Hiring and Cultivating Great Developers

by Hal Helms (hal@halhelms.com)

sponsored by Vin 65: www.vin65.com

An Interview with a Software Manager

Me: *I heard you just fired one of your developers -- Brad? May I ask why?*
SM: He just wasn't working out. His work was sub-standard, he wasn't really a team player and he just seemed to lack passion.

Me: *How long had Brad worked for you?*
SM: About 6 months.

Me: *How did you come to hire him?*
SM: He came highly recommended from a friend of one of our developers. So I interviewed him, then passed him on to the team, and we all thought it was going to be a good hire.

Me: *When you say his work was "sub-standard", how did that manifest itself?*
SM: So, a lot of the code he checked in was buggy. It would compile and all -- but it was very fragile.

Me: *Does your company have a testing environment in place? Any policies or procedures?*
SM: No, I don't dictate to developers. That would be insulting. But I expect them to be professional enough to make sure their code works.

Me: *And you said he wasn't much of a team player? What does that mean exactly?*
SM: Well, we have planning meetings for new features and post-mortems to assess how a feature set was implemented. He very rarely said anything. The few times he did got us off-topic discussing things that, though they were related, weren't what we were there to discuss.

Me: *Things like...?*
SM: Um, source control. Requirements. He went off on this thing about "can can" or something -- some Japanese thing.

Me: *Was that "kanban" he was talking about?*
SM: Yeah, maybe. But we needed to discuss how to produce software with features our customers want on-time and on (or under) budget. Not talk about Japanese management fads.

Me: *And then you said you felt he was lacking in passion?*
SM: Definitely. Look, I'm pretty easy-going about keeping hours. I get in around 10.00 or so. I work at home at night (where I definitely get more stuff done) so I come in later. And, of course, I stay later. Most of our developers do. But Brad -- whoosh! Out the door at 5.00 on the dot.

Me: *So, he wasn't working a full forty hours?*
SM: Well, he said he was. He lived a ways away and would come in early. But that's not all of it. He just generally seemed like he wasn't someone who really *cared*. I'd tell him we really needed to hit a certain date for a release. I didn't do that all the time; I know things take time to build, but I had other developers willing to work late or sometimes even on the weekends. Not Brad. And I don't mean like I did this all the time. It was rare. I could count on the other developers more or less, but not on Brad.

Me: *Well, I knowing firing someone is unpleasant. Not as unpleasant as* being *fired, of course...*
SM: Yeah, well, I have high hopes for a new developer I'm going to interview today. With a little more luck than we had with Brad, she'll hopefully be working for us on Monday!

Me: *How do you integrate new developers into the system?*
SM: We start by having them do bug fixes. That gives them a chance to learn our system. Once they've proven themselves with that, we give them more responsibility.

Me: *And what about development of the developers?*
SM: Training, you mean? Yeah, unless it's a bad year, we let people go to a week of training somewhere. It's a reward for their hard work all year long. Lets them know management appreciates their efforts.

Me: *So overall?*
SM: Overall, I'd say we have a good team. I think we're fair with them and they're fair with us. That's what makes Brad's failure hurt. I think we did everything we could for him.

Me: *And are you happy with the progress you're making in getting projects done?*
SM: Well, there's always room for improvement there. Yeah, I guess we could actually do a lot better. I think if we get this new developer onboard, I'll call a team meeting and pump up the volume a bit -- get them charged up.

Me: *Thanks.*

Why Do Developers Fail?

The most important thing I've learned is that failure--and success--are situational. In one context, a developer may be a dud; in another, that same developer is a superstar. So what gives? What set of circumstances makes the same developer either succeed or fail? And can we control those circumstances to enhance the chance for success?

Some people believe that the die is pretty much cast once a developer is hired. Hiring is absolutely essential so ***Hire The Best***(tm). How do we determine the best? Obviously, not resumes. Joel Spolsky argues that we should hire people that are **smart and get things done**.

