

THE MISSING FACTORY

CHAPTER 5



The Missing Factory

Despite the evidence presented previously pertaining to software development as a creative art form, there is an aspect of software development that is pure manufacturing. The immature (and at times, non-existent) software manufacturing processes that I encounter in organizations I consult with constantly appalls me. I find this to be true in organizations of all sizes, despite the lip service that everyone pays to ISO 9000 and the Capability Maturity Model. Not that I am a big believer in heavyweight processes that force developers to produce more paper than work product—far from it. However, the cavalier manner in which most software is manufactured would make Henry Ford roll over in his grave.

The Road Less Traveled Starts Here

This chapter is by far the "techiest" of this text but paradoxically "business people" are likely to derive the most value from it. Why? Because software initiatives are, and will continue to be, the prime movers in the endless pursuit of competitive advantage and therefore it is incumbent on all stakeholders within the

organization to improve their literacy. Your literacy improvement course starts here. The intent is not to turn business people into technologists, but instead to encourage them to arm themselves in order to better interact with the geeks down the hall.

Why a Factory?

Even though I am a big believer in software as an art form, once the inspiration has been codified in silicon, automation needs to kick in. This is necessary both to ensure quality and to protect our intellectual assets.

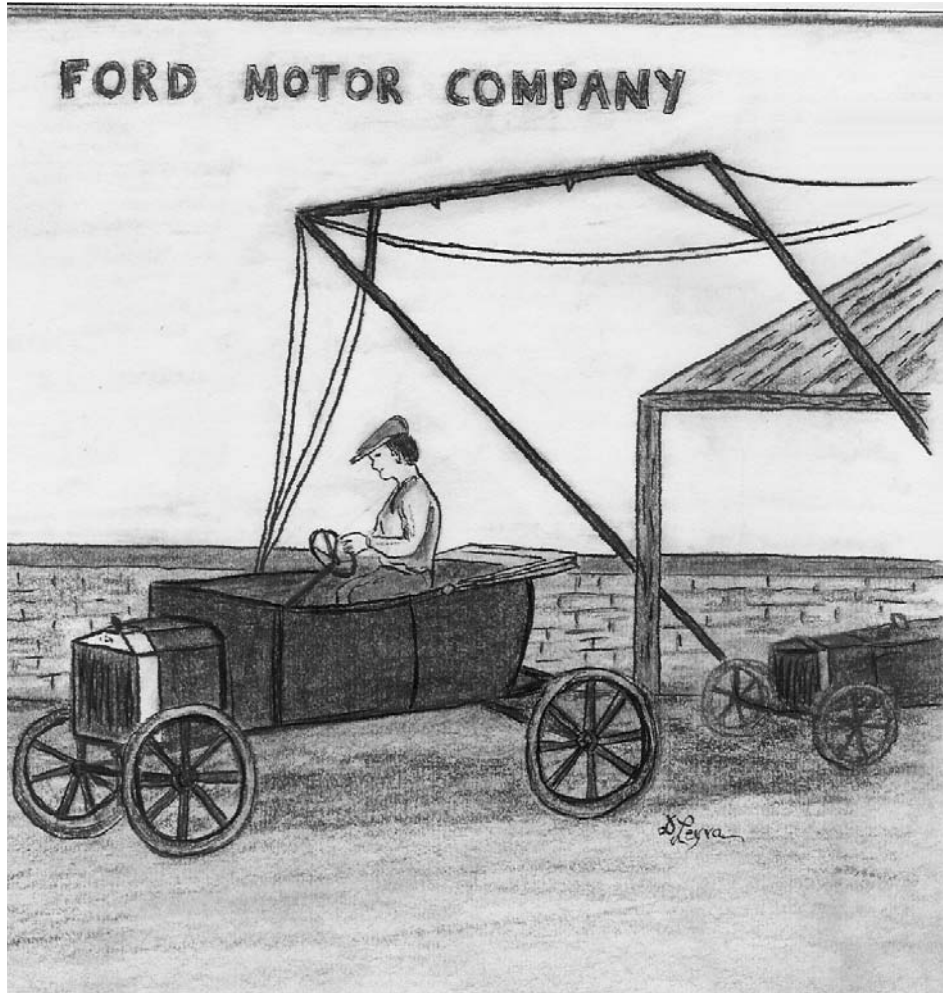
Most developers understand the basics of how things should be done, but they are often not given the time to establish adequate factory processes. In addition, they are rarely given the funds needed to buy the tools that they require, despite widespread evidence (anecdotal and otherwise) that there are tools available in the marketplace that could dramatically improve their productivity. Or worse yet, they are given tools that have been purchased, often at significant expense, but have been permanently relegated to “shelfware,” since no one has time to learn how to use them.

