Developing Software Engineers at the C-130J Software Factory

Richard Conn, *Lockheed Martin and Kennesaw and Southern Polytechnic State Universities*

Lockheed Martin's C-130J Avionics/Software Integrated Product Team (hereafter referred to as the IPT) creates software that runs a wide variety of systems on the C-130J aircraft. This team develops embedded safety-critical real-time air vehicle software and a ground-based data analysis system for aircraft analysis. The IPT operates within the infrastructure of the C-130J Software Factory,[1] which consists of Sun workstations and PCs networked to Web servers, a configuration management server, an aircraft simulator implemented in software, and laboratories comprised of the aircraft's hardware mounted in equipment racks for easy access. A Web-based digital nervous system[2] supports software engineering activities including data collection and metrics generation for software product and process evaluation.

This IPT has educational needs as diverse as the roles of the C-130J aircraft. IPT activities cover many software development domains that address corporate, Federal Aviation Administration, and national and international military and civilian requirements. Many new hires, however, lack preparation for this environment. This article discusses the IPT's diverse education and training needs, focusing on how to address shortfalls in conventional computer science and engineering education that result in mismatched expectations between the new hire and the company.

The experience of new hires at Lockheed Martin's C-130J Software Factory serves as a focal point in discussing how industry and academia must coordinate efforts to produce effective software engineers.

The C-130J Airlifter and its software factory

To appreciate this article's perspective, it's essential to understand the product we create. Lockheed Martin rolled out the first production C-130 aircraft on 10 March 1955. Since then, Lockheed Martin has built more than 2,100 C-130s, and over 60 nations worldwide fly them in dozens of variations. C-130 aircraft

- Carry troops, vehicles, and armaments into battle
- Drop paratroopers and supplies from the sky
- Serve as airborne and ground refuelers
- Serve as flying hospitals and hurricane hunters
- Provide emergency evacuation and humanitarian relief
- Perform airborne early warning and maritime surveillance
- Operate in extreme conditions, from the Sahara deserts to the ice of Antarctica
- Have helped recover space capsules

Terminology

As used here, *new hire* refers to people who recently joined the IPT fresh out of school. It does not refer to those who join the IPT with previous software engineering experience. *Avoinics/Software Integrated Product Team (IPT)* refers to the team of people who develop the avionics and software for the aircraft and integrate the various aircraft systems into a cohesive whole. This team works in the *Software Factory*, an environment of networked computers, laboratories, and software and hardware simulators.

Mature software engineering organization refers to maturity as defined by the Software Engineering Institute's Software Capability Maturity Model. CMM Levels 4 and 5 designate mature software engineering organizations. *Data primitives* refer to raw data such as defect counts and cost measurements. *Measurements* refer to information items derived from the data primitives, such as the number of defects per lines of code.

Problem space refers to a body of knowledge that an IPT or individual must possess to accomplish a mission. It can cover many knowledge domains.

In May 1992, Lockheed Martin delivered the 2,000th C-130, a C-130H. In September 1992, formal development of the C-130J began. Unlike its predecessors, the C-130J is a software-intensive system, employing modern avionics that have significantly improved its performance. By March 2001, the C-130J was flying with a full complement of embedded safety-critical, real-time air vehicle software and had set 50 world records in the National Aeronautics Association's C-1-N (aircraft weighing between 132,276 and 176,368 pounds) and short take-off and landing (STOL) categories for speed, altitude, time-to-climb, and other aspects of the aircraft operation (see www.lmasc.com and www.lockheedmartin.com.)

The C-130J Airlifter incorporates distinct products, such as display panels, radars, and engines, from over 20 suppliers. Two mission computers backed up by two bus interface units integrate these products into a cohesive whole and control all aircraft functions. During operation, the C-130J Airlifter is supported by a ground-based data system that offloads data contained in the digital flight data recorders for analysis, including future failure prediction.

The IPT develops the mission computer and bus interface unit software and the ground-based data system software, and integrates the supplier's products. The IPT is supported by a Web-based digital nervous system that tracks problem reports, documents and tracks software changes, conducts software product evaluations, reports costs, and records and analyzes various process and product data.

