Usability and Cognitive Models of Program Comprehension

Michael Hansen
Goldstone Lab Meeting
April 18th, 2012
“Many claims are made for the efficacy and utility of new approaches to software engineering – structured methodologies, new programming paradigms, new tools, and so on.

Evidence to support such claims is thin and such evidence, as there is, is largely anecdotal. Of proper scientific evidence there is remarkably little.”

- Frank Bott, 2001
Usability Claims

• “Methods where NbLinesOfCode is higher than 20 are hard to understand and maintain. Methods where NbLinesOfCode is higher than 40 are extremely complex and should be split in smaller methods (except if they are automatically generated).”

• “Our experience writing programs in Ur/Web suggests that the feature set we have chosen is more than sufficient for our application domain.”
  – Ur/Web meta-programming language overview (2010)

• “We found that in practice it was quite difficult to achieve consistency of usage-site type annotations, so that type errors were not uncommon.”
  – Scala overview (2006)

• “…it provides a natural and simple medium for the expression of a large class of algorithms.”
  – ACM-GAMM report on FORTRAN (1958)
Principles to Follow?

Stepanov, 2007

1. the code should be partitioned into functions;
2. every function should be most 20 lines of code;
3. functions should not depend on the global state but only on the arguments;
4. every function is either general or application specific, where general function is useful to other applications;
5. every function that could be made general – should be made general;
6. the interface to every function should be documented;
7. the global state should be documented by describing both semantics of individual variables and the global invariants.

Veldhuizen, 2007

1. For each component and use case combination, write code that uses the component to implement the use case. If the component cannot be adapted to the use case, then write the simplest possible implementation of the use case without the component.
2. For each component, count the tokens required to:
   • implement the component; and
   • adapt it to each use case.
3. The MDL principle, as adapted for components, suggests that the component minimizing the count of (2) possesses the ‘right’ level of generality.
Usability Tradeoffs (Blackwell, 2001) – Cog. Dimensions of Notation

- **Abstraction level** – minimum and maximum levels of abstraction exposed by the API
- **Working framework** – size of the conceptual chunk needed to work effectively.
- **Work-step unit** – how much of a programming task can be completed in a single step
- **Progressive evaluation** – what extent partially completed code can be executed
- **Premature commitment** – amount of decisions developers have to make in advance
- **Penetrability** – how the API facilitates exploration and understanding of its components
- **Elaboration** – extent to which the API must be adapted
- **Viscosity** – barriers to change inherent in the API (effort needed to make a change)
- **Consistency** – how much of the rest of an API can be inferred once part of it is learned
- **Role expressiveness** - how apparent relationships are between components & program
- **Domain correspondence** - how clearly API components map to the domain

- Adapted by Clarke, 2004
Which Syntax is Better (Round 1)?

```csharp
var names = select p.Name
            from p in people
```

```csharp
var names = from p in people
            select p.Name
```
Which Syntax is Better (Round 1)?

```csharp
var names = select p.Name
            from p in people
```

```csharp
var names = from p in people
            select p.Name
```

[Image of the names array with properties like Age, Birthday, Equals, GetHashCode, GetType, Name, ToString]
Which Syntax is Better (Round 2)?

```python
for x in [10, 20, 30]:
    print x
  10
  20
  30

for x in [10, 20, 30]:
    for y in [1, 2, 3]:
        print x+y
    11
    12
    13
    21
    22
    23
    31
    32
    33
```
Which Syntax is Better (Round 2)?

```python
for x in [10, 20, 30]; y in [1, 2]:
    print x+y
```

?
Which Syntax is Better (Round 2)?

```python
for x in [10, 20, 30]; y in [1, 2]:
    print x+y
```

11  11  error!
12  22
21
22
31
32

1st  2nd  3rd

Why?
“Furthermore, such [scientific evidence] as there is can be described as ‘black box’, that is, demonstrates a correlation between the use of a certain technique and an improvement in some aspect of the development.

It does not demonstrate how the technique achieves the observed effect.”

- Frank Bott, 2001
History of the Psychology of Programming

• **First Period** (1960-1979)
  – Imported theories and methods from Psychology
    • Short-term memory, statistics
  – Correlations between task and language/human factors
    • Comments, defects detected

• **Second Period** (1980-present)
  – Cognitive models
    • Knowledge, strategies, task, environment/tools
  – Response times, eye movements, intermediary code
  – Experts and students
Memory Model of the First Period (1960’s – 1970’s)

```cpp
#include <iostream>
using namespace std;
int main(int argc, char **argv) {
    cout << "Hello World" << endl;
}
```
Complexity Metrics & Miller’s Magic Number

- **Lines of code**
  - Coupled with everything!
- **Cyclomatic Complexity**
  - # of branches
- **Halstead Effort**
  - Operators, operands
- **Object-Oriented metrics**
  - Coupling, cohesion, inheritance
History of the Psychology of Programming

