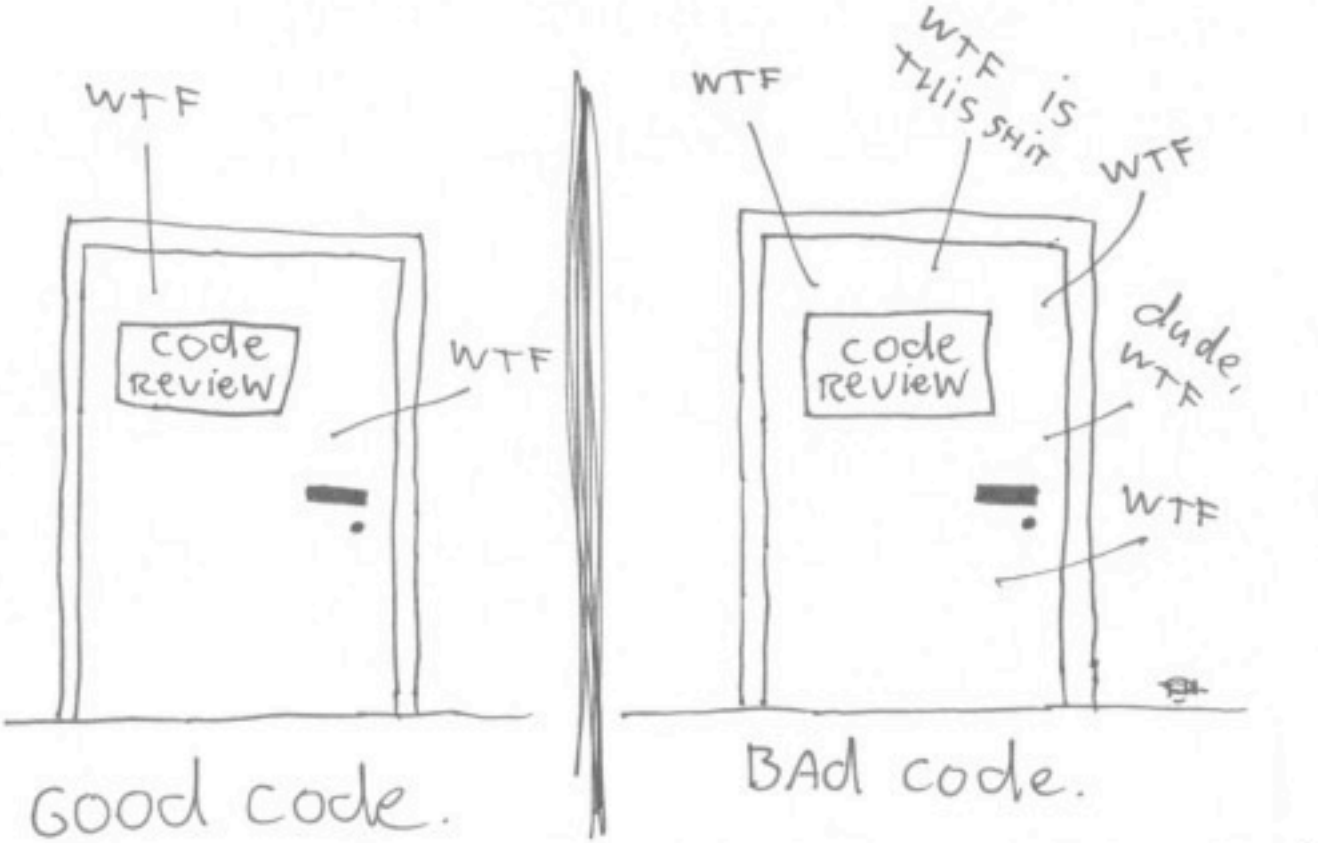
