S05 Code Review

Software Complexity
Code Review
Software Design
Comments and Documentation

ANU – School of Computing – Structured Programming 1110 / 1140 / 6710

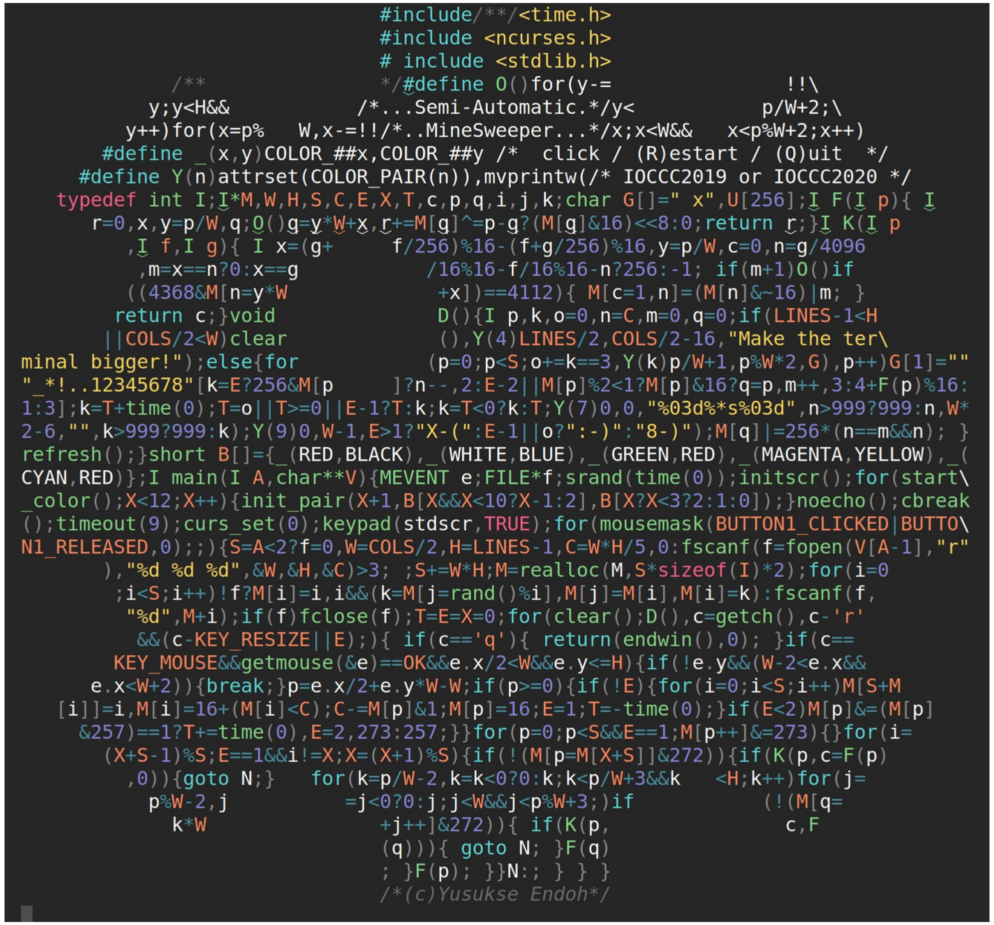
Software Complexity

```
++++++++[>++++[>++>+++>+++>+<<<<-]>+>+>->>+
[<]<-]>>.>---.+++++++..+++.>>.<-.<.++
+.------.--------.>>+.>++.
```

- “Hello World” in the BrainF#@k language (apparently: source wikipedia)
- Syntax only 8 characters, *Turing complete*
- Simple or complex?

