Assignment 09

Alex Clemmer
Student number: u0458675

Randy Sylvester—L-3

Whenever I hear a talk by someone who has job that is either in the government, or closely related to government work, I have the same question: why? Arguably the best thing about CS is that failure is cheap—if I was a bioengineer, I'd need a lab and 20 years of experience to do what I want. In CS I just need my toolchain. I think this is why there are a lot of awesome tech companies to work for. In contrast, the government is slow and horribly bureaucratic, and companies that must deal with this tend to adopt similar qualities. I don't understand why anyone would elect that option.

Mr Sylvester spent a lot of time talking about the technology that they deal with on a regular basis, which tends to be interesting and even impressive. L-3 offers the ability to work with high-reliability systems, and machines that in general are not really available elsewhere. They design and build machines like the Predator drone. Their systems must be robust and completely reliable.

L-3 is also a large company. They employ thousands of people and have them doing all manner of different things. If you're in CS at all, there's probably someone at L-3 working on something you're at least remotely interested in.

The downside is that you do not seem to have the freedom to do many things that you just want to do. L-3 is a company whose money primarily comes from government contract. What they do is thus explicitly limited to thing in that area. I do not get the impression that managers expect (or even want) you to pursue projects on your own, just because you can.

One problem for L-3 that strikes me as I heard Mr Sylvester talk was that most of the really excellent people I know actually do want these freedoms. They want the opportunity to do these things. They're good because they can do what they want. L-3 hires a lot of people in Utah, but I do wonder how they acquire and the retain the really talented people. It doesn't seem like they have this ability, and I think that quality is manifest at least in some part by the sorts of managers who make it to the top. The lead technologist for the local L-3 is different from the lead technologist at the local Google building. I'd of course be interested in hearing about how they combat this problem, although I suppose it is possible that they do not have to.

Culture and management issues aside, the tech issues really are sort of impressive. Autonomous drones, for example, rely on a huge and disparate set of technology, not just to manage critical things like air speed, elevation, and direction, but also to manage things like location data, and to do things like find the stuff you wanted to take pictures of.

The interesting thing here is not getting these things to work, but getting these things to work *reliably*. If you have a bug in the bit of software that deals with airspeed (or, say, autodestruct), you can immediately cost your client millions of dollars. This sort of pipeline requires very strict testing, and probably much stricter code conventions.

Here's the weird thing: I know a lot of people who work at L-3 because they have an excellent co-op program. Most of their technology is really old. Their version control system, for example, is a version of IBM Rational ClearCase from more than 10 year ago. Their IDEs, debuggers, and so on are similarly old.

I suppose the argument to be made there is that these technologies are stable. I think that's probably a bit of a simplification: what's stable is the *company*. To transition a company to other technology, and then to ensure the correctness of their solution, is very expensive. For example, Rational ClearCase from 10 years ago is probably a worse VCS solution than other (free) systems you can pick for almost any task you have at hand. It's not that RCC is stable, it's that L-3 can *stably use it*. They don't have to re-train people, and they don't have to transition.

Somewhere there must be a balance between this. It's silly technology from 10 years ago for something as crucial as version control, and it is equally silly to be switching all the time. On the other hand, it's almost as silly to switch all the time, without good reasons. But it is important: I read a study at one point that showed that companies with a significant ability to either adopt new technology or develop new tools in-house stood a *much* better chance of doing well in general.

I actually think the solution to the transition cost problem is probably to use smaller teams, where the cost of making these changes is reasonable. I don't think it's an accident that Google and Facebook are set up like a bunch of small startups. L-3, on the other hand, seems to believe that its products must be developed with large teams. I'm not sure that's true, but it would certainly be a challenge to do. One reason may be that it's hard to be accountable for everything that happens in a company without building this prohibitive

development model. It reminds me of the static typing problem: you get certain guarantees about the types of errors you'll encounter when you develop something in, say, Java, but at the cost of development flexibility.