- **First Period** (1960-1979)
  - Imported theories and methods from Psychology
    - Short-term memory, statistics
  - Correlations between task and language/human factors
    - Comments, defects detected

- **Second Period** (1980-present)
  - Cognitive models
    - Knowledge, strategies, task, environment/tools
  - Response times, eye movements, intermediary code
  - Experts and students
#include <iostream>
using namespace std;
int main(int argc, char **argv) {
    cout << "Hello World" << endl;
}
From Chase & Simon (1973) to Soloway & Ehrlich (1984)
What Goes Here (Part 1) ? (Soloway & Erlich, 1984)

```plaintext
PROGRAM Green(input, output).
VAR I INTEGER,
    Letter, LeastLetter Char,
BEGIN
    LeastLetter = 'z'.
    FOR I = 1 TO 10 DO
        BEGIN
            READLN(Letter).
        -------
        -------
        END.
    Writeln(LeastLetter).
END
```
What Goes Here (Part 1)? (Soloway & Erlich, 1984)

```plaintext
PROGRAM Green(input, output),
VAR I INTEGER,
    Letter, LeastLetter Char.
BEGIN
    LeastLetter = 'z',
    FOR I = 1 TO 10 DO
        BEGIN
            READLN(Letter),
            If Letter < LeastLetter THEN LeastLetter = Letter.
        END
    End,
    WriteLn(LeastLetter),
END
```

```plaintext
PROGRAM Green(input, output),
VAR I INTEGER,
    Letter, LeastLetter Char.
BEGIN
    LeastLetter = 'z',
    FOR I = 1 TO 10 DO
        BEGIN
            READLN(Letter),
            If Letter < LeastLetter THEN LeastLetter = Letter.
        END
    End,
    WriteLn(LeastLetter),
END
```
What Goes Here (Part 2)? (Soloway & Erlich, 1984)

```plaintext
PROGRAM Orange(input, output).
VAR Sum, Count, Num INTEGER,
    Average REAL,
BEGIN
    Sum  = -99999,
    ------------------------
    |       |       |
    |       |       |
    |       |       |
    ------------------------
    REPEAT
        READLN(Num).
        Sum  = Sum + Num.
        Count = Count + 1.
    UNTIL Num  = 99999,
    Average = Sum/Count.
    WRITELN(Average),
END
```
What Goes Here (Part 2)? (Soloway & Erlich, 1984)
The Integrated Meta-Model (Von Mayrhauser, 1995)
More Modern Working Memory Model

```cpp
#include <iostream>
using namespace std;
int main(int argc, char **argv) {
    cout << "Hello World" << endl;
}
```

Eye Movements

Visual Input

Central Executive

Working Storage

- Verbal Rehearsal
- Visuospatial Sketchpad

Learning and Retrieval

Long-term Memory

- Perceptual
- Visual
- Autobiographical
- Declarative
- Linguistic/Semantic
- Habits/Motor Skills
The Stores Model of Code Cognition (Douce, 2008)
Quantifying Usability Tradeoffs

• **Problems**
  – Models are getting more complex
  – Verbal-conceptual theories

• **Precise Predictions**
  – Can eliminate bad designs with user studies, but…
  – Want to predict trade-offs in advance
  – Choose between “good” designs

• **TODOs**
  – Quantify cognitive model(s)
  – Simulate model(s) with different designs
  – Interpret simulation output as trade-offs
The Cognitive Complexity Metric (Cant, 1995)

\[ C_i = R_i + \sum_{j \in N} C_j + \sum_{j \in N} T_j \]

Chunk 0

\[ \vdots \]

Chunk 1

\[ \vdots \]

Chunk 2

Chunking

Tracing
The Cognitive Complexity Metric (Cant, 1995)

\[ C_i = R_i + \sum_{j \in N} C_j + \sum_{j \in N} T_j \]

\[ R_F(R_S + R_C + R_E + R_R + R_V + R_D) \]

Chunk Complexity (R) – 1/3

- $R_S = \text{size of chunk}$
  - $R_S = \begin{cases} aS_i & \text{if } S_i \leq L \\ aS_i + b \left( \frac{S_i - L}{L} \right) & \text{else} \end{cases}$
  - $S_i = \text{size measure, } L = \text{programmer's limit}$

- $R_C = \text{difficulty in comprehending control structure}$
  - $R_C = \sum_j C_i(j)p(j)$
  - $C_i(j) = \text{complexity of chunk } i \text{ after } j^{th} \text{ iteration}$
  - $p(j) = \text{the probability of termination}$

- $R_E = \text{difficulty in comprehending boolean expressions}$
  - $R_E = b_1 \sum_j B_j$
  - $B_j = \text{num. of predicates in } j^{th} \text{ boolean expression}$
Chunk Complexity (R) – 2/3

