THE VIEW FROM THE CUTTING EDGE 2

or

Things I Learned in the Great Software Wars

by Larry Bernstein

ON CUSTOMERS

You must be a world-class customer before you can be a world-class supplier.

Before I can trust you as a partner I must have confidence in you as a supplier.

"The best – and cheapest – way to keep customers satisfied is to serve them well from the start."

People want systems to increase their status, not eliminate or simplify their jobs.
Fortune Magazine

Don't automate an undisciplined workflow. The computer won't solve what the customer's management can't.

Before the first release, customers want lower prices, more features and shorter schedules. When the system becomes operational, they want better reliability, throughput and response time.

In dealing with customers, you must understand that there are things they want and things they need. A successful project manager distills the needs from the wants and satisfies them while delivering the system at reasonable cost.

Train a "super user" on the customer's staff. Find people who know the job and let them participate in system and module design. They will field many problems without distracting the development organization.

An organization with good customer relations is twice as productive as one with poor customer relationships.

Hostages: those sent to the customer's site to get the system working when the data and tools are only available at home. They "show the flag," look intelligent and spend long hours talking to the customers.

System Abuse: when customers use the system in ways not intended. System abuse can arise from poor operations, overload or improperly maintained databases.

Keep the "should be's" separate from the "is's."

ON RISK

"Whenever you win something, you also lose something."

Confucius.

The major risk factors in software development, listed in decreasing importance, are:

1. The skill, experience and quality of the staff.
2. The number of operating system changes being made in parallel with the application software.
3. The number of design options.
4. The size of the organization.
5. Real-time performance shortfalls
6. Schedule compression
7. Excessive paperwork
8. Poor Human Interface Designers
9. Gold-plating
10. Creeping featurism
11. Straining Software Technology
12. Inaccurate Sizing
13. Management Malpractice
14. External suppliers do not deliver

If a project is faced with a large number of operating system changes, a large number of design options and has a lot of new people working on it, its chance of success are small even if it is well managed. For these projects, success depends on the ease of integrating its piece parts and the ease of accommodating the changing user environment.

Examine project dependencies to find risks.

Always have a contingency plan for an identified risk.

.

ON REQUIREMENTS

"The requirements and design phases are the most important steps in a (software) project. If these steps are not done right, the product quality is almost guaranteed to be low. It is good to perform these steps with paper designs to maintain a fluid, dynamic design methodology. Any computer investment activity (except prototyping) during the interval, including early coding, imposes a psychological restraint to change. Unrestrained change is necessary for a quality design."

Hallin and Hansen

> Software built without controlled requirements might work, but will not do what is specified.
>
> Software built to controlled invalidated requirements will probably be late, costly, and too complicated and not do what is needed.
>
> Software built to controlled and validated requirements may work but only do a subset of what is needed and required.

"There are four kinds of problems that arise when one fails to do adequate requirements analysis:

1. Top-down design is impossible
2. Testing is impossible
3. The user is frozen out
4. Management is not in control

Although these problems are lumped under various headings to simplify discussion, they are actually all variations of one theme–poor management. Good project management of software procurements is impossible without some form of explicit (validated) and governing requirements.

Royce Practical Strategies for Developing Large Software Systems, pg. 59

Software functions and performance need to be modeled.

"Invalidated functional requirements with no regard to performance can lead to unmaintainable products." *Jackson*

"We should be interested in processes, not functions." *Jackson*

The content of specifications, not their format is important.

"Information and communication problems today require solutions which are integration oriented - first for people, then for technologies."

Steele, Center for Information and Communications Services.

ON ENGINEERING

Systems are networks. They are not hierarchies.

Software engineering is the process of making tradeoffs among costs, schedules, features, and design.

"A good system engineer has smart friends."

Casulli

Software engineers design systems to fit within the architecture of the target machine. Computer scientists hide the target machine from the solution.

When system performance is unacceptable find the bottleneck; fix it, and call the result "fast path", as in fast path I/O, fast path polling, etc.

"Find software modules needing redesign for performance. Find bottlenecks before the customer finds them. Insure that the installation is well engineered, that is, that all resources exhaust at about the same load point."

Bolksy and Boyle, Bell Labs Record, November, 1977.

The system engineer comes up with a solution; the developer comes up with the solution.

Don't collect data until you know how you are going to use it.

Don't produce reports until you know how they will be used.

ON SOFTWARE ARCHITECTURE

Use operating system software and hardware familiar to the developers. If you can't, provide extensive training.

Partition the software into separate modules. Modularize with well-defined interfaces to simplify testing and feature packaging.

Estimate performance, and then measure it. Establish design margins.

Maximize the re-use of common modules across product lines.

Isolate hardware and data structure dependencies from feature software. Pay the performance penalty to reduce development risks.

Minimize cross feature dependencies.

You can change some of the interfaces some of the time, or some of the modules some of the time, but don't even think about changing both the interfaces and the modules at the same time.

"As to the adjective (requirement or feature), when in doubt, strike it out."

Twain

Beware of dangling interfaces.

"...[A] network of interconnected bottom-up design will constitute a de facto architecture for a portion of the overall application which may be highly useful during the period when more lengthy top-down, goal-oriented efforts are puzzled through."

Ross

Checklist for Architecture Review

by Joe Maranzono email: jfm@att.net

1. Start with the Overall architecture picture.
2. Use pictures to describe the components of the subsystem.
3. List major components of the subsystem and the functionality provided by each component.
4. Walk through scenarios of how data/information flows through components.
5. Describe/list data error handling data flow.
6. List all **interfaces** with other systems and/or subsystems.
7. Describe the user interfaces
8. For each interface describe:
   - The IPC mechanism to be used for data and for control
   - Any expected issues with the IPC mechanism (e.g., performance degradation at some capacity level, problems with overload control, new technology)
   - Failure modes and error handling for each type of failure
   - Error recovery to prevent lost data, if needed
   - Effective bandwidth of the interface mechanism
9. Strategies to keep design simple and avoid unnecessary complexity.
10. Robustness of component choices, COTS or custom-made (e.g., available support)
11. Extensibility/Flexibility of components
12. Any local data bases or data stores for temporary data storage or reference data storage

- **Performance and Capacity Requirements**
  The expected traffic in terms of transaction profile and background processing:
  1. Transaction Profile in terms of average load and busy hour
  2. Number of simultaneous users
  3. Expected system response times
  4. Peak arrival rates

5. Over night processing and calculations
6. Database Sizing
7. Network demands including congestion strategy

- **OA&M**
  The Operations, Administration and Maintenance approach:
  1. Operational Environment
  2. Interfaces to external sources and systems
  3. System availability
  4. Failure avoidance and handling
  5. Error handling
  6. Disaster recovery
  7. Security
  8. Data Consistency and Accuracy