That's pretty good advice, but how do we know if they're smart? Why, we give them tests. We ask them questions ("*How many manhole covers are there in Seattle*" is a famous question from one of the uber-employers.)

Then put them in front of a whiteboard and have them write code. Run the candidate through a series of tests and, if in doubt, reject them. Hey--it may be cold, but the "Hire the Best" crowd knows that if we get this wrong, all our other efforts will be in vain. So, take the time and make sure they're *smart and get things done*.

Now, if you have any experience managing others, you may see something of a flaw in this way of thinking. It's not that we don't want smart people or people that can actually produce something, but when we look at situations where a developer has been let go, it's very seldom that the problem was that they just weren't smart enough.

"*What? You don't know how to do matrix math? Get out of this office at once!*" Now, if your work requires someone to know matrix math -- or have any esoteric knowledge -- then just ignore this. Nothing to see here. But if you're in the business of building web applications, that's not where failure is going to come from. And while we certainly want smart people, that's frankly only one requirement. Leave Google to hire geniuses; most of us wouldn't have any useful work for them.

So, if failure is seldom because the person wasn't smart enough, maybe they couldn't "get things done"? But how do we know this *before* we hire the person? We could check their open source contributions -- or their StackOverflow reputations.

But that selects for a certain type of developer -- one who has plenty of free time on their hands. And it tells us that they're enhancing their own reputations, but not how they'll work out *for us*. And it means that we'll often exclude anyone not in their 20's and with that exclusion, we'll miss out on the wisdom that comes only from experience.

It's The System, Stupid

The problem with the "*Hire The Best*" notion is that it doesn't work very well. First, what

does “the best” mean? The smartest? The fastest? The best “team player”? And how are we *all* going to hire the best? Isn’t this like Garrison Keiller assuring us that, in his hometown, “all the children are above average”?

While the idea of hiring the best plays to our vanity (hey, after all, *we’re* the best, so we should only hire people like us!), it ignores the importance of *systems* in determining success.

To understand this better, let’s look at some seminal work done by the “man who taught the Japanese about quality”, Edwards Deming. In 1982, Dr. Deming invented a game he called the "Red Bead Game" for his seminars on management.

The game pieces consisted of a paddle resembling a Chinese Checkers board and a container, into which the paddle could be slipped. The paddle could hold 50 marbles; the container held hundreds.

With the game pieces in place, Dr. Deming then asked four volunteers to come on stage. He then explained the rules of the game.

"*We have a vat of white marbles with some red marbles mixed in. Your goal is to dunk the paddle into the vat and come out with white marbles only. The fewer red marbles you get, the better. Ready?*"

Ann is up first. She dunks the paddle, pulls it out and, let's see: 1...2...3: 3 red beads. Dr. Deming looks pleased.

Now, it's Bob's turn: Bob dunks the paddle: 1...2...3...4...5...6...7...8: 8 red beads. Dr. Deming frowns as he enters notes into Bob's personnel file.

Next up is Carla. Her total of red beads is 6. Dr. Deming doesn't look too pleased, but Carla figures she's safe since Bob made such a bad showing.

Finally we have Dave. Dave confidently plunges the paddle in and comes up with...2 red beads! Amazing. Dr. Deming is very happy. Ann feels sick: she was *so close*.

Well, things are looking up for Dave. He'll be given an appropriate reward: "Employee of the Month", perhaps. A nicer parking spot. A gift certificate to a local restaurant. Yes, we need more "Daves" in the world. Perhaps we can hire friends of Dave on the theory that winners tend to associate with winners.

Bob, though -- he'll have to go. We just can't have such low standards. Why, we want only the best! And we better give Carla a written warning to reinforce our desire for all-white beads.

By this point, the audience is laughing -- and groaning. It's absurd, of course: none of the volunteers had any effect on what came out on the paddle -- it was the sheer luck of the draw.

And yet, individuals were getting the praise -- or blame.