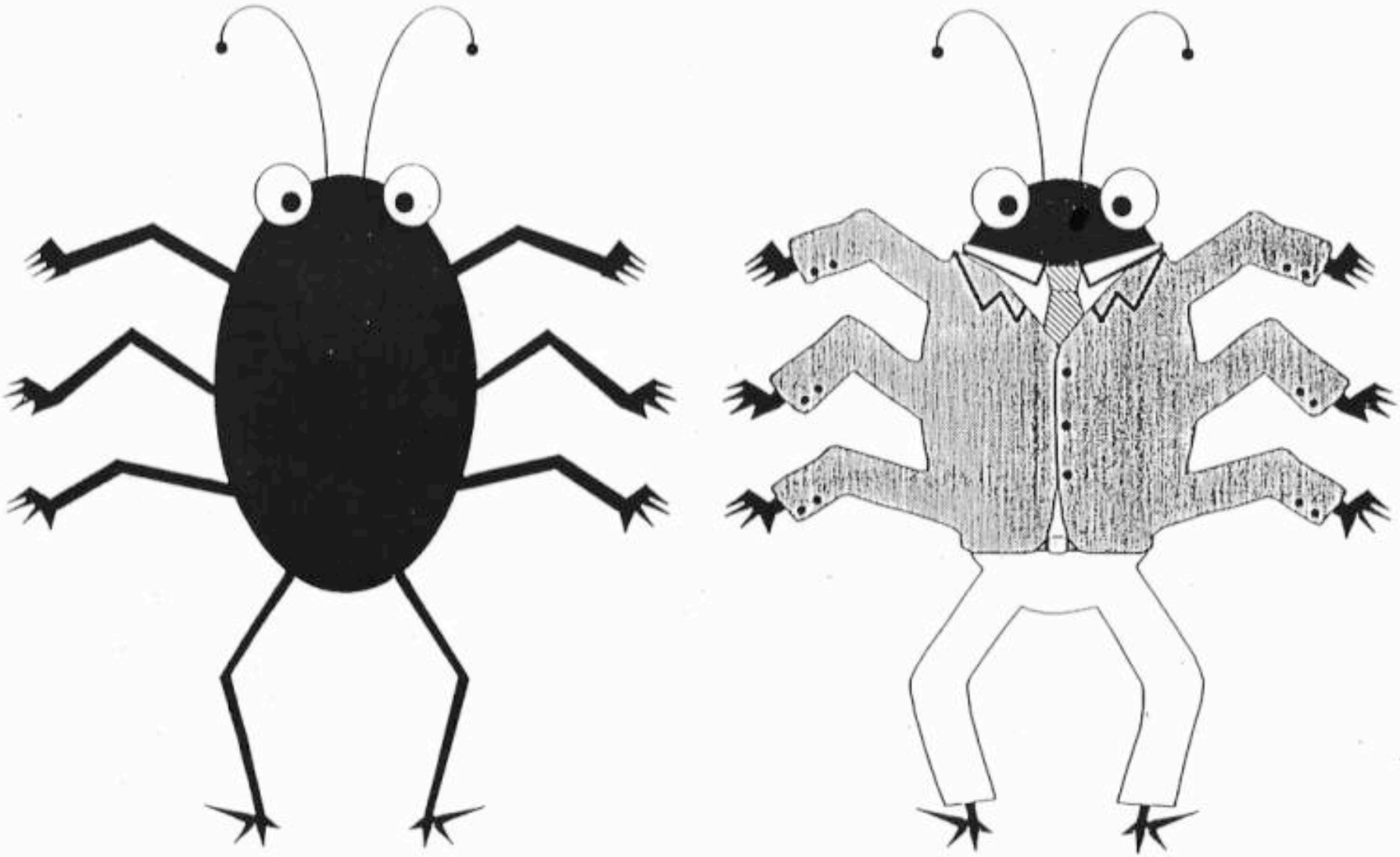
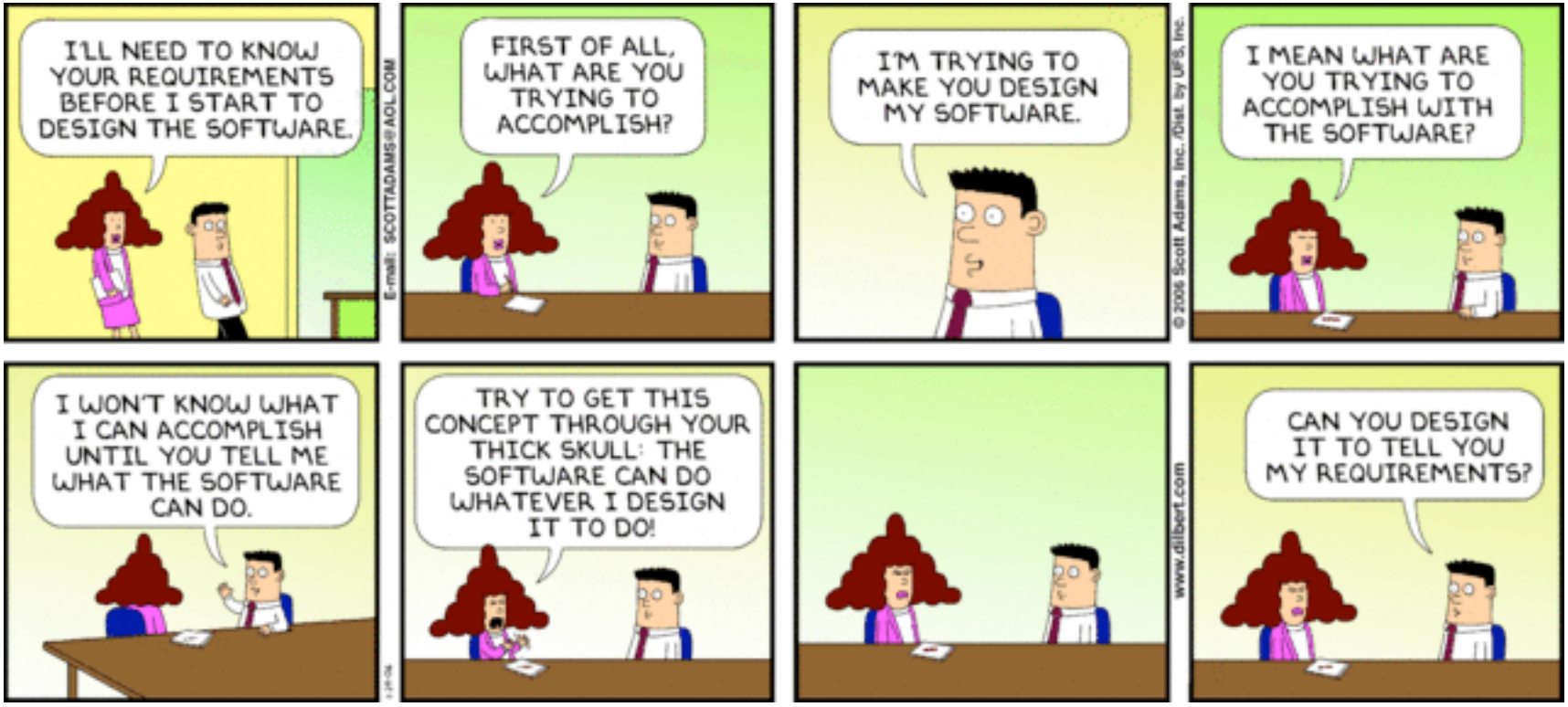
The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE

