**

now

British Library programming

wiki + google + tweak

find the solution

... on the net
### programming then and now

<table>
<thead>
<tr>
<th>C20</th>
<th>C21</th>
</tr>
</thead>
<tbody>
<tr>
<td>- problem solving</td>
<td>- recipe tinkering</td>
</tr>
<tr>
<td>- mathematics</td>
<td>- library science</td>
</tr>
<tr>
<td>- pre-planned</td>
<td>- emergent</td>
</tr>
<tr>
<td>- top down</td>
<td>- bottom up</td>
</tr>
<tr>
<td>- few well-understood problems</td>
<td>- many partially documented APIs</td>
</tr>
<tr>
<td>- specification</td>
<td>- exploration</td>
</tr>
<tr>
<td>- reuse rare and difficult ... the odd library</td>
<td>- all about libraries APIs and code fragments</td>
</tr>
</tbody>
</table>

---

### my code and the world

<table>
<thead>
<tr>
<th>C20</th>
<th>C21</th>
</tr>
</thead>
<tbody>
<tr>
<td>- algorithmics</td>
<td>- systemics</td>
</tr>
<tr>
<td>- my code</td>
<td>- plug-ins, services, ...</td>
</tr>
<tr>
<td>- single locus</td>
<td>- distributed</td>
</tr>
<tr>
<td>- input/output, batch, pipeline</td>
<td>- interactional / transactional</td>
</tr>
<tr>
<td>- procedural</td>
<td>- events and callbacks</td>
</tr>
<tr>
<td>- sub-classing</td>
<td>- mix-ins</td>
</tr>
<tr>
<td>- type inheritance what you are</td>
<td>- aggregate inheritance where you are – context</td>
</tr>
<tr>
<td>- SE – make systems like algorithms</td>
<td>- ? make algorithms like systems?</td>
</tr>
</tbody>
</table>

---

Wegner

Aspects

Janet@York
procedures -> events

linear

\[ x = 3 \times y \]
\[ z = x + y \]
\[ z = z \times z \]

loop

\[
\text{while } (x<3) \{
    y = y + x; \\
    x = x + 1; 
\}
\]

procedure

\[
\text{int } f(x) \{ ... \}
\]
\[
a = 42 \\
b = f(a) \\
a = 2 \times b;
\]

event / callback

\[
\text{int } f(x) \{ ... \}
\]
\[
\text{j = new Ajax()}
\]
\[
\text{j.setHandler(f)}
\]
\[
\text{j.invoke();}
\]

AJAX in action!

```javascript
function getSnipActions(div_id, snipid) {
    tellmeabout_snip(div_id, snipid, actionsCallback)
}

function actionsCallback(div_id, matches) {
    matchesObj = $(div_id);
    var html = formatter.format(matches);
    matchesObj.innerHTML = html;
}

function tellmeabout_snip(id, snipid, callback, includeNoActions) {
    if (arguments.length < 4) includeNoActions = false;
    doAjaxCall(TellmeaboutUrl, {snipid: snipid, op: 'actions'},
        tellmeaboutResponse, {id: id, snipid: snipid, callback: callback,
            includeNoActions: includeNoActions});
}
```
Wordpress plug-ins

```php
function save_status($new_status) {
    if ( $new_status === 'publish') {
        return 'private';
    } else {
        return $new_status;
    }
}

add_filter('status_save_pre', 'save_status');
```

bugs and debugging

**C20**
- algorithm
- single component
- unexpected input/event
- internal – logical failure in own new code
- lack of skill

**C21**
- structural
- feature interaction
- malicious attack
- external – error in infrastructure
- lack of knowledge

**ALWAYS**
- trial and error vs. systematic
- quick fix vs. understand problem
in summary...

things aren’t what they used to be
some better, some worse
but programming is different
and the way we think about it is different