Kitchen Soap

On Being A Senior Engineer

allspaw 6 years ago

I think that there's a lot of institutional knowledge in our field, especially about what makes for a productive engineer. But while there are a good deal of books in the management field about "expert" roles and responsibilities of non-technical individual contributors, I don't see too many modern books or posts that might shed light directly on what makes for a good *senior* engineer. One notable exception is of course Kate Matsudaira, who has been posting quite a good deal recently about the cultural sides of engineering.

Yet at the same time, a good lot of successful engineers whom I have known all remember the mentor who taught them what it meant to be "senior".

I do, however, agree 100% with my friend Theo's words about being "senior" in his chapter of the Web Operations book by O'Reilly:

> "Generation X (and even more so generation Y) are cultures of immediate gratification. I've worked with a staggering number of engineers that expect the "career path" to take them to the highest ranks of the engineering group inside 5 years just because they are smart. This is simply impossible in the staggering numbers I've witnessed. Not everyone can be senior. If, after five years, you are senior, are you at the peak of your game? After five more years will you not have accrued more invaluable experience? What then? "Super engineer"? Five more years? "Super-duper engineer." I blame the youth of our discipline for this affliction. The truth is that there are very few engineers that have been in the field of web operations for fifteen years. Given the dynamics of our industry many elected to move on to managerial positions or risk an entrepreneurial run at things."

He's right: this field of web operations is still quite young. So we can't be surprised when people who have a title of 'senior' exhibit unsurprisingly immature behavior, both technical and non-technical. If you haven't read Theo's chapter, I suggest you do.

Having said that, what does it actually mean to be 'senior' in this discipline? I certainly have an opinion of what it means, given that I'm charged with hiring, supporting, and retaining engineers whom are deemed to be senior. This notion that there is a bar to be passed in terms of career development is a good one, but I'd also add that these criteria exist on a spectrum, as opposed to a simple list of check-boxes. You don't wake up one day and you are "senior" just because your title reflects that upon a promotion. Senior engineers don't know everything. They're not perfect in their technical knowledge, and they're OK with that.

In order not to confuse titles with expectations that are fuzzy, sometimes I'll refer to engineering ***maturity***.

Meaning: I expect a "senior" engineer to be a *mature* engineer.

I'm going to gloss over the part where one could simply list the technical areas in which a mature engineer should have some level of mastery or understanding (such as "Networking", "Filesystems", "Algorithms", etc.) and instead highlight the personal characteristics that in my mind give me indication that someone can influence an organization or a business positively in the domain of engineering.

Over on Quora, someone once asked me "What are the attributes (other than technical ability/experience) that makes a great VP of Technical Operations?". The list of attributes that I mentioned in the answer came with the understanding that they are perpetual aspirations of my own. This post is similar to that answer.

I might first argue that senior engineers in web development and operations have the same characteristics as senior engineers in other fields of engineering (mechanical, electrical, chemical, etc.) in which case The Unwritten Laws of Engineering are applicable. Again, if you haven't read this, please go do so. It was originally written in 1944, published by the American Society of Mechanical Engineers. A good excerpt from the book is here.

While the book's structure and prose still has a dated feel (*"…refrain from using profanity in the workplace…"* or *"…men should pay particular attention to shaving habits and the trimming of beards and mustaches…"*), it gives a good outline of the non-technical expectations, responsibilities, and inner workings of an engineering organization with respect to how both managers and mature engineers might behave.

Obligatory Pithy Characteristics of Mature Engineers

All posts that attempt to give insight to aspirational characteristics must have an over-abundance of bullet points, and the field of engineering has a fair share of them. Therefore, I'm going to give you some, some mine and some pulled from various sources, many from the *Unwritten Laws* mentioned above.

Mature engineers seek out constructive criticism of their designs.

Every successful engineer I've met, upon finishing up a design or getting ready for a project, will continually ask their peers questions along the lines of:

- "What could I be missing?"
- "How will this *not* work?"
- "Will you please shoot as many holes as possible into my thinking on this?"
- "Even if it's technically sound, is it understandable enough for the rest of the organization to operate, troubleshoot, and extend it?"

This is because they know that nothing they make will ever only be in their hands, and that good peer review is what makes better design decisions. As it's been said elsewhere, they "beg for the bad news."

Mature engineers understand the non-technical areas of how they are perceived.

