Stress Analysis of a Software Project [long]

Jerry Leichter <leichter@lrw.com>
Tue, 26 Apr 94 08:42:32 EDT

The following, which claims to be an internal Silicon Graphics memo, has already seen fairly broad network distribution. I have no way of verifying that it is what it claims to be, but (a) I'm told by someone with close dealings with SGI that it fits with what he's heard; (b) if it's a fake, someone put a huge amount of effort into producing it.

I forward it to RISKS as a wonderful record of what goes wrong with large software projects, and why. It would be as useful if all the names, including the company and product names, were removed. This memo should not be seen as an indictment of SGI, which is hardly unique. There is good evidence that Sun, for example, had very similar problems in producing Solaris; and I watched the same thing happen with the late, unlamented DEC Professional series of PC's, and something like it almost happen with firmware for DEC terminals a number of years back.

I hope that Tom Davis's position hasn't been badly hurt by the broad distribution of his memo - but based on the traditional reaction to bearers of bad news, especially when the bad news becomes widely known, I can't say I'm sanguine about it.
-- Jerry

------- Begin Document -------

Software Usability II
October 5, 1993
Tom Davis

Last May, I published my first report on software usability, which Rocky Rhodes and I presented to at Tom Jermoluk's staff meeting (with Ed, but without Tom). Subsequently, I made it available to quite a few other people.

This sequel is to satisfy all those people who have urged me to bring it up to date. I begin with a summary; details follow.

Please read at least the summary.

SUMMARY

Release 5.1 is a disappointment. Performance for common operations has dropped 40% from 4.0.5, we shipped with 500 priority 1 and 2 bugs, and a base Indy is much more sluggish than a Macintosh. Disk space requirements have increased dramatically.

The primary cause is that we attempted far too much in too little time. Management would not cut features early, so we were forced to make massive cuts in the final weeks of the release.

What shall we do now? Let's not look for scapegoats, but learn from our mistakes and do better next time.

A December release of 5.1.2 is too early to fix much -- we'll spend much more time on the release process than fixing things. Allow enough time for a solid release so we don't get: 5.1.2.1, 5.1.2.2, 5.1.2.3, ...

Let’s decide ahead of time exactly what features are in 5.1.2. If we pick a reasonable set we’ll avoid emergency feature cuts at the end.

Nobody knows what’s wrong -- opinions are as common as senior engineers. The software environment is so convoluted that at times it seems to rival the US economy for complexity and unpredictability. I propose massive code walk-throughs and design reviews to analyze the software. We’ll be forced to look closely at the code, and fresh reviewers can provide fresh insights.

For the long term, let’s change the way we do things so that the contents and scheduling of releases are better planned and executed. Make sure marketing and engineering expectations are in agreement.

INTRODUCTION

We’ve addressed some of the problems presented in the original May report, but not enough. Most of the report’s warnings and predictions have come true in 5.1. If we keep doing the exact same thing, we’ll keep getting the exact same results.

I’m preparing this report in ASCII to make it widely available. It’s easy to distribute via news and mail, and everyone can read it.

An ASCII version of the May 12 report can be found in:

bedlam.asd:/usr/tmp/report.text

The included quotations are not verbatim. Although the wordings are inexact, I believe they capture the spirit of the originals.

BLOAT UPDATE

> "Do you want to be a bloat detective? It’s easy;
> just pick any executable. There! You found some!"
>
> -- Rolf van Widenfelt

In the May report, I listed a bunch of executable sizes, and pointed out that they were unacceptable if we intended to run without serious paging problems on a 16 megabyte system. Between May and the 5.1 release, many have grown even larger. IRIX went up from 4.8 megabytes to 8.1 megabytes, and has a memory leak that causes it to grow. Within a week, my newly-booted 5.1 IRIX was larger than 13.8 megabytes -- a big chunk of a 16 megabyte system. It’s wrong to require our users to reboot every week.

There are too many daemons. In a vanilla 5.1 installation with Toto, there are 37 background processes.

DSOs were supposed to reduce physical memory usage, but have had just the opposite effect, and their indirection has reduced performance.

Programs like Roger Chickering’s "Bloatview" based on Wiltse Carpenter’s work make some problems obvious. The news reader "xrn", starts out small, but leaks memory so badly that within a week or so it grows to 9 or 10 megabytes, along with plenty of other large programs. But what’s really embarrassing is that even the kernel leaks memory that can’t be recovered except by rebooting!

Showcase grew from 3.2 megabytes to 4.0 megabytes, and the master and

status gizmos which are run by default occupy another 1.7 megabytes. Much of this happened simply by recompiling under 5.1 -- not because of additional code.

The window system (Xsgi + 4Dwm) is up from 3.2 MB to 3.6 MB, and the miscellaneous stuff has grown as well. As I type now, I have the default non-toto environment plus a single shell and a single text editor, jot. The total physical memory usage is 21.9 megabytes, and only because I rebooted IRIX yesterday evening to reduce the kernel size. Luckily, I'm on a 32 megabyte system without Toto, or I'd be swamped by paging.

