CS 5500

Spring 2013

Plan to Throw One Away
(chapter 11 of *The Mythical Man-Month*)

In most projects, the first system built is barely usable....

Hence *plan to throw one away; you will, anyhow*.

— Fred Brooks

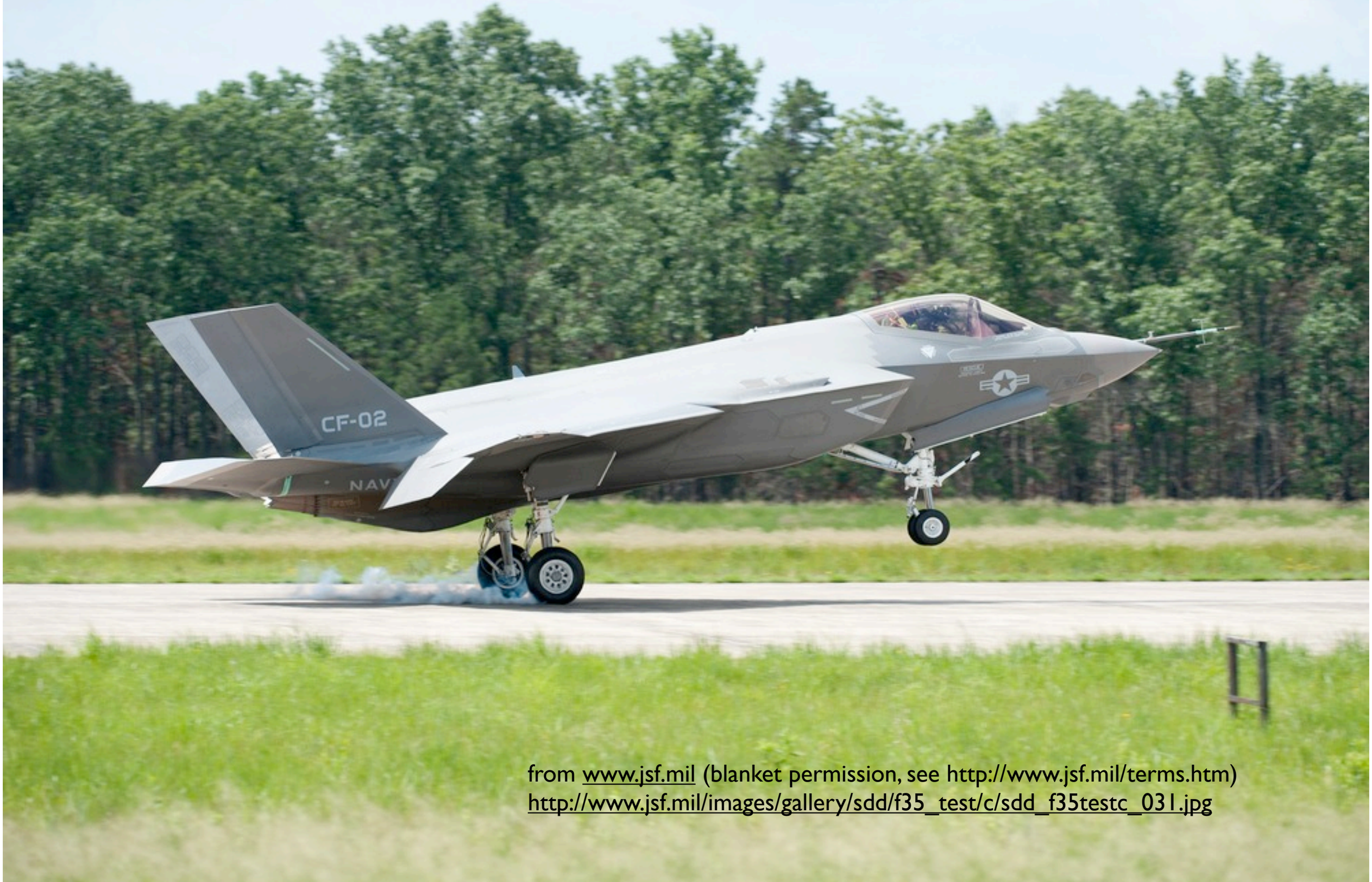
Rapid Prototyping

get something running
take shortcuts
make your mistakes quickly
learn from those mistakes

The Only Constancy is Change Itself

...both the actual need and the user's perception of that need will change as programs are built, tested, and used.

Design Changes Can Be Expensive



from www.jsf.mil (blanket permission, see http://www.jsf.mil/terms.htm)
http://www.jsf.mil/images/gallery/sdd/f35_test/c/sdd_f35testc_031.jpg

Plan the System for Change

strong abstraction barriers
abstract data types
representation independence
well-specified interfaces
good documentation
modularity