Dr. Deming, renowned for sparking the quality revolution that transformed Japan from a producer of cheap goods to a brand of quality, pointed out what should have been obvious: *products are produced by systems*.

A system is the integrated working of people, processes, products -- geared towards producing something. Systems produce what they were designed for. In the case of Dr. Deming's red and white beads game, he left out one crucial detail: the distribution of the beads. 94% were white; 6% were red. Given this, it's simple math to determine what "product" the system would produce. And all the coercing, threatening, cajoling, "incentivizing" in all possible worlds won't have any effect on the product.

That's almost always the case with software development shops -- *i.e.* software development systems. Systems produce the quality of software they were designed to. The problem is we're often unaware that we're part of any system design. A system is the combination of decisions made -- decisions about the environment developers work in, the way requirements are uncovered, the process by which software is developed, the testing procedures in place -- and a myriad of other decisions.

Managers are incredibly important. They have a huge impact -- both on the lives of developers and the quality of software they produce. Too often, managers misunderstand their role: they think they're job is to manage *developers* when actually, the job they need to do is to manage *the system*.

What is this “system” that managers are supposed to be managing? It’s all the components that go into producing the product--software in our case. This is shockingly difficult to do well, which may explain why many managers decide to punt, preferring instead to pursue the chimera of “hiring the best” developers. Developers, though, are only one part (albeit an extremely important one) in a system.

My advice to managers: don’t worry quite so much about hiring the “right” developers. Spend your energy on creating the right system. The system for producing excellent software involves a number of things:

- creating a mechanism for discovering requirements
- breaking software development into discrete, measurable units (*e.g.* user experience, software architecture, algorithmic coding, data persistence, *etc.*)
- providing tools to protect against disaster (e.g. source control)
- developing testing protocols and procedures to prevent bugs from overwhelming the system
- facilitating real communication among developers through things like learning lunches
- motivating programmers to adopt proven best practices, even when those may not be universally loved (pair programming, anyone?)
- focusing native curiosity on subjects that benefit the system by providing in-house

training

This is a very incomplete list and working on the system is never complete, but if you want to produce great software, it's essential to continue to refine and adapt the system. And notably, a truly excellent system does not require either "superstar" programmers or superhuman efforts (working weekends, nights, etc.)

But surely we need good developers? I mean, we're not just producing a machine that can write code on its own? Yes, of course we do. But again, what is a "good" developer? Only the system can determine that -- and managers should hire to fit that system. (For a truly excellent and inspiring analysis of how the Oakland A's do this with baseball, I urge you to read *Moneyball*.)

Having each developer work on their own, self-contained project is not a system. Instead, it's a guarantee of mediocrity. But this seems to be the norm: take a developer, give them a certain project and have them write the HTML, JavaScript, do the system architecture, create APIs, write the algorithms, create data structures and then persist them to a database. Every developer acts as a separate, stand-alone code machine. No economies of scale are possible; no diversification of expertise is gained.

If managers looked at code as nothing more than the *product of the system*, they would understand that "hiring the best" is just a way of abdicating their responsibility. Instead of *hiring* the best, they would invest in *cultivating* the best.

Revisiting the Interview With the Software Manager

Let's take another look at my interview with the software manager -- only this time, I've added a somewhat acerbic software systems consultant (SC) familiar with the situation.

Me: *I heard you just fired one of your developers -- Brad? May I ask why?*
SM: He just wasn't working out. His work was sub-standard, he wasn't really a team player and he just seemed to lack passion.
SC: Uh-oh. I think I'm seeing the red bead game being run.

Me: *How long had Brad worked for you?*
SM: About 6 months.

Me: *How did you come to hire him?*
SM: He came highly recommended from a friend of one of our developers. So I interviewed him, then passed him on to the team, and we all thought it was going to be a good hire.
SC: By "interviewing", he means he chatted him up, then fobbed him off to programmers who were tasked with the impossible job of determining if Brad would or could work in such a chaotic system.