Much of the problem seems to be due to DSOs that load whole libraries instead of individual routines. Many SGI applications link with 20 or so large DSOs, virtually guaranteeing enormous executables.

In spite of the DSOs, large chunks of Motif programs remain unshared, and duplicated in all Motif applications.

PERFORMANCE UPDATE

"Indy: an Indigo without the 'go'".

-- Mark Hughes (?)

"X and Motif are the reasons that UNIX deserves to die."

-- Larry Kaplan

The performance story is just as bad. I was tempted to write simply, "Try to do some real work on a 16 megabyte Indy. Case closed.", but I'll include some details.

In May, I listed some unacceptable Motif performance measurements. Just before 5.1 MR, someone reran my tests and discovered that the performance had gotten even worse. Some effort was expended to tune the software so that instead of being intolerable, it was back to merely unacceptable performance.

We no longer report benchmark results on our standard system. The benchmarks are not done with the DSO libraries; they are all compiled non-DSO so that the performance in 5.1 has not declined too much.

Before I upgraded from 4.0.5 to the MR version of 5.1, I ran some timings of some everyday activities to see what would happen. These timings were all made with the wall clock, so they represent precisely what our users will see. I run a 32 megabyte R4000 Elan.

Test	4.0.5	5.1	% change
C compile of a small application	25 sec	35 sec	40%
C++ compile of a small application	68 sec	105 sec	54%
Showcase startup, May report file	13 sec	18 sec	38%
Start a shell	<2 sec	~3 sec	~50%

Jot 2 MB file	<2 sec	~3 sec	~50%

What's most frightening about the 5.1 performance is that nobody knows exactly where it went. If you start asking around, you get plenty of finger-pointing and theories, but few facts. In the May report, I proposed a "5% theory", which states that each little thing we add (Motif, internationalization, drag-and-drop, DSOs, multiple fonts, and so on) costs roughly 5% of the machine. After 15 or 20 of these, most of the performance is gone.

Bloating by itself causes problems. There's heavy paging, there's so much code and it's so scattered that the cache may as well not be there. The window manager and X and Toto are so tangled that many minor operations like moving the mouse or deleting a file wake up all the processes on the machine, causing additional paging, and perhaps graphics context swaps.

But bloat isn't the whole story. Rocky Rhodes recently ran a small application on an Indy, and noticed that when he held the mouse button down and slid it back and forth across the menu bar, the (small) pop-up menus got as much as 25 seconds behind. He submitted a bug, which was dismissed as paging due to lack of memory. But Rocky was running with 160 megabytes of memory, so there was no paging. The problem turned out to be Motif code modified for the SGI look that is even more sluggish than regular Motif. Perhaps the problem is simply due to the huge number of context swaps necessary for all the daemons we're shipping.

The complexity of our system software has surpassed the ability of average SGI programmers to understand it. And perhaps not just average programmers. Get a room full of 10 of our best software people, and you'll get 10 different opinions of what's causing the lousy performance and bloat. What's wrong is that the software has simply become too complicated for anyone to understand.

WHAT WENT WRONG IN 5.1?

The one sentence answer is: we bit off more than we could chew. As a company, we still don't understand how difficult software is.

We planned to make major changes in everything -- a new operating system, new compilers, a new user environment, new tools, and lots of new features in the multi-media area. Not only that, but the new stuff was promised to do everything the old software had done, and with major enhancements. (Early warning: version 6.0 promises to be even more disruptive.)

About 9 months ago, Rocky and I pointed out the impossibility of what we were attempting. Rather than reduce the scope of the projects, a decision was made to hire a couple of contractors (who know nothing about our system) to handle the worst user interface problems in the Roxy project. In addition, promises were obtained from various executives that a significant effort would be made to improve software performance.

Management was basically afraid to cut any features, so we continued to work on a project that was far too large. The desperate attempt to do everything caused programmers to cut corners, with disastrous effects on the bug count. And the bug count was high simply because 5.1 was so big.

Only when the situation was beyond hope of repair did we start to do something. Features and entire products were removed wholesale from the release, and hundreds of high-priority bugs were classified as exceptions, so that we could ship with "no priority 1 and 2 bugs". We did, however, ship with over 500 "exceptions". The release was deemed too crummy to push to all our machines, but was restricted to the Indys, the high-end machines, and a few others where new hardware required the new software. Due to the massive bug count, virtually no performance tuning was done.

When the schedule is impossible as it was in 5.1, the release process itself can get in the way. The schedule imposes a code freeze long before the software is stable, and fixing things becomes much more difficult. If you know you're going to be late, slip before the code freeze, not after. We're trying to wrap up the box before the stuff inside is finished, and then trying to fix things inside the box without undoing the wrapping -- it has to be less efficient.

Management Issues:

There was never an overall software architect, and there still is not, and until Way Ting was given the job near the end, there was no manager in charge of the 5.1 release, either.

