THE VIEW FROM THE CUTTING EDGE

or

Things I Learned in the Great Software Wars

by Larry Bernstein

INTRODUCTION

I developed these observations developing software during 35 years at Bell Laboratories. Many are not original nor are they unique to software, but they are insightful.

The software development process differs little from that for any technically intensive product. I've adopted ideas from many people and applied them to software. I've tried to give credit where due but I have not always been successful, to those I slighted, I apologize.

I offer these thoughts in the hope that you can apply them to management in general and to software in particular, with the faith good practices are universal and, as Will Rogers said, "Everything is funny as long as it is happening to someone else."

ON PROJECT SUCCESS

Your objective is 'not to get it to work, but to get it to work right.'

Jackson

First, make it work. Then make it work right. Then make it work better.

Vyssotsky

Fail small, succeed big.

Every project builds prototypes; it's just that some don't plan it that way.

"You may have noticed that the less I know about a subject, the more confidence I have, and the more new light I throw on it."

Twain

"I was gratified to be able to answer promptly, which I did. I said, 'I didn't know.'"
Twain

A Road Map to Success

1. Start with a small team with broad objectives and build a prototype.

2. Put the prototype into the field and use it. Estimate the size of the job. Use function points and an estimation tool such as Checkpoint or Cocomo.

3. After an analysis of the prototype, enlarge the organization from a small team to a large one. If necessary, write detailed requirements and control them.

4. Using top-down design, partition the project into modules, define and control interfaces, and appoint module owners. Use modern software interface conventions such as object classes, pipes, tag value data, etc.

5. Reduce complexity in the design with a formal "design minimization" effort. Establish a target of 40% simplification by maximizing reuse, eliminating redundancy and simplifying algorithms.

6. Implement designs, using structured programming techniques, only after they have been inspected. Submit tested software and work practices through an independent manufacturer (or builder) to the quality assurance and integration organization.

7. Test incrementally. Create a simple working system and then add sets of changes to gradually increase capability. Do regression tests on each new increment using test cases developed for the previous increment.

8. Find a friendly operational site where the operators are willing to let developers try out new features before they are formally released.

9. Have a soak site for new product releases.

10. Avoid developing a new application on new hardware and/or new operating system software.

11. Have maintainers share some of the continuing development responsibility.

ON LEADERSHIP

When things are the very worst and nothing is working, the leader shows unwarranted optimism.

To lead is to be in front.

A leader must be seen often by her people. Managers should go to their peoples' work stations periodically.

"Enthusiasm and optimism are contagious; so is idleness."

Yuhas

"Be good, and you'll be lonesome."

Twain

"When in doubt, tell the truth."

Twain

"We have to undo a 100 year old concept and convince our managers that their role is not to control people and stay 'on top' of things, but rather to guide, energize and excite."

Welch, CEO of General Electric

Jack Welch's Six Rules

1. Face reality as it is, not as it was or as you wish it to be.
2. Be candid with everyone.
3. Don't manage, lead.
4. Change before you have to.
5. If you don't have a competitive advantage, don't compete.
6. Control your own destiny, or someone else will.

ON PLANNING

"Management planning is not complicated but it is tedious – that's why the temptation is so strong to avoid it."

Sloma

First define what is to be done and plan the order of doing it; then say who is to do it.

"Get your facts first, and then you can distort them as much as you please."

Twain

Plan by using pert charts or schedule charts but track by schedule lists. For each milestone keep two dates, one the "schedule" which you own, the other, a "current estimate" which the developer owns.

Manage the tradeoff between control and stability on the one hand and rapid response time and flexibility on the other. Your plan must acoomdate both.

Instead of imposing procedures to control projects, replace the managers. Procedural controls are weak substitutes for competent managers.

Japan's 1972 Plan

An approach to setting goals for computer technology was the Japanese "Plan for Information Society – A National Goal toward Year 2000." The "Plan," proposed in 1972, was an ambitious and detailed attempt at long-range planning involving both business and government, with a proposed long-range budget of 65 billion yen a year. The authors described it as presenting "a picture of Japan's planned information society which is scheduled to be established by 1985, and the means of attaining this national goal and its schedule."