Me: *When you say his work was "sub-standard", how did that manifest itself?*

SM: So, a lot of the code he checked in was buggy. It would compile and all -- but it was very fragile.
SC: Of course, he doesn't mention that there were no testing tools, no policies or procedures in place. Even though his developers had mentioned on multiple occasions that they wanted real testing.
SM: Hey, wait a minute. We're a "fast-paced environment". We don't have time for a ton of rituals on writing code.
SC: So, there's no time to do it right -- but always time to do it over?

Me: *Does your company have a testing environment in place? Any policies or procedures?*
SM: No, I don't dictate to developers. That would be insulting. But I expect them to be professional enough to make sure their code works.
SC: What's insulting to developers is pretending that goading them ever onwards does anything productive. Telling me how much you *need* something done does absolutely nothing to help me do it.

Me: *And you said he wasn't much of a team player? What does that mean exactly?*
SM: Well, we have planning meetings for new features and post-mortems to assess how a feature set was implemented. He very rarely said anything. The few times he did got us off-topic discussing things that, though they were related, weren't what we were there to discuss.
SC: Ah, pass the buck and blame the victim. These are favorite tools of bad managers. No wonder Brad refused to participate in these farces.

Me: *Things like...?*
SM: Um, source control. Requirements. He went off on this thing about "can can" or something -- some Japanese thing.

Me: *Was that "kanban" he was talking about?*
SM: Yeah, maybe. But we needed to discuss how to produce software with features our customers want on-time and on (or under) budget. Not talk about Japanese management fads.
SC: Right. Because "real" developers can write code without knowing exactly what to write. And although they're producing some of the most complicated products humans are capable of (without the benefit of seeing something physical), they don't need tools. Those are just "fads". All they really need is to be told to try harder. What a disappointment that such enlightened management techniques as these so often are associated with failure...

Me: *And then you said you felt he was lacking in passion?*
SM: Definitely. Look, I'm pretty easy-going about keeping hours. I get in around 10.00 or so. I work at home at night (where I definitely get more stuff done) so I come in later. And, of course, I stay later. Most of our developers do. But Brad -- whoosh! Out the door at 5.00 on the dot.
SC: In other words, there are no boundaries. Developers are expected to be available at the pleasure of their manager. Any other responsibilities they might are nothing more than clear indications that they lack "passion".

Me: *So, he wasn't working a full forty hours?*
SM: Well, he said he was. He lived a ways away and would come in early. But that's not all of it. He just generally seemed like he wasn't someone who really *cared*. I'd tell him we really needed to hit a certain date for a release. I didn't do that all the time; I know things take time to build, but I had other developers willing to work late or sometimes even on the weekends. Not Brad. And I don't mean like I did this all the time. It was rare. I could count on the other developers more or less, but not on Brad.
SC: So, he succeeded in brow-beating the other developers into donating their free time to try to make up for a complete failure of management -- only Brad wouldn't play ball. Did it ever occur to him to wonder why the system failed so completely that extraordinary (and unsustainable) efforts were required to circumvent it and actually produce what the system failed to?

Me: *Well, I knowing firing someone is unpleasant. Not as unpleasant as* being *fired, of course...*
SM: Yeah, well, I have high hopes for a new developer I'm going to interview today. With a little more luck than we had with Brad, she'll hopefully be working for us on Monday!
SC: On Monday. Really? So, off with Brad's head. Let's bring up a new candidate for future execution. We won't make any real efforts to see if this is a good fit for the person. Why bother when we can just hide our mistakes by firing perfectly capable programmers?

Me: *How do you integrate new developers into the system?*
SM: We start by having them do bug fixes. That gives them a chance to learn our system. Once they've proven themselves with that, we give them more responsibility.
SC: So, let's take the person who knows the *least* about the current code and then give them the task of squashing the bugs that have eluded those who know the code best? Never mind that some programmers work better in some phases of development far better than others (an architect typically makes a lousy maintenance programmer -- and *vice versa*). This should be humorous; instead, it's all too common.