Software Complexity

- The International **Obfuscated** C Code Contest
- Yusuke Endoh one of the 2020 winners: Minesweeper Solver



```c
                               #include/**/<time.h>
                               #include <ncurses.h>
                               # include <stdlib.h>
             /**               */#define O()for(y-=               !!\
           y;y<H&&           /*...Semi-Automatic.*/y<           p/W+2;\
         y++)for(x=p%   W,x-=!!/*..MineSweeper...*/x;x<W&&   x<p%W+2;x++)
       #define _(x,y)COLOR_##x,COLOR_##y /*  click / (R)estart / (Q)uit  */
     #define Y(n)attrset(COLOR_PAIR(n)),mvprintw(/* IOCCC2019 or IOCCC2020 */
   typedef int I;I*M,W,H,S,C,E,X,T,c,p,q,i,j,k;char G[]=" x",U[256];I F(I p){ I
      r=0,x,y=p/W,q;O()g=y*W+x,r+=M[g]^=p-g?(M[g]&16)<<8:0;return r;}I K(I p
         ,I f,I g){ I x=(g+     f/256)%16-(f+g/256)%16,y=p/W,c=0,n=g/4096
          ,m=x==n?0:x==g           /16%16-f/16%16-n?256:-1; if(m+1)O()if
         ((4368&M[n=y*W             +x])==4112){ M[c=1,n]=(M[n]&~16)|m; }
        return c;}void              D(){I p,k,o=0,n=C,m=0,q=0;if(LINES-1<H
       ||COLS/2<W)clear             (),Y(4)LINES/2,COLS/2-16,"Make the ter\
minal bigger!");else{for           (p=0;p<S;o+=k==3,Y(k)p/W+1,p%W*2,G),p++)G[1]=""
"_*!..12345678"[k=E?256&M[p     ]?n--,2:E-2||M[p]%2<1?M[p]&16?q=p,m++,3:4+F(p)%16:
1:3];k=T+time(0);T=o||T>=0||E-1?T:k;k=T<0?k:T;Y(7)0,0,"%03d%*s%03d",n>999?999:n,W*
2-6,"",k>999?999:k);Y(9)0,W-1,E>1?"X-(":E-1||o?":-)":"8-)");M[q]|=256*(n==m&&n); }
refresh();}short B[]={_(RED,BLACK),_(WHITE,BLUE),_(GREEN,RED),_(MAGENTA,YELLOW),_(
CYAN,RED)};I main(I A,char**V){MEVENT e;FILE*f;srand(time(0));initscr();for(start\
_color();X<12;X++){init_pair(X+1,B[X&&X<10?X-1:2],B[X?X<3?2:1:0]);}noecho();cbreak
();timeout(9);curs_set(0);keypad(stdscr,TRUE);for(mousemask(BUTTON1_CLICKED|BUTTO\
N1_RELEASED,0);;){S=A<2?f=0,W=COLS/2,H=LINES-1,C=W*H/5,0:fscanf(f=fopen(V[A-1],"r"
       ),"%d %d %d",&W,&H,&C)>3; ;S+=W*H;M=realloc(M,S*sizeof(I)*2);for(i=0
        ;i<S;i++)!f?M[i]=i,i&&(k=M[j=rand()%i],M[j]=M[i],M[i]=k):fscanf(f,
         "%d",M+i);if(f)fclose(f);T=E=X=0;for(clear();D(),c=getch(),c-'r'
          &&(c-KEY_RESIZE||E);){ if(c=='q'){ return(endwin(),0); }if(c==
        KEY_MOUSE&&getmouse(&e)==OK&&e.x/2<W&&e.y<=H){if(!e.y&&(W-2<e.x&&
      e.x<W+2)){break;}p=e.x/2+e.y*W-W;if(p>=0){if(!E){for(i=0;i<S;i++)M[S+M
   [i]]=i,M[i]=16+(M[i]<C);C-=M[p]&1;M[p]=16;E=1;T=-time(0);}if(E<2)M[p]&=(M[p]
     &257)==1?T+=time(0),E=2,273:257;}}for(p=0;p<S&&E==1;M[p++]&=273){}for(i=
       (X+S-1)%S;E==1&&i!=X;X=(X+1)%S){if(!(M[p=M[X+S]]&272)){if(K(p,c=F(p)
         ,0)){goto N;}   for(k=p/W-2,k=k<0?0:k;k<p/W+3&&k   <H;k++)for(j=
           p%W-2,j          =j<0?0:j;j<W&&j<p%W+3;)if           (!(M[q=
             k*W               +j++]&272)){ if(K(p,               c,F
                               (q))){ goto N; }F(q)
                               ; }F(p); }}N:; } } }
                               /*(c)Yusukse Endoh*/
```

What is Software Complexity?