Being able to write a Bloom Filter in Erlang, or write multi-threaded C in your sleep is insufficient. None of that matters if no one wants to work with you. Mature engineers know that no matter how complete, elegant, or superior their designs are, it won't matter if no one wants to work alongside them because they are assholes. Condescension, belittling, narcissism, and ego-boosting behavior send the message to other engineers (maybe tacitly) to stay away. Part of being happy in engineering comes from enjoying the company of the people you work with while designing and building things. An engineer who is quick to call someone a moron is someone destined to stunt his or her career.

This also means that mature engineers have self-awareness when it comes to their communication. This isn't to say that every mature engineer communicates perfectly, only that they have some notion about where they could be better, and continually ask for a gut-check from peers and managers on how they're doing. They aim to be *assertive*, not passive or aggressive in how they get their ideas across.

I've mentioned it elsewhere, but I must emphasize the point more: the degree to which other people want to work with you is a direct indication on how successful you'll be in your career as an engineer. Be the engineer that everyone wants to work with.

Now this isn't to say that you should shy away from giving (or getting) constructive criticism on the **work** produced by engineering (as opposed to the engineer personally), for fear of pissing someone off. There's a difference between calling someone a moron and pointing out faults in their code or product. In a conversation with Theo, he pointed out another possible area where our field may grow up:

> "We as an industry need to (of course) refrain from critiques of human character and condition, but not shy away from critiques of work product. We need to get tougher skin and be able to receive critique through a lens that attempts to eliminate personal focus.
>
> There will be assholes, they should be shunned. But the attitude that someone's code is their baby should come to an end. Code doesn't have feelings, doesn't develop complexes and certainly doesn't exhibit the most important trait (the ability to reproduce) of that which carries for your genetic strains."

See also below #2 and #10 in The Ten Commandments of Egoless Programming.

I think this has a corollary from the *Unwritten Laws* (emphasis mine):

> Be careful about whom you mark for copies of letters, memos, etc., when the interests of other departments are involved.
>
> A lot of mischief has been caused by young people ***broadcasting memorandum containing damaging or embarrassing statements***. Of course it is sometimes difficult for a novice to recognize the "dynamite" in such a document but, in general, it is apt to cause trouble if it steps too heavily upon someone's toes or reveals a serious shortcoming on anybody's part. If it has wide distribution or if it concerns manufacturing or customer difficulties, you'd better get the boss to approve it before it goes out unless you're very sure of your ground.

This of course underscores the dated feel of the book, but in the modern era, I still believe the main point to be true. Nothing indicates that you have a lack of perspective and awareness like a poorly thought out and nonconstructive tweet that slings venomous insults. It's a junior engineer mistake to toss insults about a piece of complex technology in 140 characters.

I certainly (much like Christopher Brown mentioned in his keynote at Velocity London) pay attention to those sorts of public remarks when I come across them so that I can note who I would reconsider hiring if they ever applied to work at Etsy.

Mature engineers do not shy away from making estimates, and are always trying to get better at it.

From the *Unwritten Laws:*

> Promises, schedules, and estimates are necessary and important instruments in a well-ordered business. Many engineers fail to realize this, or habitually try to dodge the irksome responsibility for making commitments. You must make promises based upon your own estimates for the part of the job for which you are responsible, together with estimates obtained from contributing departments for their parts. ***No one should be allowed to avoid the issue by the old formula, "I can't give a promise because it depends upon so many uncertain factors."***

Avoiding responsibility for estimates is another way of saying, "I'm not ready to be relied upon for building critical pieces of infrastructure." All businesses rely on estimates, and all engineers working on a project are involved in Joint Activity, which means that they have a responsibility to others to make themselves *interpredictable*. In general, mature engineers are comfortable with working within some nonzero amount of uncertainty and risk.

Mature engineers have an innate sense of anticipation, even if they don't know they do.

This code looks good, I'm proud of myself. I've asked other people to review it, and I've taken their feedback. Now: how long will it last before it's rewritten? Once it's in production, how will its execution affect resource usage? How much so I expect

CPU/memory/disk/network to increase or decrease? Will others be able to understand this code? Am I making it as easy as I can for others to extend or introspect this work?

Mature engineers understand that not all of their projects are filled with rockstar-on-stage work.