Me: *And what about development of the developers?*
SM: Training, you mean? Yeah, unless it's a bad year, we let people go to a week of training somewhere. It's a reward for their hard work all year long. Lets them know management appreciates their efforts.
SC: Well, if he sees it as a reward, instead of an investment in elevating the capabilities of the system, he'll spend as little as possible and do it even more grudgingly. Instead, he should understand the areas in which the system needs improvement and bring in experts and trainers to provide empowering knowledge to his programmers. That would give them real job satisfaction while simultaneously ensuring that the system was constantly improving.

Me: *So overall?*
SM: Overall, I'd say we have a good team. I think we're fair with them and they're fair with us.

That's what makes Brad's failure hurt. I think we did everything we could for him.
SC: Translation: Everyone's on their own. If too many red beads come out, you're done for.

Me: *And are you happy with the progress you're making in getting projects done?*
SM: Well, there's always room for improvement there. Yeah, I guess we could actually do a lot better. I think if we get this new developer onboard, I'll call a team meeting and pump up the volume a bit -- get them charged up.
SC: Right--because the real problem is that the developers don't have enough impossible tasks to do. Clearly what they need is to *try harder*. Too many red beads! Try harder!

Me: *Thanks.*
SC: For nothing.

What's a Boss To Do?

While I've tried to stress how important the *system* is, it's also true that the system is composed of individual parts -- and we want to make sure the parts fit well. When it comes to hiring, what should we do?

Let's leave the number of manhole covers to municipal engineers. Here is a simple **5-part plan** that should help identify programmers that you want as part of your system.

Step 1: Fizz-Buzz

Having weeded through resumes (and discounting most of what they say), you're ready to interview a candidate. Blah blah. Did this. Blah blah. Did that. OK, let's give them a nice, simple test. This isn't meant to stymie anyone; it's just there to weed out the people who really can't program.

The idea comes from a blog post by a developer known simply as "Imran". Here's what he has to say about it:

"*After a fair bit of trial and error I've come to discover that people who struggle to code don't just struggle on big problems, or even smallish problems (i.e. write a implementation of a linked list). They struggle with tiny problems.*

So I set out to develop questions that can identify this kind of developer and came up with a class of questions I call "FizzBuzz Questions" named after a game children often play (or are made to play) in schools in the UK."

Here's an example Fizz-Buzz question:

Write a program that prints the numbers from 1 to 100. But, for multiples of three, print “Fizz” instead of the number and, for the multiples of five, print “Buzz”. For numbers which are multiples of *both* three and five print “FizzBuzz”.

In step 1, we give the candidate this simple of a test. You can leave them alone: pressure makes some people seize up. We’re testing for basic skills here, not the ability to withstand pressure.

BTW, the popularity of this particular Fizz-Buzz question might mean that you shouldn’t use it, as your candidate may have already worked through it. But be careful in devising your own that you keep it *just as simple* as this one.

If the candidate just can’t do it, you thank them for their time and move on to the next one.

Step 2: Fix This Code

Having successfully fizz-buzzed, I suggest a short test where you give the candidate some code that has some errors in it. Again, we’re not trying to play “Stump the Chump”. We just want to get a sense for how comfortable the candidate is with the basics of the language -- and to make sure they can analyze reasonably-written code and understand (a) what it’s trying to do, and (b) why it’s failing to do so.

This step should take the candidate 20-30 minutes. Unlike the go/no-go aspect of the first step, “Fix This Code” is just to help you evaluate the candidate’s strengths and weaknesses in one very particular aspect. But this is very sensitive to how people respond to pressure. I’ve known some excellent programmers who, when put on the spot with something like this, freeze and just can’t think.