Assembly Line Tools & Processes

First of all, let me say, fuck ISO 9000 and the Capability Maturity Model. I am not so much interested in attaining a state of automation nirvana as I am in ensuring that the organizations I work with at least have the basics covered. Besides, I believe more in the common sense of developers than I do in a bunch of aging bureaucrats who haven't done anything real in the last thirty years.

A Slight Detour

If you live in the valley, you are probably rolling your eyes back in your head by now. I know that you find this shit boring. Even though y'all (I was raised in “Nawlins” remember) think you're God's gift to silicon, most software is developed elsewhere. So, fuck off; this chapter ain't for you. Why don't you go



The Assembly Line

outside and polish your BMWs, or go to Starbucks and get a double shot of espresso, or hug a fucking tree?

Hey, if you were really bad asses, then you would go hack Bill's security system or paint graffiti on Larry's yacht, just for sport. Now these are no lightweight challenges. You don't think that Bill is stupid enough to use his own weak shit on something as important as protecting Mamma Bill and Baby Bill? Nah, I'd be willing to bet that it is Linux and Java based security. As for Larry, well he's got



Hug A Tree

an “in” with the Lord, so you won’t be touching that yacht anytime soon. Talk about kick ass security.

In fact, I overheard Larry and the Lord having a conversation, and the Lord told Larry that he is tired of him talking trash and carrying on. The Lord said, “Prove to me that you’re no fool; walk across Bill’s swimming pool.” Larry said, “Not only will I do it, I’ll do it on prime time television and rake in a billion greenbacks on your behalf. I will do this in your honor, Lord, because even though Bill has more money, everyone knows that I have gotten laid more often because I’m way fucking better looking.”

The Lord watched in amazement as Larry walked across the pool, and then said, “That’s nice Larry, and I appreciate the billion greenbacks. I’m getting tired of watching these television evangelists milk the poor for money. But, I was just commenting to Scott that even though you *are* better looking than Bill, and you *have* gotten laid more often, most of the innovation these days appears to be coming out of the Northwest. Bill was blessed with far more imagination. The last time I checked, the database was the last good idea you had.”

“Of course, Scott just lost that shit eating grin he’s always wearing and said to me, ‘Lord, what do you mean by saying that *most* of the innovation appears to be coming out of the Northwest?’ I replied, ‘Scott, quit whining, you are still a box peddler and a lesser evolved species. You missed the opportunity to ride the Java train to the next evolutionary plateau. You lack Larry’s good looks and Bill’s imagination. Just run along and be a nice little yuppie. I hear Jack’s got more time on his hands these days so maybe you can squeeze in a few rounds.’”

Getting Back On Track

Okay, I must confess that I find this shit boring and painful too. I have never known a developer that gets excited about factory processes. It falls into the category of necessary evils, like performing paperwork after defecating or paying taxes. Well, I take that back, developers that sell products into this space often become religious about it, and rightfully so, since they tend to do quite well, thank you very much.

If most factories weren't in such a state of disarray, this chapter would not be necessary. However, I feel compelled to discuss Factory 101 basics and add my name to the long list of people whom the development community hates for pointing out things they know they *should* be doing, but often don't. If you happen to be the manager that has been tasked with factory building, then do not despair; the web contains all the raw materials you need to put a working factory together in fairly short order.

In the rest of this chapter, I intend to provide high-level process insights, while at the same time preventing the chapter from turning into a manual. Most product references are specific to the Microsoft universe, since that is where most of my time has been spent in the last few years. However, there are certainly analogous products available from the Java/Unix/Linux camp.