> *However menial and trivial your early assignments may appear, give them your best effort.*

Getting things done means doing things you might not be interested in. No matter how sexy a project is, there are always boring tasks. Tedious tasks. Tasks that a less mature engineer may deem beneath their dignity or their job title. My good friend Kellan Elliot-McCrea (Etsy's CTO) had this to say about it:

> "Sometimes the saving grace of a tedious task is their simplicity and maturity manifests in finishing them quickly and moving on. Sometimes tasks are tedious because they require extreme discipline and malleable attention span. It's an odd phenomena that the most tedious tasks, only to be carried out by the most senior engineers, can also be the most terrifying."

Mature engineers lift the skills and expertise of those around them.

They recognize that at some point, their individual contribution and potential cannot be exercised singularly. They recognize that there is only so much that can be produced by a single person, and the world's best engineering feats are executed by *teams*, not singularly brilliant and lone engineers. Tom Limoncelli makes this point quite well in his post.

At Etsy we call this a "generosity of spirit." Generosity of spirit is one of our core engineering values, but also a primary responsibility of our Staff Engineer position, a career-level position. These engineers spend the time to make sure that more junior or new engineers unfamiliar with the tech or processes we have not only understand what they are doing, but also *why* they are doing it. "Teaching to fish" is a mandatory skill at this level, and that requires having both patience and a perspective of investment in the rest of the organization.

Therefore instead of: "OK, move over, lemme just do it for you", it's instead: "Ok, let's work on this together. I can show you how I'm writing/troubleshooting/etc. Then, you do it so I can be sure you know why/how we're doing it this way, etc."

Related: see below about getting credit.

Mature engineers make their trade-offs explicit when making judgements and decisions.

They realize all engineering decisions, implementations, and designs exist within a spectrum; we do not live in a binary world. They can quickly point out contexts where one successful approach or solution could work and where it could not. They know that one cannot be both *efficient* and *thorough* at the same time (The ETTO Principle), that most projects engineers work on exist on an axis of **optimality** and **brittleness**, and that whether the problems they are solving are **acute** or **chronic**.

They know that they work within a spectrum of ideal and non-ideal, and are *OK* with that. They are comfortable with it because they strive to make the ideal and non-ideal in a design explicit. Later on in the lifecycle of a design, when the original design is not scaling anymore or needs to be replaced or rewritten, they can look back not with a perspective of how short-sighted those earlier decisions were, but instead say "yep, we made it this far with it and knew we'd have to extend or change it at some point. Looks like that time is now, let's get to work!" instead of responding with a cranky-pants, passive-aggressive Hindsight Bias-filled remark with counterfactuals (e.g.. "those idiots didn't do it right the first time!", "they cut corners!", "I TOLD them this wouldn't work!")

Many pithy quotes exist that shine light on this notion of trade-offs, and mature engineers know that there are limits to any philosophy-laden quotes (including the ones I'm writing here):

- "Premature optimization is the root of all evil." – a very abused maxim, and I've written about it before. A corollary to that might be (taken from here) *'Understanding what is and isn't "premature" is what separates senior engineers from junior engineers.'*
- "Right tool for the job" – another abused one. The intention here is reasonable: who wants to use a tool that isn't appropriate? But a rare perspective is that this can be detrimental when taken to the extreme. A carpenter doesn't arm himself with every variation and size of hammer that is available, even thought he may encounter

hammering tasks that could be ideally handled by each one. Why? Because lugging around (and maintaining) a gazillion hammers incurs a cost. As such, decisions on this axis have trade-offs.

The **tl;dr** on trade-offs is that everyone cuts corners, in every project. Immature engineers discover them in hindsight, disgusted. Mature engineers spell them out at the onset of a project, accept them and recognize them as part of good engineering.

(Related: Your Code May Be Elegant, But Mine Fucking Works)

Mature engineers don't practice CYAE ("Cover Your Ass Engineering")

The scenario where someone will stand on ceremony as an excuse for not attempting to understand how his or her code (or infrastructure) could be touched by other parts of the system or business is a losing proposition. Covering your ass sends the implicit message that you are someone willing to throw others (on your team? in your company? in your community?) under the proverbial bus at the mere hint that your work had any flaw. Mature engineers stand up and accept the responsibility given to them. If they find they don't have the requisite authority to be held accountable for their work, they seek out ways to rectify that.

An example of CYAE is "It's not my fault. They broke it, they used it wrong. I built it to spec, I can't be held responsible for their mistakes or improper specification."

Mature engineers are empathetic.

In complex projects, there are usually a number of stakeholders. In any project, the designers, product managers, operations engineers, developers, and business development folks all have goals and perspectives, and mature engineers realize that those goals and views may be different. They understand this so that they can navigate effectively in the work that they do. Being empathetic in this sense means having the ability to view the project from another person's perspective and to take that into consideration into your own work.