Problem spaces

All new IPT hires have at least a Bachelor's degree in computer science or one of the engineering disciplines, such as electrical or mechanical engineering. They have written relatively small programs during their educational career, and, in virtually all cases, these programs had a well-defined problem space or domain.

One of the first adjustments new hires face is realizing that they do not understand the entire problem space. Among some of the surprises:

- No one person completely understands all of the aircraft's software. Teams in over 20 companies write the software, which the IPT then integrates into a cohesive whole. The various teams specialize in domains such as propulsion, radar, communications, air traffic control, and collision avoidance. Depending on the customer, even more specialized domains can exist, such as hurricane monitoring and atmospheric test equipment.
- Even within the domain of integrating the various aircraft systems, the software engineer does not know enough to just sit down and write the code. Software engineers work with systems engineers, customer representatives, flight test specialists, and others to draw up the requirements for the code they will later have to write. Software quality assurance specialists, auditors, and other team members check their work. To most new hires' surprise, requirements specification is the single greatest source of injected defects in our code.
- The code is large, well beyond most new hires' previous experience. The IPT constructs the C-130J airborne software baseline from which the IPT then derives customer-specific variants, and this adds up to over five million lines of code. The new hire's experience with relatively small programs that cap off at 10,000 lines of code in a well-understood domain offers scant preparation for the more constrained discipline needed to build a larger, safety-critical software system like that on the aircraft.

College graduates as raw material

This new environment comes as quite a cultural shock to most new hires, who, proud of their accomplishments at school, find that their education to date is really just a springboard for another step. Many companies view people straight out of school as raw material to be shaped to the corporate culture. In such a setting, new hires face some common frustrations.

Process. A mature software development organization follows a defined process that, while mutable, does not change easily or quickly. Many ideas the new hire carries from academia might not fit into the process and thus simply fall by the wayside. New hires may feel like they are taking a step backward, and in some cases they are—but for a good reason. The mature software development organization's objective is to get a defined product out the door on schedule and within budget with no major defects and few minor defects. Unwisely introducing new technology and ideas can disrupt these deliveries. CMM Level 5 recognizes as key the transfer of new technology into the organization, but only when disruption will be minimal.

The place of coding. Coding is often a very small part of a software engineer's life, while in academia coding was probably the end-all and be-all of the new hire's view. In a mature software development organization, much of a software engineer's life is spent in meetings, discussing requirements, planning, evaluating software products (requirements specifications, designs, code, test scripts, deliverable documents, and more), documenting and reporting, and testing. The IPT's software development process incorporates over 110 distinct subprocesses, only a handful of which involve writing code.

Early team roles. Many new hires expect to jump into the "fun" coding work right away, but this is seldom the case. Before they move into the development groups that actually write the code, new hires are usually assigned to test code and participate in software product evaluations on code, requirements specifications, designs, test scripts, and other deliverables and nondeliverables. This is often more challenging and frustrating than coding.

These early roles, however, serve an important purpose: catching defects before product delivery. These roles also serve an important secondary purpose in that new hires learn about the corporate culture and the people they will work with later as requirements engineers, designers, coders, and testers. Role-specific training for all IPT members is controlled by the Software Factory's Web-based Learning Navigator tool and augmented by mentoring.

Metrics. Providing various data primitives and metrics to management, rather than just concentrating on the work at hand, form a significant part of the corporate culture. Few new hires have experience in using metrics for project management or see the value in doing so.

More mature software engineering organizations, by contrast, use data primitives gathered throughout their processes' execution. They also use metrics computed from those data primitives to measure the quality of the products produced and processes executed by their teams. They establish "metrics yardsticks" by which to measure process and product quality and then use these to establish new goals. Metrics help senior management gain visibility into programs and projects for which they are responsible without having to take part in the day-to-day activities. Metrics help first- and second-line management gain insight into where defects are being injected into their processes and products, making it easier to improve the quality. Metrics are a way of life in mature software engineering organizations.