Source Code Management (SCM)

Managing source code is one of your most important job responsibilities as a developer, project lead, or manager. Anytime you have more than one developer working on a project, then source code management (using a commercial grade SCM product) is a must. There are a number of relatively inexpensive (and some free) products available, so cost should not be a problem. In the Microsoft universe for example, Visual SourceSafe is part of Visual Studio and essentially free.

There are two basic considerations when setting up a source code management process. The first is to define a set of roles and responsibilities:

- Developer
- Quality Assurance Engineer
- Project Lead
- Release Engineer
- Manager
- Administrator

The second is to define a set of environments that will be supported:

- Development
- Quality Assurance
- Production
- Release

Each environment comprises a distinct code repository. Security access should be granted to specific environments based on an individual's role. A formal procedure with signoffs should be instituted for moving code between repositories. The movement of code between Development and Quality Assurance is usually where organizational standards are enforced (e.g. code reviews).

Discussions concerning source code-naming standards (for variables, methods, etc.) often stir up religious fervor among opposing factions. All of the leading commercial software providers have documentation available on their web sites, and their recommended approaches are clearly presented. The most important thing to do is to select one of the recommended approaches and consistently adhere to it. There is absolutely no justification for rolling your own.

I am a big believer in the value of in-source documentation, both at the module/class level and at the method level. Again, the exact template that you decide on is not nearly as important as getting developers to actually do it. The only feasible way that I have found to encourage this practice is to convince developers that if they get in the habit of doing it as they go, it is not nearly as

painful as they might imagine. Adding in source documentation after the fact, on the other hand, is extremely painful and almost never gets done.

Incident Management

What is an incident? An “incident” can be a bug that has to be fixed, a feature that has to be implemented, a customer question that has to be answered, or any other important item that has to be tracked until it is dealt with. Associated with each incident is a general description of the issue, a priority, a person to whom it is assigned, and a status. In addition, an incident may be associated with one or more contacts (e.g. customers) or files (e.g. documentation or source code). Finally, an incident is associated with a specific project.

A product such as *Visual Intercept by Elsinore Technologies* is best in its class, and is so tightly integrated with the Visual Studio IDE that it is indistinguishable from one of Microsoft’s own offerings. An incident management product is the “glue” that holds the development process together, and it is an absolute requirement for all but the most trivial of projects.

Effective management of source code and incidents, using world-class tools and processes, make up 80% of the critical “must have” factory components! Homegrown alternatives should no longer be tolerated as acceptable replacements.

Backup and Recovery

Backup and recovery represents a set of strategies that determine how quickly an organization can respond after a disaster strikes. A disaster can be anything from a hard disk failure on your one and only mission-critical web server to an act of God (i.e. tornado, hurricane, flood, etc.) that brings the site down. In order to minimize downtime, many factors need to be considered:

- Backups
- Offsite Storage

- Redundant Sites
- Transition Processes
- Recovery Processes

Usually, what drives the decision is cost versus the amount of downtime that the organization can tolerate. It is my experience that recoverability can be improved dramatically simply by focusing on low cost, but effective, best practices. These practices tend to focus on process issues requiring minimal incremental capital expenditures. Providing absolute recoverability requires increased complexity and cost, and it is usually not warranted for work performed by development groups. However, if you are responsible for E*TRADE's or Amazon's production site, then that is a completely different story.

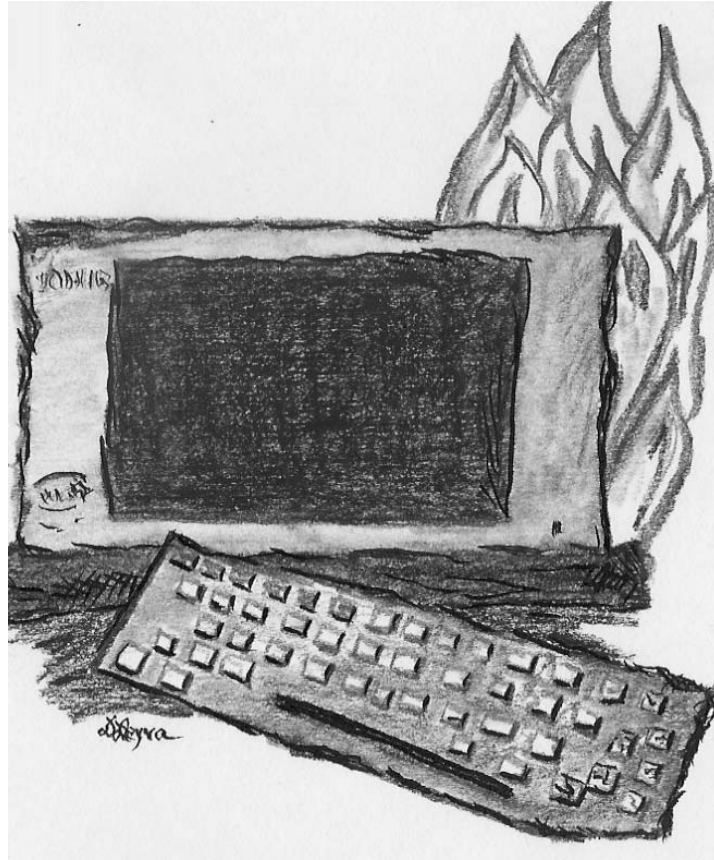
Here is a list of common sense practices:

- Make a full backup once a week.
- Make incremental backups daily.
- Store backups in a fire proof safe.
- Rotate backups offsite once a week (or at least once a month).
- Document the shit out of the process.
- Ensure that you have more than one responsible party.

Once you have a working process, destroy shit on purpose from time to time just to ensure that it is actually working. Obviously, this should be done in a controlled manner, but it must be done. The most elegant backup and recovery procedures can develop insidious bugs that go undetected for long periods of time. The only way to prevent this from happening is to test it often.

Tools of the Trade

I am always surprised by the fact that even great developers often do not insist that the organizations they work for purchase the necessary tools that would make them even better and more productive. Debuggers are, of course, part of everyone's toolbox. However, other tools—such as memory corruption detectors,



Backup and Recovery

source code coverage, and performance profilers—go virtually unused. Part of the reason for this is that developers constantly face time crunches, and the last thing they need is yet another learning curve to climb.

The way around this quandary is to designate one team member as the “toolsmith.” This individual would be responsible for learning the innards of each tool within the toolbox and disseminating this knowledge to his or her partners in crime.

Documentation

If there is anything that developers hate worse than documentation, I have yet to discover what it is. Most developers would prefer Chinese water torture to writing a document in a natural language. I have, through trial and error, hit on a strategy that I have used with considerable success:

- Establish a Process
- Minimize the Pain
- Hire the Pros

Establish a process that outlines the level and quality of documentation that is required. Make it clear to the team that you do not expect a Pulitzer Prize winning novel, or even a document with perfectly correct grammar. What you *do* want is a brain dump that will be used as input to the documentation assembly line. Develop a standard set of templates that remove formatting issues from being a discretionary consideration. Provide basic training related to screen capture utilities, MS Visio, MS PowerPoint and any other tools required to do the job. Then, hire professional technical writers to take this raw input and produce a professional text.

The Shipping Department

By shipping department, I mean everything required for getting the actual physical product out the door. Some of the tasks include:

- Creating the Installation Programs and Media
- Packaging
- Managing Product Licensing
- Logistics

There are a host of other vitally important details that need to be tended to as well. However, my only recommendation in this area is to hire the youth. Hire really bright young individuals who can get excited about this kind of stuff

because they perceive it to be a stepping-stone into the biz. Help them create a working process, and then celebrate their contributions.

1-800-TechSupport

Technical support in this day and age, when it is often difficult to find a human being to talk to when problems arise (as they always do), is an area that should be perceived as an opportunity to derive competitive advantage. Do not staff this function with your weakest players. I am of the opinion that everyone should be required to do time in support. Not only can it be a personally rewarding experience, it is often one of the best opportunities for the geeks among us to learn valuable interpersonal skills.

Parting Thoughts

Factories are judged by the quality of the products they ship. This is not a monument building exercise. Develop and institute these processes, and then get on with the business at hand. If you implement the basics described in this chapter, you will be ahead of most of your competition and equal to the rest. If you choose to take it to the next level, then you are a sick individual who has way too much time on your hands! I would never hire you.

This chapter has focused on factory best practices with respect to software development. Many of these ideas are applicable to other domains wherein soft goods are produced (e.g. the publishing business). Therefore, you should consider adopting those ideas that you find useful.