Goal conflicts are inherent in all engineering work, and complaining about them (instead of embracing them as requirements for success) is a sign of a less mature engineer.

They don't make empty complaints.

Instead, they express judgements based on empirical evidence and bring with those judgements options for solving the problem which they've identified. A great manager of mine said to never go to your boss with a complaint about anything without at least one (ideally more than one) suggestion for a solution. Even demonstrating that you've tried working the problem on your own and came up empty-handed is better than an empty complaint.

Mature engineers are aware of cognitive biases

This isn't to say that every mature engineer needs to have a degree in psychology, but cognitive biases are what can limit the growth of an engineer's career at a certain point. Even if they're not aware of the details of how they appear or how these biases can be guarded against, most mature engineers I know have a level of self-awareness to at least recognize they (like everyone) are susceptible to them.

Culturally, engineers work day-to-day in empirical evidence in research. Basically: show me the data. The issue with cognitive biases is that we can be blissfully unaware of when we are interpreting data with our own brains in ways that defy empirical data, and can have a surprising effect on how we get work done and work on teams.

A great list of them exists on Wikipedia, but some of the ones that I've seen engineers (including myself) fall prey to are:

- Self-Serving Bias – basically: if something is good, it's probably because of something I did or thought of. If it's bad, it's probably the doing of someone else.
- Fundamental Attribution Error – basically: the bad results that someone else got from his work must have something to do with how he is, personally (stupid, clumsy, sloppy, etc.) whereas if I get bad results, it's because of the context that I was in, the pressure I was under, the situation I was in, etc.
- Hindsight Bias – (it is said that this is the most-studied phenomenon in the history of modern psychology) basically: after an untoward or negative event (a severe bug, an outage, etc.) "I knew it all along!". It is the very strong tendency to view the past more simply than it was in reality. You can tell there is Hindsight Bias going on when descriptions involve counterfactuals, or "…they should have…", or "…how did they not see that, it's so obvious!".
- Outcome Bias – like above, this comes up after a surprising or negative event. If the event was *very* damaging, expensive to clean up, or severe, then the decisions or actions

that contributed to that event are judged to be *very* stupid, reckless, or negligent. The judgement is proportional to how severe the event was.
- Planning Fallacy – (related to the point about making estimates under uncertainty, above) basically: being more optimistic about forecasting the time a particular project will take.

There are plenty of others, all of which I find personally fascinating and I can get lost in learning more about them. Highly suggested reading, if you're at all interested in learning about how you might be limiting your own effectiveness.

The Ten Commandments of Egoless Programming

Appropriate, even if old…I've seen it referenced as coming from The Psychology of Computer Programming, written in 1971, but I don't actually see it in the text. Regardless, here are The Ten Commandments of Egoless Programming, found on @wyattdanger's blog post on receiving advice from his dad:

1. **Understand and accept that you will make mistakes.** The point is to find them early, before they make it into production. Fortunately, except for the few of us developing rocket guidance software at JPL, mistakes are rarely fatal in our industry. We can, and should, learn, laugh, and move on.
2. **You are not your code.** Remember that the entire point of a review is to find problems, and problems will be found. Don't take it personally when one is uncovered. *(Allspaw note – related: see below, number #10, and the points Theo made above.)*
3. **No matter how much "karate" you know, someone else will always know more.** Such an individual can teach you some new moves if you ask. Seek and accept input from others, especially when you think it's not needed.
4. **Don't rewrite code without consultation.** There's a fine line between "fixing code" and "rewriting code." Know the difference, and pursue stylistic changes within the framework of a code review, not as a lone enforcer.
5. **Treat people who know less than you with respect, deference, and patience.** Non-technical people who deal with developers on a regular basis almost universally hold the opinion that we are prima donnas at best and crybabies at worst. Don't reinforce this stereotype with anger and impatience.
6. **The only constant in the world is change. Be open to it and accept it with a smile.** Look at each change to your requirements, platform, or tool as a new challenge, rather than

some serious inconvenience to be fought.