Plan the System for Change

high-level programming languages
automatic garbage collection

Plan the System for Change

table-driven
automatic translation of specs into code
declarative programming

Automatic Translation

just one of 231 states

```
(define (state0 c)
  (case c
    ((#\`) (consumeChar) (accept 'backquote))
    ((#\') (consumeChar) (accept 'quote))
    ((#\]) (consumeChar) (accept 'rbracket))
    ((#\[) (consumeChar) (accept 'lbracket))
    ((#\)) (consumeChar) (accept 'rparen))
    ((#\() (consumeChar) (accept 'lparen))
    ((#\tab #\newline #\vtab #\page #\return #\space)
     (consumeChar)
     (begin
       (set! string_accumulator_length 0)
       (state0 (scanChar))))
    ((#\;) (consumeChar) (state201 (scanChar)))
    ((#\#) (consumeChar) (state200 (scanChar)))
    ((#\0 #\2 #\3 #\4 #\5 #\6 #\7 #\8 #\9)
     (consumeChar)
     (state131 (scanChar)))
    ((#\a #\b #\c #\d #\e #\f #\g #\h #\i #\j #\k #\l #\m
      #\n #\o #\p #\q #\r #\s #\t #\u #\v #\w #\x #\y #\z
      #\A #\B #\C #\D #\E #\F #\G #\H #\I #\J #\K #\L #\M
      #\N #\O #\P #\Q #\R #\S #\T #\U #\V #\W #\X #\Y #\Z
      #\! #\$ #\% #\& #\* #\/ #\: #\< #\= #\> #\? #\^ #\_
      #\~ #\@ #\|)
     (consumeChar)
     (state68 (scanChar)))
    ((#\\) (consumeChar) (state67 (scanChar)))
    ((#\+) (consumeChar) (state61 (scanChar)))
    ((#\-) (consumeChar) (state60 (scanChar)))
    ((#\1) (consumeChar) (state57 (scanChar)))
    ((#\.) (consumeChar) (state56 (scanChar)))
    ((#\,) (consumeChar) (state5 (scanChar)))
    ((#\") (consumeChar) (state4 (scanChar)))
    (else
     (if ((lambda (c)
            (and (char? c)
                 (> (char->integer c) 127)
                 (let ((cat (char-general-category c)))
                   (memq cat
                         '(Lu Ll Lt Lm Lo Mn Nl No
                           Pd Pc Po Sc Sm Sk So Co)))))
          c)
         (begin (consumeChar) (state68 (scanChar)))
         (if (eof-object? c)
             (begin (consumeChar) (accept 'eofobj))
             (if ((lambda (c) (and (char? c) (char-whitespace? c)))
                  c)
                 (begin
                   (consumeChar)
                   (begin
                     (set! string_accumulator_length 0)
                     (state0 (scanChar))))
                 (if ((lambda (c)
                        (and (char? c) (char=? c (integer->char 133))))
                      c)
                     (begin
                       (consumeChar)
                       (begin
                         (set! string_accumulator_length 0)
                         (state0 (scanChar))))
                     (scannerError errIncompleteToken)))))))))
```

Automatic Translation

regular expression for identifiers

```
(id (! ((! ; constituent
           %a..z %A..Z isOtherConstituent
           ; special initial
           #\! #\$ #\% #\& #\* #\/ #\: #\< #\= #\> #\? #\^ #\_ #\~
           ; inline hex escape
           (#\\ #\x (! %0..9 %a..f %A..F)
                    (* (! %0..9 %a..f %A..F)) #\;)
           (#\\ isAscii)                                  ; backslash escape
           (#\# #\%)                                   ; MzScheme randomness
           #\. #\@ #\|                ; leading dot or @ or vertical bar
           (#\+ #\:)                                          ; leading +:
           (#\- #\:)                                          ; leading -:
           ; peculiar prefix
           (#\- #\>))
       (* (! %a..z %A..Z isOtherConstituent
             #\! #\$ #\% #\& #\* #\/ #\: #\< #\= #\> #\? #\^ #\  #\~
             (#\\ #\x (! %0..9 %a..f %A..F)
                      (* (! %0..9 %a..f %A..F)) #\;)
             (#\\ isAscii)                                ; backslash escape
             (#\# #\%)                                 ; MzScheme randomness
             #\|                      ; embedded or trailing vertical bar
             %0..9 #\+ #\- #\. #\@ isNdMcMe)))
; peculiar identifiers
#\+ #\- (#\. #\. #\.)
(#\- #\-)                                                          ;  --
(#\- #\1 #\+)                                                      ; -1+
(#\1 #\+)                                                          ;  1+
(#\1 #\-)                                                          ;  1-
))
```

Automatic Translation

```
<datum>          ::=  boolean                          #makeBool
                 ::=  number                           #makeNum
                 ::=  character                        #makeChar
                 ::=  string                           #makeString
                 ::=  id                               #makeSym
                 ::=  miscflag                         #makeFlag
                 ::=  <location> <structured>          #makeStructured
<structured>     ::=  <list>                           #identity
                 ::=  <vector>                         #identity
                 ::=  <bytevector>                     #identity
```

Automatic Translation

```
(define (parse-datum)
  (case (next-token)
    ((boolean) (begin (consume-token!) (makeBool)))
    ((number) (begin (consume-token!) (makeNum)))
    ((character) (begin (consume-token!) (makeChar)))
    ((string) (begin (consume-token!) (makeString)))
    ((id) (begin (consume-token!) (makeSym)))
    ((miscflag) (begin (consume-token!) (makeFlag)))
    ((unsyntaxsplicing unsyntax quasisyntax syntax
     splicing comma backquote quote lbracket lparen
     vecstart bvecstart)
     (let ((ast1 (parse-location)))
       (let ((ast2 (parse-structured)))
         (makeStructured ast1 ast2))))
   (else
    (parse-error
      '<datum>
      '(backquote
        boolean
        ...)))))
```

Plan the System for Change

version control

Plan the Organization for Change

Maintenance

adds new features
improves performance or usability
fixes bugs that weren't caught before release
fixes minor bugs that were caught but not fixed
finds latent bugs
repairs bit rot

Bit Rot

```
#if WIN32
  switch(GetSystemDEPPolicy())
  {
  case 0: break; /* AlwaysOff: we don't care about this */

  case 1:        /* AlwaysOn: won't work */

        panic_exit("DEP policy is too restrictive for Larceny to run");
        break;

  case 2: break; /* OptIn: we don't care about this either */

  case 3:        /* OptOut: we can opt out */

        if(!SetProcessDEPPolicy(0))
        {
            consolemsg("Failed to set DEP policy");
        }
        break;
  }
#endif /* WIN32 */
```

Two Steps Forward and One Step Back

One Step Forward and One Step Back