“Gotcha” interviewers will use this to disqualify the person. I think that’s a mistake. Unless you plan on your employees entering some global competition to save the galaxy from algorithmic aliens, the ability to perform “on the spot” just isn’t that valuable. My advice: leave them alone in a quiet room and tell them to come get you when they’re done.

Step 3: Communication

Step 3 asks the applicant to write something. What’s the “something”? It should be related to the position you’re hiring for. (If you don’t know *specifically* what you’re hiring for, you shouldn’t be hiring at all; you should be determining what area of the *system* will an added person will strengthen.) Need a project manager? Then ask them to do a 1-2 page writeup outlining the steps needed to complete some specific project. Does the job involve heavy database lifting? Perhaps you could ask them to outline the major points around the SQL/No-SQL debate.

You want to learn two things here: how does the applicant think, and how clearly does s/he

communicate it? Of the two, frankly, communication is more important. A lowered ability to communicate impedes *everything*. I'm not interested in how gregarious they are or whether they're intro- or extroverts. But can they clearly get across their points in a way that *others* can understand?

How important is this? Extraordinarily important. If we go back to the first question, "Why do developers fail?", I'd say the most common reason is because they cannot communicate. In the system you're building, communication is like the oil that makes all the parts run smoothly. Remove that and your system becomes slower and produces worse product. After a time, it may not run at all.

Now, I anticipate this step will be the one most likely to be skipped. "Eh, we're hiring programmers, not novelists. Leave that touchy-feely stuff to the marketing people." And since people interviewing developer candidates may themselves not communicate well (volumes could be written on this!), it will be easy to brush this step aside. But if your goal is to hire developers that you can work with to build a truly world-class system, I don't believe you can get there without good, clear communication.

Is this a go/no-go step? Maybe. At least, it would give me very considerable concern if the candidate simply can't write clearly. (For language geeks, no points off for bad grammar or spelling! We're trying to evaluate communication skills and yes, it's annoying that so many people are insensitive to the differences between "their", "there", and "they're", but it's not germane to the process.)

Step 4: Pair Programming

"But we don't do pair programming!" you say? Humor me and do it for the purpose of finding the right developer for your position. (Oh--and your competitors thank you for not extending the practice into your system.)

Our goal is to have the candidate work with an experienced programmer -- most preferably someone who has had some real-world experience with pair programming. Give the candidate a choice among 4-5 areas to work on. Each one should take between 2-3 hours.

Begin with your experienced programmer "driving". (For the sake of discussion, let's assume that to be you.) As a candidate, it can be pretty unnerving to be put in the hot seat. Your goal in interviewing (as in managing) should be to *drive fear out of the system*. So, you drive for the first 15 minutes or so. During this time, you want to encourage simple conversation. How should you (plural) approach the problem? Did either of you ever run into something similar previously? Any lessons to be learned from that experience?

At some point, ask the other person to drive. Continue the conversation as before. In real pair-programming, the team acts as one. You want to encourage that here, also. After, perhaps, 20-30 minutes, volunteer to drive again. (Make sure your candidate understands that there's an ebb and flow to pair programming between the two individuals and that you're not seizing

control because you're horrified by the code they've been writing!)

This is not the time for a quiz. This is you working with a fellow developer on a problem. As a member of a paired team, you are partners. Don't do anything other than try to be a good member of the team. Hopefully, the experience will be good for both of you. Just be vigilant not to turn this into an interview or make the candidate feel they're being grilled. That's not what this is about.

Step 5: Evaluation

After Step 4 is complete, thank the candidate for their time. (In fact, if you wanted to *pay* the candidate for the pair programming session, I think that would show an admirable respect for the candidate's time. The candidate is providing value by helping you improve your system. In our society money equates to value in commercial settings. Why *not* pay them?)

Now, immediately write down your impressions of the pair programming session. Was the candidate communicative? Did they seem to know their way around the language? Did they come up with any surprising insights? Were they enjoyable to work with? Were there any red flags you sensed -- even if you are not able to fully articulate your concerns?

