A schooling on their implications for software engineering [trends]

Ann K. Miller

University of Missouri--Rolla

Follow this and additional works at: http://scholarsmine.mst.edu/faculty_work

Part of the Electrical and Computer Engineering Commons

Recommended Citation

http://scholarsmine.mst.edu/faculty_work/1371

This Article is brought to you for free and open access by Scholars' Mine. It has been accepted for inclusion in Faculty Research & Creative Works by an authorized administrator of Scholars' Mine. For more information, please contact weaverjr@mst.edu.
Some say the most accurate weather forecast, when averaged over an entire year, is to predict that tomorrow’s weather will be just like today’s. So when writing on trends in software engineering, it is tempting simply to extrapolate some of the popular, current trends. And, in fact, we will examine a few of these obvious indicators. But, weather also can exhibit a radical change from one day to the next. So, too, software engineering has had new approaches that are more revolutionary than evolutionary.

In particular, we will try to discern trends in software engineering based on trends in delivered software systems. Read them, think about them, disagree with them, create your own opinions, but, consider the various possibilities—because it is your future, too.

What follows are the opinions of someone who is more a practitioner than a theoretician and admittedly not much of a forecaster. If you had asked me when I first got out of graduate school and started teaching at a university if I would ever work in industry, I would have said “no.” If you had asked me on my 10th anniversary of working in industry if I would leave to work in senior government service, I would have replied “definitely not.” And yet I did just that. Now I have returned full circle to academic life. These varied opportunities provide the insights on which these forecasts are made.

System trend 1: more, bigger, better

This trend, obviously, is based on an extrapolation of many years of historical data. Application programs as well as operating systems have been and likely will continue to grow. This phenomenon has been aided by the incredible growth in density and speed of integrated circuits that Gordon Moore accurately forecast in 1965. Thus, software products, whether measured by thousands of lines of code (KLOC) or by bytes of program memory, have exhibited a software version of Moore’s Law. But what does this continued growth in sheer size imply for changes in software engineering? Several things.

Basically, there will be new and better methodologies for developing large-scale systems. Why is that? This is a “tomorrow’s weather will be like today’s weather” prediction. If we look at the history of large systems, they grew initially in an ad hoc, rather chaotic, fashion. Since some of the largest early programs were for military applications, it became imperative that there be some rigor in the development process. Winston Royce was the first to use the term “waterfall model” in 1970. His paper became the basis for the U.S. Department of Defense (DoD) standard 2167A for the development of software systems. The classical waterfall model of systems analysis, software requirements specification, analysis, preliminary design, detailed code and unit test, integration and test, and software quality test with on-going maintenance (Fig. 1) was an important first step in establishing rigor and repeatability in large-scale system development. However, the waterfall model quickly became impractical to use; this was due to various reasons, most notably changing requirements, new requirements and increased customer expectations.

As a result, methodologies such as the incremental model (Fig. 2), the evolutionary model (Fig. 3) and the backbone model (Fig. 4) have been adopted. These are all variations of the waterfall model. But they improve the way changes and growth of requirements can be managed.

So what new improvements will occur? Improvements will occur in automated tracing of requirements and in life-cycle support of changes, including better configuration management tools for multiple releases of a product in various stages of development. This may not sound like a big deal if you have only written short software programs required for courses and labs. However, Lehman and Belady studied the history of successive releases of a large operating system and found that the total number of modules increased linearly with the release number. More significantly, the total number of modules affected by changes increased exponentially with the release number.

One reason systems are getting larger is that they include whole programs inside the newer systems. In many cases, the smaller components are com-
mmercial-off-the-shelf (COTS) products such as a database system or a computa-
tional package. The advantage to this approach is that the overall larger sys-
tem can be developed more quickly. This, in turn, lets the company bring the 
product to market sooner.

A disadvantage to this approach is that rarely is the COTS component a per-
frect match for what is needed. That is, it may not perform all the needed func-
tions. Just as seriously, it may perform more functions than what is needed. This
is a potential problem. The incorporated component may cause the entire system
to behave unexpectedly due to an “extra” function being invoked. Thus, we will
see better black-box testing techniques to verify that the functions that should be
performed are performed correctly and, equally important, extraneous functions
cannot be performed.

A second type of incorporated software is a “legacy system.” This is a pro-
gram that has been fielded for years but may have been written in a different
language than a newer system; or it may have been developed in a non-object-
oriented approach but will be part of an upgraded object-oriented system. We
will have improved wrapper techniques for encapsulating such legacy code into
updated systems.

Not only will the program size be large, but the amount of data manipulat-
ed by the programs will continue to grow. Thus, there need to be improved tech-
tiques for managing large datasets, including visualization techniques. Common mechanisms are the use of
graphics, color and animation. Much work has been done using three-dimen-
sional graphics and animation. However, there remains a dire need for
better visualization techniques for massive datasets. And while visual tech-
niques will predominate, there may be excellent applications where other sen-
ses such as sound or touch can provide additional assistance to the user in either
a virtual reality setting or in a multimedia environment.

Lastly, a forecast that is a bit more of a stretch. We will see the emergence of
true visual programming languages. Not just a graphical user interface (GUI),
but a high level language that lets the developer point and click to various
components (not just small routines) from multiple libraries/multiple sources
and to specify the type of connection between these components. The final
integration of these components will actually be completed by the language’s
compiler/assembler.

