Software Engineering

Shouldn't SoftwareDevelopment be more like engineering? Or is this a case of DisciplineEnvy. Many CS (that "S" is for "Science", as in ComputerScience, another envy) departments have been moved into engineering schools, though that may simply be a tactic to grab industry dollars.
These steps have lead to an engineering discipline ...

- SoftwareProcess
- SoftwareMetrics
- SoftwareHandbook

Have they, though? Most of the process we have in the software world doesn't work very well (and are rarely applied anyway), very few development shops use metrics, the contents of our "handbooks" tend to be highly domain specific and are always out of date.
Check out DavidParnas's claim that "[...]new methods in [Electrical] Engineering were not adopted as a result of the kind of things we are trying in Software Engineering. Engineers use methods if they work. [...] It wasn't necessary for Gauss or Kirchoff to form "User Groups" or write "Ten Commandments" style articles to sell their ideas. The ideas sold themselves. [...] We are putting too much effort into selling methods that are not yet ready. If they were ready, we would not have to sell them."

Software engineering is just like software development, except with the following difference: imagine signing the paper below and giving it to your employer:
I, _____, do hereby pledge that the project I am running
works for the requirements described by my employer.
If it does not work, I accept full personal, financial,
and criminal responsibility.

signed,

_____
Engineer.

That's it. Engineering isn't about design, or architecture, or abstraction, or algorithms, or fiddling with bits. It's about a personal guarantee. Canadian engineering societies weren't formed because people wanted "higher standards of practice" - they were formed because a bridge in Quebec fell down three times, killing many people, and nobody was responsible. Engineering is sucking it up and saying "I'm responsible". Not the compiler vendor, not Microsoft, not your boss, and not your subordinates. Your job is to make sure that everything you use is safe - if Microsoft's module isn't safe, you don't use it. If your employee's code isn't safe, you don't use it. "I didn't know" isn't an excuse. It was your job to know. If your bosses have ordered you to do something that isn't safe - you don't sign off on it. You argue until they either give in or fire you, and if they fire you then you take them to court and prove that they fired you because they wanted you to sign off on something unsafe for their customers, employees, or public.
This means that nobody wants an engineer, or to be an engineer, without a good reason. Engineers are needed where it is _necessary_ - not because it's a neat idea. An engineer is perfectly capable of working in software development, but it isn't engineering until he gives his personal guarantee - otherwise, he's just an engineer working as a software developer.
All the other stuff about standards of practice, conventions, etc. come as a product of this first accountability - not the other way around.

But being responsible is a two-way street: You cannot be responsible if you aren't allowed the freedom to make your own choices, especially as to toolset and materials and the fundamental methodology. Imagine trying to find an engineer to build you a bridge if you demanded all calculations be done on four-function calculators, cheap plywood and glue construction, and a disregard of the accepted best practices. (Best practices are a legal concept, as I understand them: They're a way of saying "I did my best according to what everyone else qualified to judge me says is the right way of doing things.") That is an obvious absurdity and you would never get a competent engineer to sign on to the project. (By definition, any engineer who *would* is incompetent.) So why do we demand responsibility when the current common tools are junk, the current common languages are worse, and the standard methods are a SoftwarePatent minefield? You can tell me SoftwareEngineering exists as something other than a hyperbolic term when the people who create software have as much freedom (including freedom from the WesternWorld's current psychotic patent system) as the people who design bridges.

Where can I go to study software engineering? ComputerScience is great, but I don't want to write a compiler or an operating system. I want to write software (apps and libraries). Do I have to teach myself?
Top tier universities require you to take courses on compilers and operating systems because they realize that such things are just part of understanding what you are doing when you go off and do any kind of software at all. If you write apps and libraries and know nothing of operating systems, compilers, assembly language, etc, then you will continually run into trouble and won't even know why, whereas someone who has learned the proper foundations will not have those difficulties.

If you don't care, and just want to do what you want to do, and screw the consequences, then yes, either teach yourself, or go to a trade school (although they, also, are annoying in trying to teach you more than you might want to know :-) But less so than a university)
*But you'd be cheating yourself. You **cannot** do "software engineering" without learning **all** of the fundamentals. At best you can learn to do a little programming.*
Obligatory quote: "There is no royal road to geometry." (EuclidOfAlexandria) See http://www.vermontgop.org/royal_road.shtml if you are unfamiliar with the quote.