The "Plan" was based on the assumption that industrial society will be succeeded by an information society in which the goal of mass consumption of consumer goods will be replaced by "a general flourishing state of human intellectual creativity." Some of the projects proposed include remote medical systems, computer-oriented education, a data bank for government agencies, a pollution prevention system, community-based information systems, a think tank center, and a computer peace corps.

By 1996 they found their technology out-of-date and were lagging the industry The Japanese see clearly that the key to designing the future is software. In addition, they consider a period of "social information building" as a necessary prerequisite. Besides realignment of institutions and education of the people, and so on, this seems to be envisioned as a time for verification of information concepts. The authors of the "Plan" believed that it is to Japan's advantage that "the nation and it culture are homogeneous." While homogeneity provides secure institutions, good communication, and common overall goals, innovation requires a milieu in which variations may thrive.

Plans that change are useful; those that don't should be abandoned. Planning for the future is far from foolproof. The best of plans need built-in flexibility, and be prepared changed quickly as the need arises. Yet, having a plan provides leverage on the future.

ON HUMANIZING COMPUTERS

Keep human interactions simple and natural to minimize training and documentation.

Don't try to correct poor software design with good documentation.

Gilb

Keep human interactions modular to permit the customer to organize jobs the way he sees fit.

When the answer is always the same, don't keep asking the question.

"The meanings of various lengths of elapsed time do not vary widely from one person to another:

Less than 1/3 sec is 'instantaneous',
Less than l sec is 'fast'
Less than 5 sec is a 'pause' and
Greater than 10 sec is a 'wait'.

Time sharing interactions should be 'instantaneous.' Transaction interactions should be without 'wait.'"

- *Jones in For Principles of Main Computer Dialog Computer Aided Design, Vol. 10, No. 3, pg 197, May 1978*

Here is a table from my new book, " Basic Understanding of telecommuniucations Networks: Cooper to Sand to Glass to Air." ISBN 0-306-46237-0, Kluwer Academic Publishing

System/Software Usability Principles

Principle	Explanation
Speak the users' language.	Use words, data labels, and concepts familiar to the user. Present information in a natural and logical order in the user's context.
Be consistent.	Indicate similar concepts through identical terminology and graphics. Adhere to uniform conventions for layout, formatting, typefaces, and labels.
Minimize the users' memory load.	Rely on recognition, not recall. Do not force user to remember information across documents. Rely on human factor specialists to determine appropriate levels of memory demand based on context and user skill.
Design for flexibility and	Accommodate a range of user sophistication and diverse user

efficiency.	goals.
Design aesthetic and minimalist graphics.	Create visually pleasing, efficient displays that capitalize on known human capabilities in pattern recognition, color differentiation, etc.
Recognize the power of *chunking*.	The capacity of human short term memory is small, but can be amplified by grouping subsets of information around keywords, by completing single thoughts in one document, and by keeping tasks short but information-rich.
Structure progressive levels of detail	Organize information hierarchically, with more general information appearing before more specific detail. Allow the user to stop when sufficient information is received.
Facilitate navigation through program structure.	Allow user to determine current position in the program structure. Make it easy to jump among related tasks. Make it easy to return to an initial state.

ON TESTING

"Systems not tested do not work."

Casulli

Test incrementally.
Test under no-load.
Test under medium-load.
Test under heavy-load.
Test under overload.
Test the error recovery code.
Tests that do not cause the system to fail are unsuccessful.

Have separate system integration and test groups to insure that the system works and, even more importantly, that it can be delivered.

Three times the effort is needed to fix a bug found in the test lab than if it were found by the original programmer. Ten times the effort is needed to fix a bug found in the field than in the test lab.

Testing without data analysis is debugging; debugging is trouble shooting; trouble shooting is time consuming. Data analysis requires an investment in data reduction and in the software having self-diagnostic capability.

Expect that 50% of time and 20% of cost will be spent testing, unless you use regression testing. Regression testing cuts testing costs in half.

Nowak, private conversation.

A 2% reduction in defects is usually accompanied by a 10% increase in productivity.

Lynas, Harvard Business Review, August, 1981

Stress testing is not complete until the points where the system breaks are found. Design margins are the difference between the breaking point and the operating point.

When you do volume testing, make sure that someone checks the appearance and validity of the output - people don't always look at the output from stress tests.

Models are useful for test case development.

Spend more time testing stability and performance than features. Have a "friendly" operational site at which developers might do some feature testing before the software is released. Test the recovery system very well.