- ***Accidental* Complexity**
  - Software that is designed or presented in a way that is more difficult for a **human to understand, use and modify** *than it needs to be*.
  - It is difficult to write elegant, clear, reusable code.
- ***Essential* Complexity**
  - Inherent to the problem being solved. Irreducible.
- **Not to be confused with** computational complexity (about performance).

Software Complexity

- Some **contributing factors**:
  - Poorly named variables
  - Not following conventions / inconsistency
  - Interlinking many components
  - Unstated assumptions
  - Non-local changes, unintuitive side-effects
  - Duplication / lack of encapsulation / exposure to details
- Often **incrementally** works its way into a project, e.g., *feature creep*, dealing with *legacy*.

Code Review

- One or more people review code who are removed from the implementation.
- Commonly done for a specific change (e.g., set of git commits) but can also be done for a complete project / implementation.
  - **Fix** a specific bug
  - **Implement** a new feature
  - **Refactor** part of the code
- Gitlab offers a “merge request” workflow (“pull request” on github) where reviewers / maintainers review the changes **before** they are merged into the mainline branch.

Code Review Motivations

- Barrier to ensure project remains **maintainable**.
  - Improve implementation / quality.
  - Clarify code, double-check edge cases.
  - On-balance rejection of a feature (accidental or essential complexity).
- Second pair of eyes: potentially less biased, can consider bigger picture, can bring new insight.
- Effective way to **learn** a new code-base and a team’s processes / conventions. Highlights interrelated parts.
- Can catch some bugs before reaching production… but implementer really should have adequate tests developed and passing.

Doing a Code Review

- **Objective**: is it in scope of this project
- **Functionality** (for end-users and developers):
  - does it do what is intended
  - edge cases / bugs
  - might have to run code for UI changes etc
- **Tests**: present, appropriate
- **Complexity**: design minimises / encapsulates complexity
- **Good names**: convey information and not too long
- **Comments**: help to understand decisions and the why, not repeating code, appropriately documenting interfaces
- **Conformance** to project style guide / conventions.

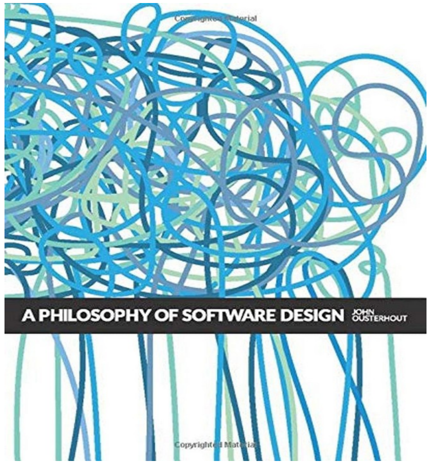
Further Tips

- Be considerate.
- Point out things that are good!
- Clearly label *nitpicks* as such.
- No code is ever perfect. Tailor to circumstances:
  - flight control software
  - a game

Good Software Design

- Many opinions. Conventions / preferences vary between communities.
- Recommendation:

  A Philosophy of Software Design, John Ousterhout



- Design principles
- Red flags

Some Principles (Ousterhout)

- **Deep “modules”** (method, class, package, or module)
  - Simple interfaces* (narrow)
  - Encapsulate lots of complexity (depth)
  - General-purpose
- Prefer **simple interface** over simple implementation
- Design **errors out of existence**
- Design for **ease of reading**, not ease of writing
- Extra: Don’t Repeat Yourself (**DRY**) and **SOLID** principles

* Interfaces in the broad sense, not just the Java keyword

Some Red Flags (Ousterhout)

- **Shallow module**: interface not much simpler than implementation
- **Overexposure**: user needs to be aware of rarely-used features
- **Repetition**: non-trivial code is repeated
- **Conjoined methods**: methods are so co-dependent that you have to understand implementation of both
- **Comment repeats code**
- **Hard to name entity**
- Extra: **Deeply nested control-flow blocks**

Code Comments / Documentation

- **Class or method comments – always for** `public`
  - How to use, edge cases, side-effects, pre/post-conditions, invariants, explain abstraction, examples.
  - Should not leak the implementation details.
- **Implementation comments – as required**
  - Give intuition where implementation is non-obvious to a likely contributor / your future self
  - Highlight where edge cases are handled if hidden
  - Rationale for the design if not the obvious choice
  - Should not just repeat code