There is a fundamental failure here in definitions. People think "engineering" is prestigious, like design and architecture. They think being a software "engineer" sounds better but don't realize they're completely different things. Engineering isn't about "design", or about "theory". It's about safety, failures, and fault-tolerance. You want to be a "software engineer"? Start programming in AdaLanguage. The difference between an engineering and a designer is personal liability. If a designer's product fails, the worst he can get is fired. If an engineer's product fails, he can lose his license and be sued into oblivion.

An essay on this topic by JimCoplien appears in the July/August issue of the CppReport [what year???]. The main tenet of this essay is that patterns draw on art and the soft sciences and on people issues, thereby providing a strong counterpoint to computer *science* and software *engineering* and their pretense at formalism. We are interested in architecture, and would be well-instructed to heed Rybczynski's formula:

> Engineering + culture = architecture

Patterns help us embrace cultural aesthetics and help us put science and engineering in perspective.
The essay explores the classical architectural literature in depth, and underscores largely unsung Alexandrian principles.

See SoftwareEngineeringBodyOfKnowledge.
Also see TheSourceCodeIsTheDesign for a different view of what a software engineer really does. This viewpoint tends to clash with what the above views of "engineering discipline" would prescribe.

I can't help but see DisciplineEnvy as a huge RedHerring (RedTuna?) brought about by a misconception about other engineering activities. Mature branches of engineering do tend to have big manuals of accepted practice, and quantitative methods in support of those practices. But that isn't what *defines* them. All engineers, the SoftwareEngineer included, use technology to solve economic problems. -- KeithBraithwaite
Perhaps *artifact envy* would be a more correct term. -- WardCunningham - or even *artifact envy*
I'm willing to stick with DisciplineEnvy, because people actually seem to believe their own words when they say "Civil Engineering is/has ..." and "Chemical Engineers do...". The speakers aren't

referring to the artifacts of those professions, they are referring to their own interpretation as to the practices and rules of the named profession.
I had a recent such discussion with a representative from the Provost's office of the university. He simply sneered at the mention of "Computer Engineering", wanting to make a curriculum as "high" as Chemical Engineering. He wasn't looking at artifacts; he was looking at reputation, alumni donations, federal and industry grants.
When I have bumped into civil engineers who build buildings and ships and things, their life struck me as remarkably like ours. Simulations with lots of tests. Overdone engineering procedures that suffocated projects. Made me think that those with DisciplineEnvy haven't really checked the other disciplines. -- AlistairCockburn
My father's a civil engineer, so I know how it goes. But it seems that we're doomed to design software systems in AutoCAD anyway. -- SunirShah
I come from a university named after an engineer which graduates large numbers of engineers. So I have engineering friends. I've worked with some of them on joint hardware and software projects. One thing I have noted is that they tended to draw pictures before they started making things. These pictures tended to be untidy unless it is a large project: e.g., remaking a house. They fitted the paperwork to the project. Lightweight for small projects (on a napkin:-), heavyweight (get out the drawing board) on larger ones. -- DickBotting
Perhaps we just envy the wrong disciplines. See
http://xpdeveloper.com/cgi-bin/wiki.cgi?ParallelsWithFilmMaking

---

Possible **goals** of SoftwareEngineering:

- Create software cheaper
- Create software faster
- Create software that is easy to modify for unanticipated requirements
- Create software without bugs
- Create software that uses fewer resources to get the job done
- Create software to satisfy customer requirements
- Create software that is easy for developers to understand
- Create software that is easy for users to use
- Create software that open new doors (e.g. new business models were made possible by browser technologies)

(Not necessarily in order of importance)

A management goal of software engineering is to create software that can be programmed by non-programmers. Marketing and management can dream up program ideas in 10 minutes, so what can possibly be so hard about making it work? It implicitly recognizes the difficulty of the task - if it actually was easy, they wouldn't have hired programmers in the first place - while disrespecting the value of a programmer's contribution.
I see that attitude a lot from non-programmers. They are used to software that makes it easy to render 3d scenes, spell check, chat online - and they try to extrapolate that to writing software.

They completely miss the fact that all those "easy to do things" are because of programmers. Programmers making easy to use software does not extrapolate into software that makes it easy to replace programmers.
Every now and then someone comes out with a wizard/automation/case tool they say can "generate complex software from a few simple fields" - the part they don't mention is the extremely small scope it can handle. Automated generation of database application? Great! . . . until it has to do something trivial like load a compressed binary file.
The best analogy I can think of is if mechanical engineers had to put up with products that claimed to "automatically generate complex blueprints and schematics, as well as the parts - from a few simple fields!" The real difficulty isn't writing software, it's balancing the mismatch between human nature and hard computer science.
Known as the TheRadBottleneck.