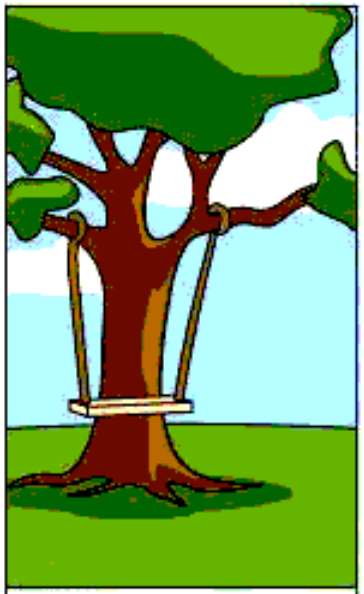



WE'RE GOING TO TRY SOMETHING CALLED AGILE PROGRAMMING.
www.dilbert.com scottadams@aol.com
THAT MEANS NO MORE PLANNING AND NO MORE DOCUMENTATION. JUST START WRITING CODE AND COMPLAINING.
I'M GLAD IT HAS A NAME.
THAT WAS YOUR TRAINING.
11-26-07 ©2007 Scott Adams, Inc./Dist. by UFS, Inc.


BUG
FEATURE

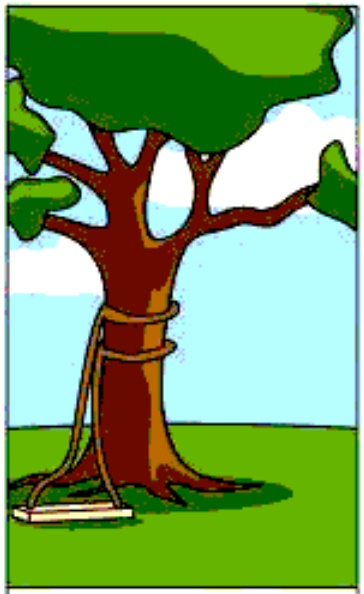

I'LL NEED TO KNOW YOUR REQUIREMENTS BEFORE I START TO DESIGN THE SOFTWARE.
E-mail: SCOTTADAMS@AOL.COM
FIRST OF ALL, WHAT ARE YOU TRYING TO ACCOMPLISH?
I'M TRYING TO MAKE YOU DESIGN MY SOFTWARE.
© 2006 Scott Adams, Inc./Dist. by UFS, Inc.
I MEAN WHAT ARE YOU TRYING TO ACCOMPLISH WITH THE SOFTWARE?
I WON'T KNOW WHAT I CAN ACCOMPLISH UNTIL YOU TELL ME WHAT THE SOFTWARE CAN DO.
TRY TO GET THIS CONCEPT THROUGH YOUR THICK SKULL: THE SOFTWARE CAN DO WHATEVER I DESIGN IT TO DO!
www.dilbert.com
CAN YOU DESIGN IT TO TELL YOU MY REQUIREMENTS?