Soft skills: A missing ingredient

A common and significant omission in most new hires' education is the development of "soft skills"—communication, sales and marketing, and teamwork in particular. One of the greatest shocks to a new hire is the amount of communication a software engineer must do. Some of the communication that software engineers perform routinely includes documenting designs, preparing and giving presentations, discussing and evaluating problems and challenges, understanding and evaluating software products, tracking action items, and documenting changes to software products. They also must write requirements, test plans and cases, defect reports on software products, and activity reports. Few new hires did much or any of this in school.

Many new hires are surprised by the need for sales and marketing skills. Teams seldom accept new ideas readily from any team member, particularly a new hire. Team members must communicate their ideas effectively to sell both software engineers and management (which might have little or no background in software engineering) on them, and they must package and market them with increasing skill as they go further up the engineering and management chain. To new hires, the frequent rebuff of their ideas can be a constant source of frustration. Even experienced software engineers find this frustrating, but the better ones

Coding is often a very small part of a software engineer's life, while in academia coding was probably the end-all and be-all of the new hire's view.

have the advantage of a track record that gives them an edge new hires do not enjoy. New hires' education-oriented track records seldom count in these situations.

Adjusting to working on a mature software development team poses several additional problems. Most new hires' rewards throughout their educational career have come from their individual efforts, and they typically have little exposure to the team-oriented processes found in mature software development organizations. They might be used to meeting deadlines through individual heroic effort, often at the last minute, which contradicts a controlled, process-driven software development approach where deadlines are met through consistent team effort over time—sometimes months or years. Heroic efforts might prove necessary from time to time, but as both the software and systems engineering activities mature, such efforts become less frequent.

New hires also might find software product evaluations discouraging. When a team critically examines the new hire's software (such as a test plan or a requirements specification), the new hire might view such critiques as attacks on his or her work. Educational experience rarely prepares the new hire for a world in which software cannot be produced without teamwork and evaluation. This cultural mismatch creates vastly different expectations for new hires and the mature software development organization employing them.

Unless our educational system can produce graduates familiar with and ready to adopt more mature software development practices, software failures will persist.

Retention and investments lost

This mismatch of expectations can lead to problems in retaining new hires. The first job might become just a training ground for the next as the new hire becomes increasingly frustrated with the discipline of mature software engineering. When a company loses a new hire, it loses its investment in that person. Organizations must plan for such a loss, being ready to fill the gap until they can find a replacement. The safeguards in place within mature software engineering organizations ensure that the work the new hire has done is not lost, but nothing can replace the initial investment.

IPT Solutions

Overcoming one culture and instilling a different one in a group of people can be difficult and time-consuming. As happens at most companies, new IPT hires attend a new employee orientation, a sort of preliminary indoctrination to the culture. This indoctrination covers a lot of material, and time constraints prevent in-depth explanations of why things are the way they are.

New hires then go to their specific assignments, where they receive job-specific material to study and are assigned mentors to guide them in their daily work. They also receive role-based training, which offers the opportunity to explore the rationale behind company processes and procedures. However, most new hires don't ask such questions, simply accepting the training at face value. We therefore try to work into the training information on the rationale.

What universities can do

Being educational rather than training institutions, universities and colleges produce graduates whose educational foundation lets them technically adapt to challenges both foreseen and unforeseen. Corporations like Lockheed Martin then take the new hires as raw material, training them to deal with organizational issues such as

- Software process and its execution—how to produce requirements specifications, create designs, develop code, evaluate software products, test software products against the requirements, and so on
- How to use the organization's software tools and programming languages within the bounds of the processes and methodologies adopted
- Organization member information, such as chain of command, problem reporting, and daily operation details

The IPT interacts with local universities, particularly Southern Polytechnic State University and Kennesaw State University, and such institutions as the Software Productivity Consortium (SPC; www.software.org) and the Software Engineering Institute (SEI; www.sei.cmu.edu), to help address its employees' educational needs. The resulting cultural change we see in new hires from these universities—from a focus on coding to involvement in other software engineering activities such as requirements definition—eases their entry into the corporate culture. We would see more of this across the industry if universities would interact more with local firms, collaborate with organizations like the SPC and SEI, and become involved with groups such as the Forum for Advancing Software Engineering Education (www.cs.ttu.edu/fase) to help faculty better understand the issues facing industry.

