Alex Groce (agroce@gmail.com), Northern Arizona University

Tom DeMarco and Timothy Lister's *Peopleware: Productive Projects and Teams (Third Edition)* is a fairly famous book. The first edition was published in 1987, and it has been widely read ever since. I reviewed DeMarco and Lister's *Waltzing with Bears* in this column before *Peopleware* precisely because some people might not be aware of *Waltzing*, while I assumed most people reading SEN would know of, and have read *Peopleware*.

This was, arguably, hypocritical. You see, I just finished reading *Peopleware* for the first time, myself, about ten minutes before starting to write this column. I knew of the book, and knew it was widely admired, but I also knew it was strongly focused on how to manage people, and the proposal that software development projects' problems are "not so much *technological* as *sociological* in nature." Now, I don't have any disagreement with that claim; it seems to me that it is likely not self-evident, but largely true and too-often ignored. So, in principle, I'm on board with *Peopleware*. On the other hand, risk management (the topic of *Waltzing with Bears*) involves lots of nice graphs, uses some probability theory, and is personally interesting to me, while the actual mechanics of how to manage people are, while not inherently boring, much lower on my personal list of topic of interest. For one thing, I am (my graduate students, former and present, can attest to this) possibly a nice guy, but not much of a manager; I don't know if I'm a bad manager, because I'm really more of a non-manager. Actual management actions on my part (actions such as motivation beyond "here's an email on a cool problem" or "remember the deadline is tomorrow", or the holding of a meeting that is not strictly required by circumstances or pushed on me by someone else) are so rare as to be hard to evaluate. It seemed likely that *Peopleware* would either give advice I'd find irrelevant, or advice I would be motivated not to take, because I'd much rather read papers, walk around and think about path coverage and code mutants, write code, or run experiments than do anything resembling management. It is possible, perhaps nearly obligatory, to become more of a manager as one settles into being a professor, but I try to avoid doing that kind of thing as much as possible. Even if it's a good idea, I don't like doing it, and so far have gotten away with this. I finally read it, because I decided it's such an important book that not including it in *Passages* is a dis-service to people who do want to manage projects well, but may not be "up on the literature."

So, how's the book? Did I enjoy it, despite being in some sense only modestly (within the bounds of my general intellectual curiosity) interested in the topic? Yes. I enjoyed it very much. DeMarco and Lister are engaging writers, and unless you are much more hostile to the topic than I am, you will probably enjoy this book, especially since it is relatively short, so does not over-stay its welcome.

What's the book about? Well, as you may have noticed, it's about the "people" side of software engineering. It offers almost no "technical" ideas, no magic methodology (this is a fairly methodology-hostile book, in fact), and in fact spends some time pooh-poohing tool-based improvements in software project productivity. My impression, having just finished it, is that the book has a few basic things to say. I'll list the ones that stick in my mind in no particular order:

- Noisy work environments are horrible. Uniform cubicle farms without privacy are cruel and unusual punishment, resulting in poor performance. The "Furniture Police" and insecure managers want uniform environments where windows are reserved for People Not Doing the Actual Work.
- Managers tend to think too highly of their own role, imagining "Leadership!!" and life as the Vince Lombardi writ small. In fact, managers need to figure out what they actually do well, and let their people do the actual work.
- Interruptions are disastrous, and working modes that involve never focusing intensely on anything are simply Wrong for work that involves thought. Most software development involves thought, and if it doesn't, that's bad.
- Meetings fall into working meetings, which are useful and easy to identify, and ceremonies; ceremonies are mostly harmful.
- Turnover is murder. Pray you avoid it.
- The presence of motivational posters is a Bad Sign, and insulting to people.
- Teams form and "jell" for reasons more complex, human, and interesting than a simple strategy can control, but you can probably do some things to avoid preventing this.
- Email's good, unless you send and receive lots of pointless emails, in which case it is bad, but really it's probably not as bad as the phone, the worst thing ever invented.
- People being managed should not be insulted and degraded, and in the kind of work involved in most software development, if they have to be flogged or manipulated into doing good work, you probably will not get good work out of them.
- Hire good people, and keep them (did you see that turnover is murder? It's murder.)

Some of these things are conventional wisdom now; some are still, approximately 30 years after 1987, for some reason, not widely accepted. I'm a tenured professor, so there is zero chance anyone will visit an open-plan office on *me*, but you may not be so lucky. Despite what appears to me to be good evidence that open-plan offices (the opposite of what this book endorses: offices with 2-3 people and a nice window!) are bad for productivity and possibly a violation of the Geneva Convention, they appear to be on the rise. So it goes.

If you do manage people, you will probably enjoy the book even more than I did. If you are managed (which, again, most professors basically are not, which is to our management's credit), you may enjoy the book even more than I did. You might be inspired to launch a revolt or quit your job (the former is sometimes a good idea, the latter is often a good idea). But even if none of this applies to you, you will likely enjoy this book *even though it may not really improve your software engineering skills in any direct way at all*. This is because the authors are masterful at introducing interesting anecdotes and compelling analogies, and know a lot of things. For example, the most enjoyable thing for me in *Peopleware* is the story of IBM's Black Team, a group originally assembled (back in the day) as simply a group of developers who were a little better at testing than other developers, and so good candidates for a dedicated testing team. The hope was that they would improve testing some. The outcome was that they started wearing black, twirling (sometimes real, sometimes not) mustaches, laughing maniacally, and putting other people's software through tortures that would make the Spanish Inquisition blush. In other words, they became my kinda people. I had never heard of the Black Team, and now I have, so my life is a little better than it was before. You might find something similarly valuable in this book, even if you don't need to manage software development tasks.