System trend 2: faster, faster

As computers become embedded in more and more systems, the need for
real-time and hard real-time processing increases. By real-time, we mean that
the system must acquire data and/or process it within specified time inter-
vals. Real-time systems range from programable logic controllers that manage valves in a manufacturing plant to
microprocessors that manage fuel-injec-
tion engines to banks of microproces-
sors that provide avionics flight control
for aircraft.

These systems are difficult to con-
struct because of timing. We’ve all seen
our word processor or our web browser exhibit markedly different times to exe-
cute a function depending on size of a
file or number of users on the network or
a myriad of other factors. In a real-
time system, certain functions must fin-
ish within a specified time limit.
Otherwise, data will be lost or a switch
will not close in time or some critical
operation will not be performed.

To better design these systems, we
will see advanced simulators emerge
that can better describe and characteri-
ze the run-time operation of the system
under development. These will include
microprocessor models linked with sim-
ulators that also provide an extensive
suite of test tools for timing and for
fault detection. Fault detection tools will
be included. For while speed is essential
for embedded systems, there is another
factor for real-time systems that is increasingly important: safety.

System trend 3: safer

Embedded computers control more
and more systems, whether launching a
satellite, managing oil flow in pipelines
or controlling flight of aircraft. Thus,
there is a growing need to assure the
proper functioning of these systems.
This is due to the significant cost of the
system and/or the cost of the resource being man-
aged and/or to the fact that human life depends
on that system.

For such life-critical and safety-critical sys-
tems, there will be an increased emphasis on
validation and verifica-
tion (V&V). Validation
ensures that the system features can be traced back to the stated requirements.
(So there should not be flight simulators
embedded in future spreadsheet pro-
gams.) Verification ensures that each
function works correctly.

The classic phrases for these two
activities are: (i) are we building the
correct product and (ii) are we building
the product correctly? There are stan-
dards by various organizations such as
IEEE and the International Standards
Organization (ISO) that are necessary
but no longer sufficient. A trend for the
future is not only V&V but a further
step to certification.

For example, the Radio Technical
Commission for Aeronautics standard
DO-178B, entitled “Software
Considerations in Airborne Systems and
Equipment Certification,” is a standard
developed by the commercial air trans-
port industry for software used in com-
mercial aircraft. Essentially, all devel-
opers of aircraft systems adhere to this
standard. The standard defines various
criticality levels for software and pre-
scribes different techniques for each
level. I envision this type of standard
spreading to other application areas,
first perhaps to medical applications but
eventually to a much wider arena. With
these standards, there will need to be a
precise and testable means to determine
if the standard has been fulfilled or not.

As more and more software systems
are safety-critical and life-critical appli-
cations, there will need to be a mecha-
nism to guarantee system performance.

The National Research Council spon-
sored a committee of information technology experts who published their findings in the 1998 report: “Trust in Cyberspace.” Among their comments:

“The absence of standard metrics and a recognized organization to conduct assessments of trustworthiness is an important contributing factor to the problem of imperfect information.

“A consumer may not be able to assess accurately whether a particular drug is safe but can be reasonably confident that drugs obtained from approved sources have the endorsement of the US Food and Drug Administration (FDA) which confers important safety information. Computer system trustworthiness has nothing comparable to the FDA.

The problem is both the absence of standard metrics and a generally accepted organization that could conduct such assessments. There is no Consumer Reports for [software and information] Trustworthiness.”

The absence of a certifying organization such as Software Consumer Reports or an Underwriter’s Laboratory is a key problem. But, so, too, is the lack of sound metrics for quantifying the trustworthiness of information systems.

The frequency and sophistication of intrusions and attacks on commercial systems as well as government and military systems have led many agencies to be concerned about trustworthiness. This includes more than the traditional notion of security. Information and system survivability in the face of intrusions and attacks is recognized as a difficult but critical goal. So, while there may eventually be a national software certification office, a first step on the way to certification will be more emphasis on trustworthiness and survivability.

System trend 4: survivable

We have all seen the increase in intrusions to networks and viruses spread via e-mail. Not only will computer networks need better intrusion detection mechanisms, so too will application software. Web-based software will continue to increase. It will provide enormous advantages from increased distance learning capabilities to more secure electronic commerce applications.

However, there also will be increased concern about the fail-safe modes of operation in systems. Buffer overruns and other typical flaws will need to be examined before systems are fielded. Thus, we will see improved testing tools utilizing fault injection and other techniques. We will see more use of cryptography and better authentication techniques, including digital signatures. We will also see a greater investment by private industry in trustworthy computing.

System trend 5: smarter, synchronizing, collaborating

By this trend, I am not referring to the traditional notions of artificial intelligence but to a collection of interacting software modules. A lot of progress has been made in robotics; this trend will continue and we will begin to see “teams” of robots working together to perform a task. To do this effectively, they will need to collaborate and share information just as a human team would. Thus, they need to be able to update each other.

However, when problems occur, they will need to synchronize and return to some agreed-upon state from which to continue forward together. This provides all sorts of new areas of research in computational intelligence and robotics.

Discussing robots is a natural lead-in to the last trend—and that is people. Because people will continue to be central in software specification, development, and test. And so the last prediction will be:

People trend 1: certification re-visited. Certification at the software or system level is only a first step. We will see certification of software engineering programs and eventually certification of software engineers. A recent issue of IEEE Software discussed these topics in great detail. These included accreditation for university software engineering programs to the certification of individual software engineers similar to the Professional Engineer certification that the State of Texas has begun.