The Software Engineer As Artist and Detective

Michael J. Ciaraldi

1999/05/28

What Does a Software Engineer Need to Do Her Job?

- Knowledge (factual and procedural)
- Skills

Bloom's Taxonomy of Educational Objectives

1) Knowledge
   - recall of memorized material.
2) Comprehension
   - demonstrate understanding, e.g. restate in own words.
3) Application
   - apply to new situation, e.g. apply algorithm or formula to new problem of same type.

Bloom's Taxonomy of Educational Objectives II

4) Analysis
   - break down material or problem into component parts.
5) Synthesis
   - reassemble parts into a new whole, e.g. design or write a new program.
6) Evaluation
   - apply criteria to judge worth for a particular purpose

Knowledge

- How a computer works
  - Hardware/architecture/machine organization
- How software works
  - Compilers
  - Operating systems
- Languages
  - Different kinds, different tools.

Knowledge II

- Design and analysis techniques
- Well-known algorithms, data structures, and techniques.
- Theory
  - Formal languages, graphs, etc.

Skills

- System analysis
- Programming
  - Must be effortless if the technique and goal are well-understood.
- Documentation and communication
- How to search the literature.

All This Is the Craft of Computer Science

- Covers first 3 layers
  1) Knowledge
  2) Comprehension
  3) Application
- And part of the rest
  4) Analysis
  5) Synthesis
  6) Evaluation

Additional Needed Skills

- Problem-solving
  - Figuring out what really needs to be accomplished (from the perspective of the problem)
  - What is needed to accomplish this?

Additional Skills II

- Recognizing patterns, e.g.
  - Data abstraction & hiding.
  - Network layers.
  - Virtual machines.
  - Design for reuse.

This Is the Art of Computer Science.

- Rest of top three
  4) Analysis
  5) Synthesis
  6) Evaluation
- And beyond!

Art and Craft

- An expert house painter has to be able to put the right colors on the right part of the house. He is a skilled craftsman who knows his tools well.
- A portrait painter decides what color to put where. She is an artist.

Art and Craft II

- An artist has to be a craftsman, but that is not sufficient. She knows:
  - How to draw in the conventional style
  - What its limits are
  - When to deviate from that.

Example

John Singer Sargent captured people's personalities in their portraits. Each subject looks in a particular direction with a particular expression. He had to decide what each person was doing, then figure out how to convey that in paint.

What Does a Software Engineer Do?

- Figure out what the problem is.
- Decide how to solve it.
- Then implement the solution.

In general, if we knew how to solve the problem, we could just buy a program or library to do it. Software engineers are paid to solve new problems, or old problems in better ways.

An Iterative Process With Feedback.

- Often the problem being solved is not well-defined or even well-understood. Only by attempting to solve it do you gain the insight needed to understand it.
- User feedback--Are you solving the problem of one client or many potential customers?

How to Acquire This Knowledge and Develop These Skills?

- The craft can only be learned by practice.
  - It can be learned most efficiently if the practice is well-guided (where the teacher comes in).
- The art can only be learned/developed by trying to define & achieve goals.
  - a.k.a. problem identification & solving.

A Software Engineer Is Like a Detective

- Craft
  - Disguise
  - Chemical analysis
  - Fingerprinting
- Art
  - Determine what to look for
  - Form and test hypotheses

A Software Engineer Is Like a Detective II

- He must figure out what the problem really is.
  - Many of Sherlock Holmes' cases did not turn out to be the crime originally thought, or even a crime at all.
  - How to do this—combine knowledge, analytical skill, questioning, insight, experience, and intuition.

A Software Engineer Is Like a Detective III

- He must figure out what his tools really do. (Not what the manual says they do).
  - Sometimes the manual is misleading or ambiguous, leaves out important information, or is just plain wrong.
    - Example: putenv() is described & implemented differently in different versions of Unix.

Linux Putenv()

The putenv() function adds or changes the value of environment variables. The argument *string* is of the form *name*=*value*. If *name* does not already exist in the environment, then *string* is added to the environment. If *name* does exist, then the value of *name* in the environment is changed to *value*.

Sunos 4.1 Putenv()

...the string pointed to by *string* becomes part of the environment, so altering the string will change the environment. The space used by *string* is no longer used once a new string-defining name is passed to putenv().

WARNING: A potential error is to call putenv() with an automatic variable as the argument, then exit the calling function while *string* is still part of the environment.

SunOs (Solaris) 5.6 Putenv()

...*string* should not be an automatic variable.

string should be declared static if it is declared within function because it cannot be automatically declared.