7. **The only true authority stems from knowledge, not from position.** Knowledge engenders authority, and authority engenders respect – so if you want respect in an egoless environment, cultivate knowledge.
8. **Fight for what you believe, but gracefully accept defeat.** Understand that sometimes your ideas will be overruled. Even if you are right, don't take revenge or say "I told you so." Never make your dearly departed idea a martyr or rallying cry.
9. **Don't be "the coder in the corner."** Don't be the person in the dark office emerging only for soda. The coder in the corner is out of sight, out of touch, and out of control. This person has no voice in an open, collaborative environment. Get involved in conversations, and be a participant in your office community.
10. **Critique code instead of people – be kind to the coder, not to the code.** As much as possible, make all of your comments positive and oriented to improving the code. Relate comments to local standards, program specs, increased performance, etc.

Novices versus Experts

Now I generally don't follow too much on knowledge acquisition as a research topic, but I do believe it's hard to get away from when talking about the evolving nature of a discipline. One bit of interesting breakdown comes from a paper from Dreyfus and Dreyfus called "A Five Stage Model of the Mental Activities Involved in Directed Skill Acquisition" which has laid out characteristics of various levels of expertise:

Novice	• Rigid adherence to rules or plans • Little situational perception • No (or limited) discretionary judgment
Advanced Beginner	• Guidelines for action based on attributes and aspects, which are all equal and separate • Limited situational perception
Competent	• Conscious deliberate planning • Standardized and routine procedures
Proficient	• Sees situations holistically rather than as aspects • Perceives deviations from normal patterns

	• Uses maxims for guidance, whose meanings are contextual
Expert	• No longer relies on rules, guidelines or maxims • Intuitive grasp of situations • Analytic approach used only in novel situations

The paper goes on to state:

> *Novices operate from an explicit rules and knowledge-based perspective. They are deliberate and analytical, and therefore slower to take action, they decide or choose.*

(which means that novices are deeply subject to local rationality)

> *Experts operate from a mature, holistic well-tried understanding, intuitively and without conscious deliberation. This is a function of experience. They do not see problems as one thing and solutions as another, they act.*

(which means that experts are context driven)

I don't necessarily subscribe to the notion of such dry lines being drawn between skill levels, because I think that there is a lot more granularity and facets of expertise than just those outlined above, but I think it's helpful to be aware of the unfortunately over-simplified categories.

Dirty secret: mature engineers know the importance of (sometimes irrational) feelings people have. (gasp!)

How people *feel* about technologies, technical decisions, and technical directions is just as important (if not more) than the facts about the details. Mature engineers know this, and adjust accordingly. Again, being empathetic can help you understand how another person on your team feels about a technical decision, even if they themselves don't have an easy time articulating why they feel that way.

People's confidence in software, architectures, or patterns is heavily influenced by past experience, and can result in positive or negative reactions to using them. Used to work at a mod_perl shop that had a lot of mystifying outages? Then you can't be surprised to feel reluctant to use it in a different company, even if the supporting expertise and use cases are entirely different. All you remember is that mod_perl = major headaches, so you're going to be wary of using it in any context again.

Mature engineers understand this phenomenon when making a case to use technology that carries baggage, even if it's irrational. Convincing a group to use tools and patterns that they aren't comfortable with isn't a straightforward task. The "right tool for the job" maxim also has (sometimes unquantifiable) comfortability as a parameter.

For an illustration of how people's emotions drive technical decisions and opinions, read any flame war about anything, ever.

"It is amazing what you can accomplish if you do not care who gets credit."

This quote is commonly attributed to Harry S. Truman, but it looks like it might have first been said by a Jesuit priest in a different form. In any case, this is another indication you're working with a mature engineer: they hold the success of the project much higher than the potential praise they may get personally for working on it. The attribution of praise or credit can be the source of such dysfunction in an engineering-driven organization, and I believe it's because it's largely invisible.

The notion is liberating, and once understood and internalized, a world of progress and innovative thinking can flourish, because the engineer isn't overly concerned with the personal liability of equating the work to their own career success.

Not The End

I'm at the moment blessed to work with a number of mature engineers here at Etsy, and it's quite humbling. We are indeed a young field, and while I think we can learn a great deal from other fields of engineering on this topic, I also think we have an advantage. The web is inextricably tied to the notion of publishing and sharing information, globally. We need to continue pointing out what it means to be a "senior" and "mature" engineer if we have a hope of progressing the field into a true discipline.

Many thanks to members of the Etsy Operations team, Mike Brittain, Kellan Elliott-McCrea, Marc Hedlund, and Theo Schlossnagle for reviewing drafts of this post. They all make me a *more mature engineer.*

Categories: Culture, Etsy, Human Factors, Random, WebOps

Leave a Comment