One thing to be extremely cautious about: *the cloning trap*. It is natural for you to hire someone like -- you! But unless what they system calls for is another you, be aware of -- and resist -- this tendency. Remember: you are hiring for a specific role in your system, not necessarily someone who has the same good taste and excellent qualities that you so clearly possess!

If others have been involved with the candidate, ask for their feedback. Some managers abdicate the decision about hiring to "the team". Yes, nice, but "the team" isn't charged with improving the system: you are. So do your job. Get feedback from others, but you must make the decision about hiring and that decision must always be about improving the system, not about whether everyone like the person.

Congratulations, You're Hired! Now, Go Home -- and Other Closing Thoughts

One of the most brilliant, non-intuitive ideas on integrating new-hires into the system comes from Zappos' Tony Hsieh. After a four-week training period, Zappos offers new-hires $3,000 *to quit*!

"We don't want people at Zappos that are there just for a paycheck." Now, this is a very different attitude from the software manager in the interview above who demands "passion". Zappos actually *pays* people *not* to have passion. Why? Because only the truly passionate will turn him down.

Do I recommend this? Absolutely. I think it's sheer brilliance.

So, having hired someone who fits well into your system, what's the next step? Improving the system! Again.

There are two aspects to this: analysis of the system as a whole and optimizing the individual parts. Since developers are parts of the system, you want them to be productive and enjoy their work.

So, ask them what sort of things they're interested in. Since developers tend to be (a) naturally curious and (b) interested in technology, your job as manager is to find a way to fit their interests into your system. Does your new-hire really care about client-side programming, for example? I'm pretty sure there's a need for that in your system.

And, as a manager, you have a unique opportunity to focus their curiosity by providing training, books, etc. Are you and the developers located in the same physical environment? I have a great way of capturing your employees time for only $10/hr! Yes, create a *learning lunch*. Ask one developer to prepare a talk on some aspect of technology they're interested in, then have them present at a learning lunch. Developers are free to attend or not, but you're buying lunch if they attend. A *nice* lunch. You want to entice them to come, right?

Of course, lots of questions will arise. The best advice I can offer you is this: consider whether any action improves the system or not. If it does, adopt it. If not, no matter how nice it sounds, don't. You'll need all your energy and resources for the task of turning a bunch of people into a well-oiled system.

Speaking of "well-oiled systems", some might ask: isn't this impersonal? Surely developers are people and not just "parts of the system". Well, of course they are. As a matter of fact, we all are. People, that is. And people play different roles. The same person may be a child, parent, sibling, spouse, friend, consultant, employee -- the list is long and ever-changing.

What I'm talking about is creating an environment in which a person *in their role as employee/ developer* can flourish. In whatever role we inhabit, we want to excel. We want to be a *good* friend, a *loving* spouse, a *valued* employee. All I'm urging is that managers do their jobs and create that environment. Truly, your employees don't need "motivational meetings" any more than they need you hectoring them to "commit" to a deadline. Do you really think that, absent this deadline, they'll be goofing off -- or that by browbeating them into submission, you've done anything other than been abusive?

But I digress. That's, perhaps, a topic for another day.

Resources

Hopefully, the idea of building an excellent system in which all parties can flourish is exciting to you. If so, here are a few resources that I very highly recommend:

- *The E-Myth* and *The E-Myth Revisited* by Michael Gerber. His website is www.e-myth.com.

- Anything and everything by Edwards Deming. His insights are truly revelatory. While some of Dr. Deming's work can be challenging, he has many interpreters. A good place to start would by Mary Walton's *The Deming Management Method*.

- Works by Eli Goldratt, father of a management philosophy known as the *Theory of Constraints.* Goldratt has written several popular "business novels". No, the story and prose aren't great, but I highly recommend them for the quality and uniqueness of their messages. The two I'd start with are *The Goal* and *Critical Chain*.

A special note of thanks is due to Vin 65 who commissioned this piece -- and then generously allowed me to share it with others.