Is creating software even Engineering? I would posit that it is closer to ResearchAndDevelopment than Engineering since in may cases something which has never been built is being created (except, perhaps for small problems or mature programs in their maintenance phase). The key to what makes Engineering work is that in a given problem domain, such as building bridges, solutions templates exists (prefab design and components) which ensure a very high success rate. Software, except for the smallest problems, does not have this. And if it does, it can be largely automated to replicate itself (Such as localized versions of an office application) requiring no skilled human intervention.

Even if solutions templates exist in the case of building briges, you don't blind-fully apply them for reasons of factors such as weather, natural disaster, culture, etc.

SoftwareEngineers are programmers whose jobs are so uninteresting that we give them an ironic title to keep them smiling. -- mt

---

There is SoftwareEngineering, especially in life-critical fields such as medical equipment and fly-by-wire systems. Most software is simply developed rather than engineered because UsersWontPayForQualitySoftware. -- RobMandeville

---

I worked as a contractor on a government project that used a MIL Spec Software Engineering standard. Though, the project was standard business data processing. The part I worked on was a yearlong high-level design. The military users read and formally signed-off on the document. Low-level design was next, then development and testing, user acceptance, etc. They must have paid 5-10 times the cost of a corporate project of similar size and complexity. The same MIL Spec was used to design and build the Saturn rocket, a good spec, misused for the project I participated in.
The contractors working on the project were just programmers, the term Software Engineer wasn't popular at the time.
I believe that tools to help will eventually be developed by someone very clever, and they will address the goals TheRadBottleneck listed. But, in my 35 year career, such tools have rarely

surfaced, only three as far as I know. structured programming, the spreadsheet, and the relational database.
-- EdwinEarlRoss

If programming is at all like engineering, it's most like *Manufacturing Engineering*--designing and building the Rube Goldberg-esque machinery that turns out by the thousands the simple familiar things around us. Designing these things is 60% standards, 10% creative genius and 30% optimization. Make a mistake, and thousands of parts come out the end carrying your error. A small tweak can save a great deal of money or turn out garbage at a high rate. Does all this sound familiar?
Like software, most manufacturing systems are one-of, each one has its own challenges, and the solution to each challenge becomes part of the tradecraft for future designs. Most manufacturing systems also include software, and when you're cutting metal, an error can be very expensive, possibly deadly--so there are no errors. That leaves a question: "What can programming learn from Manufacturing Engineering?"
-- MarcThibault

The two hardest problems in software engineering are managing complex variations-on-a-theme (such as what sub-classing is sometimes used for), and managing the inherently interweaving cross-cutting concerns of the domain or the requirements. The second makes clean or pure modularization difficult since the reality we are modeling usually has "leaky" parts or categories. It's perhaps a form of the ButterflyEffect. Perhaps this is less of an issue in SystemsSoftware than domains involving or modeling business, law, politics, etc. --top
The more effectively a system handles composition -- i.e., the degree to which it is composable -- the more effectively variations-on-a-theme can be handled.
Perhaps this relates to SufficientlyFlexibleAppsResembleInterpreters. However, building such a system is usually not a trivial undertaking, takes experience to do well, and is arguably GoldPlating.
An interpreter where you don't need one is certainly GoldPlating. However, although one way to achieve a certain kind of composable system is to implement it as a language, that's not the only way. It can also be designed as a set of classes or functions and types.
Yes, API's. The devil's in the details.
Yes, an API can provide poorly composable, or highly composable, components.
Note that power-users typically are not going to be using an API, but typically some kind of "rule maker" interface. Making interfaces, API's or rule tools, that are understandable and flexible to future changes is the hard part. Typically one has to know the domain and users well, but there is always the risk that we selected the wrong abstractions or bet wrong about future ChangePatterns.
Indeed, if you put composition in user's hands, the same issues hold true.

Here's a suggestion: LicensedSoftwareEngineers

In practice, all engineering converges onto programming.

See Also CaseTool, IsComputerScience, TheChemicalEngineeringCulture, SoftwareEngineeringIsArtOfCompromise

---

Last edit October 12, 2014, See github about remodeling.