- \( R_R = \text{recognizability of chunk} \)
  - \( R_R = r_R + r_C \)
  - \( r_R = -\log(\prod_j p(t_j)) \), \( r_C = \text{cohesion measure} \)
  - \( p(t_j) = \text{prob. of the } j^{th} \text{ token being drawn from rules of discourse} \)

- \( R_V = \text{visual structure, layout of the code} \)
  - \( R_V = a_1 V \) where \( V = \{1, 2, 3\} \)
  - 1 = method (easiest), 2 = control structure (harder), 3 = neither (hardest)
Chunk Complexity (R) – 3/3

- \( R_D = \) disruptions caused by dependencies
  - \( R_D = d \sum_j C_j + e \sum_j T_j \)
  - \( C_j = \) complexity of sub-chunk \( j \)
  - \( T_j = \) difficulty in tracing sub-chunk \( j \)

- \( R_F = \) familiarity of the chunk
  - \( R_F = \sum_j f^j \) where \( f \) is a review constant (\( \approx \frac{2}{3} \))
The Cognitive Complexity Metric (Cant, 1995)

\[
C_i = R_i + \sum_{j \in N} C_j + \sum_{j \in N} T_j
\]

\[
T_F(T_L + T_A + T_S + T_C)
\]

Famil. (Localization + Ambiguity + Spacial Dist. + Cueing)
Tracing Difficulty (T) – 1/2

- $T_L =$ localization of dependencies
  - $T_L = a_2 L$ where $L = \{1, 2, 3\}$
  - 1 = embedded (easiest), 2 = local (harder) 3 = remote (hardest)

- $T_A =$ ambiguity
  - $T_A = a_3 A$ where $A = \{0, 1\}$ (1 = is ambiguous)

- $T_S =$ spacial distance
  - $T_S = b_2 \Delta S$ where $S =$ lines of code
Tracing Difficulty (T) – 2/2

- $T_C = \text{cueing}$
  - $T_C = a_4 B$ where $B = \{0, 1\}$ (1 = is obscure)
  - Some chunks are easier to find when not embedded in a larger block of text

- $T_F = \text{familiarity}$
  - $T_F = \sum_j f^j$ where $f \approx \frac{2}{3}$
The ACT-R Cognitive Framework (Anderson, 2007)

$$\text{(p encode-letter}$$

$$=\text{goal}>$$

$$\text{isa read-letters}$$

$$\text{state attend}$$

$$=\text{visual}>$$

$$\text{isa text}$$

$$\text{value } =\text{letter1}$$

$$?\text{imaginal} >$$

$$\text{buffer empty}$$

$$\Rightarrow$$

$$=\text{goal}>$$

$$\text{state wait}$$

$$+\text{imaginal} >$$

$$\text{isa array}$$

$$\text{letter1 } =\text{letter1}$$

$$)$$
## The Cognitive Complexity Metric in ACT-R?

<table>
<thead>
<tr>
<th>Process</th>
<th>Factor</th>
<th>Modules</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chunking</strong></td>
<td>Familiarity</td>
<td>Declarative</td>
<td>Repeated retrievals are faster</td>
</tr>
<tr>
<td></td>
<td>Size</td>
<td>Visual</td>
<td>more code to encode</td>
</tr>
<tr>
<td></td>
<td>Control Structures</td>
<td>Imaginal, Vocal</td>
<td>mental iteration, boolean exps</td>
</tr>
<tr>
<td></td>
<td>Boolean Expressions</td>
<td>Visual, Declarative</td>
<td>visual/mental shortcuts</td>
</tr>
<tr>
<td></td>
<td>Recognizability</td>
<td>Visual, Imaginal</td>
<td>real code, build up representation</td>
</tr>
<tr>
<td></td>
<td>Visual Structure</td>
<td>Visual</td>
<td>whitespace is significant</td>
</tr>
<tr>
<td><strong>Dependency Disruptions</strong></td>
<td>Visual, Declarative, Manual</td>
<td></td>
<td>development env.</td>
</tr>
<tr>
<td><strong>Tracing</strong></td>
<td>Familiarity</td>
<td>Declarative</td>
<td>repeated retrievals, open tabs</td>
</tr>
<tr>
<td></td>
<td>Localization</td>
<td>Visual</td>
<td>search strategy, external tools</td>
</tr>
<tr>
<td></td>
<td>Ambiguity</td>
<td>Declarative</td>
<td>partial matching (variable name)</td>
</tr>
<tr>
<td></td>
<td>Spatial Distance</td>
<td>Visual, Manual</td>
<td>what is “distance”?</td>
</tr>
<tr>
<td></td>
<td>Level of Cueing</td>
<td>Visual</td>
<td>whitespace again, but less sensitive</td>
</tr>
</tbody>
</table>
Questions?