At the IPT, organization members provide some training; other training comes from Southern Polytechnic State University's Software Engineering Retraining Program (http://cs.spsu.edu) and the Software Productivity Consortium's courseware collection, among others. Such training, however, is not appropriate as part of a regular degree program at a local university or college—an institution would compromise graduates' educational foundation if it permitted such corporate influence in its degree programs. More fitting are courses like CSIS 1020 at Kennesaw State University (http://science.kennesaw.edu/csis/), which introduces Visual Basic programming with a software engineering flavor to first-year students, and degree programs like Southern Polytechnic State University's Masters in Software Engineering (http://cs.spsu.edu), which covers general concepts such as capability maturity and software engineering principles.

Rather than making training specific to the current needs of local industry part of their degree programs, universities and colleges should foster fundamental cultural change by adding software engineering and soft-skill knowledge units. Such knowledge units might include

- Dealing with problems larger than a single student can grasp and that require a team of domain specialists to solve
- Working with a team of people under the control of a process
- Developing products that a team of peers will critically evaluate
- Testing and evaluating products effectively, to detect as many defects in the products as possible, and learning how to deal with the critical evaluation of one's own products
- Communicating concepts effectively and selling and marketing ideas to others
- Measuring product and process quality analytically, establishing criteria to assess product or process improvements

Such cultural changes can better position our graduates to produce the higher-quality products we need now and in the future. Work was already underway to place software engineering knowledge units into Computing Curriculum 2001 of the Association for Computing Machinery's Special Interest Group in Computer Science Education (www.acm.org/sigcse/cc2001/), and software engineering degree accreditation under the auspices of the Accreditation Board for Engineering and Technology (www.abet.org) will begin in Fall 2002, but it must go further than just covering software engineering topics to effect the desired cultural change.

As society relies increasingly on software for critical operations, software failures become intolerable. In aircraft, they can cost lives; in our communications systems, they can cost billions of dollars. In a microwave oven, a software failure can result in a very costly recall of thousands of units. Failure of a commercial software product can take its manufacturer out of business, and if a Web site used to transact sales fails, a company could lose millions of dollars an hour until the Web site is repaired.

Unless our educational system can produce graduates familiar with and ready to adopt more mature software development practices, however, software failures will persist. The education community must evolve at the cultural level to produce graduates who already embrace mature software engineering practices. While students acquire knowledge relatively easily, the wisdom to apply that knowledge well comes much more slowly. The cultural changes proposed here, therefore, go beyond development of technical knowledge to culturing the experience and mindset needed to create and apply high-quality software.

References

1. R. Conn, S. Traub, and S. Chung, "Avionics Modernization and the C-130J Software Factory," *Crosstalk—The Journal of Defense Software Engineering*, vol. 14, no. 9, Sept. 2001, pp. 19–23
2. B. Gates, *Business @ the Speed of Thought*, Warner Books, New York, 1999; www.speed-of-thought.com.

About the Author

Richard L. Conn is a Software Process Engineer for the C-130J Airlifter at Lockheed Martin Aeronautics Company. His research interests include software process engineering, component-based software engineering, software reuse, and software engineering education. He holds a BS from Rose-Hulman Institute of Technology and an MS from the University of Illinois, both in computer science. He has served on the Federal Advisory Board for Ada (receiving ACM/SIGAda's award for Outstanding Contributions to the Ada Community), the DoD Software Reuse Initiative, ACM and IEEE software engineering education workshops, and IEEE standards efforts. Contact him at Lockheed Martin Aeronautical Systems, 86 South Cobb Dr., Dept. 70-D6, Mail Zone 0674, Marietta, GA 30063-0674; richard.l.conn@lmco.com, http://unicoi.kennesaw.edu/~rconn, http://cs.spsu.edu/rconn.