A potential error is to call the function putenv() with a pointer to an automatic variable as the argument and to then exit the calling function while *string* is still part of the environment.

A Software Engineer Is Like a Detective IV

- He must figure out what his tools really do II.
  - Sometimes the tools are buggy
- Debugging your own code or someone else's is a form of detective work.

How to Teach Debugging

- Give the students examples and how you tracked down the problem.
  - Process of elimination
  - Exactly when it happens (corner cases)
  - Instrumented code (poor man's assertions)
- Give them programs with bugs—like the black box in electronics lab.

Basic Principles or Language-of-the-month?

- A big topic on the SIGCSE mailing list last month.
- This is a false dichotomy—you need both!

Why Do You Need the Language?

- You need a way to express and implement the problem and solution.
- You need to implement, to understand principles & techniques.
- Knowing multiple languages helps you understand different paradigms.

Why Do You Need the Language? II

- Knowing multiple languages helps you pick the most appropriate one.
  - "If the only tool you have is a hammer, everything looks like a nail."
  - In other words: The tool affects how we perceive the problem.

Why Do You Need the Language? III

- Knowing multiple languages helps you learn and/or create new languages.
  - Galileo: if I see farther than others, it is because I stand on the shoulders of giants.
  - Knuth: In computer science we are standing on each other's toes.

Why Do You Need the Language? IV

- The sad fact is that you often cannot pick your tools.
  - Compilers not available.
  - Libraries/system calls not available.

Conclusion

- Just Scheme, just Java, just C++, just Pascal, just Ada doesn’t do it. Sometimes a problem will call for Snobol, Perl, HTML, assembler, RPG, COBOL, Fortran, Prolog, or SQL.

Why Do You Need the Principles?

- To know when to apply solutions that have already been worked out. This includes knowing what their limits are.
- To adapt as needed.
- To know what has to be original.
- To get a head start on whatever is original.

Example:

- How to design a protocol which
  - Is robust when requirements change
  - Will be upward- and backward-compatible?

Example, cont.

- Experience has shown several approaches:
  - Type-length-value (e.G. Ipv6 options)
  - Paired tags (e.G. HTML).
- And the reasons why this is desirable (maintain interoperability).

Teaching Is Problem-solving

- . What's the best way to
  - Impart information
  - Find information
  - Recognize patterns
  - Figure out connections
  - Correct misconceptions

Teaching Is Problem-solving II

- How do you adapt your style to help your students succeed?
  - Are you top-down or bottom-up?
  - Are you like a textbook or the web?
- The forward reference problem is not just in compilers!
- Explain why (e.g. no GOTOs).

The Final Ingredient-- Enthusiasm!

- Jerry Feldman: “You have to love this stuff.”
- You felt it as students and feel it now.
- How do you inspire it in your students?

Miscellany

Outcomes Assessment

- Called for in new ABET criteria.
- CSAB is merging with ABET.
- What potential employers and/or graduate schools would like to see.

Internal Documentation

- The bane of any software engineer's existence is poorly-documented code.
- If you don't know what a program module is supposed to do,
  - How do you know if it's right?
  - How could you even write it?

Internal Documentation II

- Internal documentation is as important as external documentation. Sometimes it is more so, because the external documentation lags changes in the code!

Formalisms

- Formalisms are useful, but not sufficient.
  - Structured analysis
  - Use cases
  - Patterns
  - “Process”

Formalisms--Yes

- A formalism helps you
  - Move quickly over the well-understood parts
  - Get into the interesting parts of the project
  - Focus your efforts
  - Avoid missing things.

Formalisms--No

- What's wrong with being too formal?
  - Real world problems don't fit the formalisms.
  - A software system is more than its user interface or its algorithm—it is a series of interacting modules.
  - High cost of tools inhibits both teaching and use.

Is Computer Science Science?

- Yes—for several reasons!
  - Discover, develop, and understand laws
    - Mathematics
    - Complexity
    - Psychology
  - You can do experiments on code.
    - And if you have the source you can tell if you were right!

How is a Software Engineer Different From a Computer Science Researcher?

- More emphasis on immediate applicability.
- Less need to be totally original.
- More constraints on resources
  - time, memory, cost.

How is a Software Engineer Different From a Computer Science Researcher? II

- Less chance to publish
  - proprietary information
  - priorities.
- Less chance for professional development.
- More goodies from vendors!

Things I Want to Fit in Somewhere

- Active/collaborative learning
- Relation to other disciplines/courses.
- Box packing
- Leap year story

Things I Want to Fit in Somewhere II

- Adding semaphore to 7th edition Unix. Now have Linux.
- Division of labor—varies between companies.
- Project topics.