How the customer explained it



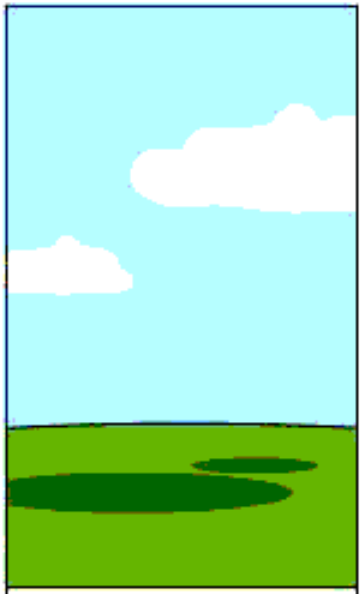
How the Project Leader understood it

How the Analyst designed it

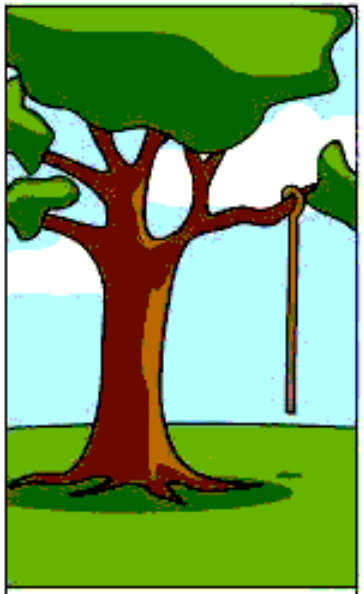


How the Programmer wrote it

How the Business Consultant described it

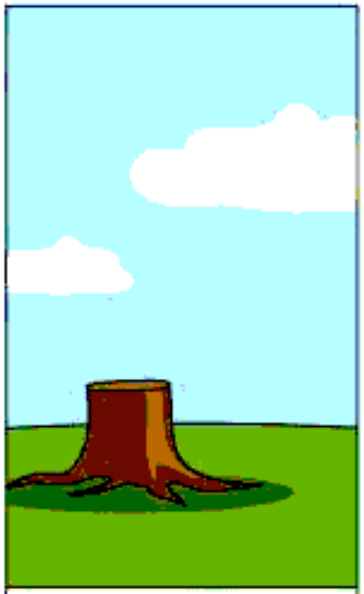


How the project was documented

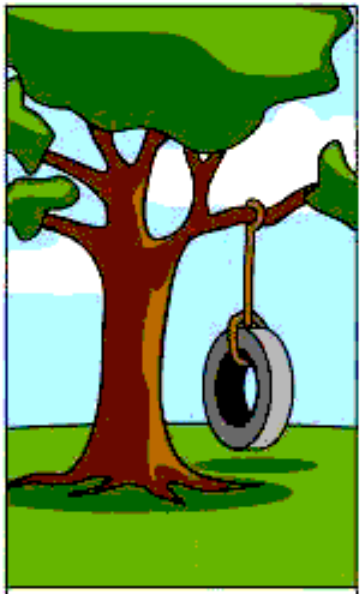


What operations installed

How the customer was billed



How it was supported



What the customer really needed

Types of documentation

- Internal documentation (comments in your code)
  - Plenty of this so far
- External programmer documentation (for other programmers who would work with your code)
  - UML documents
- User documentation (the manual for the poor fools who will be using your code)
  - DOxygen

How to write good comments

- Does your comment help your reader understand the code?
- Are you writing a comment just because you know that "comments are good"?
- Is the comment something that the reader could easily work out for themselves?
- Don't be afraid to add a reference instead of a comment for tricksy things

Some poor commenting

```
i= i+1;  /* Add one to i */

for (i= 0; i < 1000; i++) { /* Tricksy bit */
.
. Hundreds of lines of obscure uncommented code here
.
}
int x,y,q3,z4;  /* Define some variables */

int main()
/* Main routine */

while (i < 7) { /*This comment carries on and on
```

How comments can make code worse

```
while (j < ARRAYLEN) {
    printf ("J is %d\n", j);
    for (i= 0; i < MAXLEN; i++) {
        for (k= 0; k < KPOS; k++) {
            printf ("%d %d\n",i,k);
        }
    }
    j++;
}
```

Some more poor comments

```
while (j < ARRAYLEN) {
    printf ("J is %d\n", j);
    for (i= 0; i < MAXLEN; i++) {
/* These comments only */
        for (k= 0; k < KPOS; k++) {
/* Serve to break up */
            printf ("%d %d\n",i,k);
/* the program */
        }
/* And make the indentation */
    }
/* Very hard for the programmer to see */
    j++;
}
```

Review: how much to comment?

- Just because comments are good doesn't mean that you should comment every line.
- Too many comments make your code hard to read.
- Too few comments make your code hard to understand.
- Comment only where you couldn't trivially understand what was going on by looking at the code for a minute or so.

What should I comment for our project/ in general?

- Every file (if you do multi-file programming) to say what it contains
- Every function – what variables does it take and what does it return. (I like to slightly comment the prototypes too to give a hint)
- Every variable apart from "obvious" ones (`i`, `j`, `k` for loops and `FILE *fptr` don't require a comment but `int total;` might)
- Every class/typedef (unless it's really trivial)

Other rules for comments

- Comment if you do something "weird" that might fool other programmers.
- If a comment is getting long consider referring to other text instead
- Don't let comments interfere with how the code looks (e.g. make indentation hard to find)

External (programmer) documentation

- This tells other programmers what your code does
- Most large companies have their own standards for doing this
- The aim is to allow another programmer to use and modify your code without having to read and understand every line
- Your projects will include this type of documentation (but probably not enough to really be passed on to other programmers)
- Note – everyone has their own rules.

External documentation (Stage 1)

- Describe how your code works generally
- What is it supposed to do?
- What files does it read from or write to?
- What does it assume about the input?
- What algorithms does it use

External Documentation (stage 2)

- Describe the general flow of your program (no real need for a flowchart though)
- UML Diagrams can help here.
- Explain any complex algorithms which your program uses or refer to explanations elsewhere. (e.g. "I use the vcomplexsort see Knuth page 45 for more details")

External documentation(stage 3)

- If you use multi-file programming explain what each file contains
- Explain any class used a lot in your program
- You might also like to explain (and justify) any global variables you have chosen to use
  - More important for C, but mentioned for completeness

External documentation (stage 4)

- Describe every "major" member function in your classes and functions in your program
- Describe what arguments must be passed and what is returned.
- (It is up to you to decide what is a "major" function – and really depends on the level of detail you wish to document to).
- Consider which functions are doing "the real work" – they might not necessarily be the longest or most difficult to write ones.

User documentation

- This is documentation for the user of your program
- It is the "user manual"
- Entire books have been written on the subject and we will not cover it here
- Feel free to include user documentation for your project, but the minimum requirement is a README as described.