I wrote a note in sgi.bad-attitude about the "optimist effect", which I believe is mostly true. In condensed form:

> Optimists tend to be promoted, so the higher up in the organization you are, the more optimistic you tend to be. If one manager says "I can do that in 4 months", and another only promises it in 6 months, the 4 month guy gets the job. When the software is 4 months late, the overall system complexity makes it easy to assign blame elsewhere, so there's no way to judge mis-management when it's time for promotions.
>
> To look good to their boss, most people tend to put a positive spin on their reports. With many levels of management and increasing optimism all the way up, the information reaching the VPs is very filtered, and always filtered positively.

The problem is that the highly filtered estimates are completely out of line with reality (at least in recent software plans here at SGI), and there are no reality checks back from the VPs to the engineers on the bottom. I think it's great to have aggressive schedules where you try to get things out 20% or so faster than you'd expect. The problem is that in 5.1, the engineers were expected to get things out 80% faster, and it was clearly impossible, so many just gave up.

We certainly didn't win any morale prizes among the engineers with 5.1. It's the first release here at SGI where most of the engineers I talked to are ashamed of the product. There are always a few, but this time there were many. When engineers were asked to come in over the weekends before the 5.1 release to fix show-stopper bugs, I heard a comment like: "Why bother? SGI's going to release it anyway, whether they're fixed or not."

I'm not blaming the engineers. Most of them worked their hearts out for 5.1, and did the best they could, given the circumstances. They'll be happy to buy into a plan where there's a 20% stretch, but not where

there’s an 80% stretch. They figure: "It’s hopeless, and I’ll be late anyway, and I’m not going to get rewarded for that, so why kill myself?"

Marketing - Engineering Disconnect

> "Marketing -- where the rubber meets the sky."
>
> -- Unknown

There’s a disconnect between engineering and marketing. It’s not surprising -- marketing wants all the whiz-bang features, it wants to run in 16 megabytes, and it wants it yesterday. Although engineering would like the same things, it is faced with the reality of time limits, fixed costs, and the laws of nature.

It’s great to have pressure from marketing to do a better job, but at SGI, we often seem to have deadlocks that are simply not resolved. Marketing insists that Indy will work in 16 megabytes and engineering insists that it won’t, but both continue to make their plans without resolving the conflict, so today we’re shipping virtually useless 16 MB systems. Similarly for feature lists, reliability requirements, and deadlines.

Well, at least we met the deadline.

WHAT TO DO -- SHORT TERM (5.1.2)

> "We should sell ’bloat credits’, the way the government sells pollution credits. Everybody’s assigned a certain amount of bloat, and if they go over, they have to purchase bloat credits from some other group that’s been more careful."
>
> -- Bent Hagemark

There are problems in both performance and bugs, and we’d like to fix both. In addition, the first thing we should do is decide exactly what’s going into release 5.1.2.

If we are serious about a December all-platforms release, there may be very little we can do other than keep stumbling along as we have been. Three months isn’t much time to do anything, considering the overhead of a release, where perhaps half of the time will be spent in "code freeze". After 5.1, many engineers are exhausted, and it’s unreasonable to expect them to start hard work immediately. 500 outstanding priority 1 and 2 bugs is a huge list, and we haven’t even begun to hear about customer problems yet.

What Should be in Release 5.1.2:

I’m afraid the answer is going to be "everything that didn’t make it into 5.1". I know that won’t be the case, but I hope that we will carefully select what goes in now, rather than hack things out in a panic in December. The default should be "not included", and we should require a good reason to include things. Let’s make sure that there’s a minimal, solid, working set before we start adding frills.

Improving Performance:

> "SGI software has a cracked engine block, and we’re trying

to fix it with a tune-up."

-- Mark Segal

As stated above, we don't even know exactly what's wrong. We probably never will, but we should start doing things that will have as much of an impact on the problem as possible. I don't think we have time to study the problem in detail and then decide what to do -- we've got to mix the research with doing something about it.

Before we begin, we should have definite performance goals -- lose less than 5% wall-clock time on compiles of some known program over 4.0.5, have shells come up as fast as in 4.0.5, or whatever.

Some people claim that we need new software debugging tools to look at the problem, and that may be true, but it's not a short-term solution, and it runs the risk of causing us to spend all our time designing performance measurement tools, rather than fixing performance.

In fact, I don't really believe that simple "tuning" will make a large dent. To get things to run significantly faster, we've got to make significant changes. And we can't beat the "5% rule" by just speeding up all the systems by 5% -- if everything is exactly 5% faster,
the overall system will be exactly 5% faster.

There's a strong tendency to look for the "quick fix". "Get the code re-arranger to work", or "Put all the non-modifiable strings in shared code space", for example. These ideas are attractive, since they promise to speed up all the code, and they should probably be pursued, but I think we're not going to make a lot of progress until we identify the major software architectural problems and do some massive simplification. Remember that DSOs were the last "quick fix".

There's got to be more to it than tuning; there must be some amazingly bad software architecture -- from a novice's point of view, a 4 MB Macintosh runs a far more efficient, interesting system than a 16 MB Indy. The Mark Segal quote above sums it up.

Code walk-throughs and design reviews are in order for most of our software. The attendees should include not only people working in the same area, but a small cross-section of experienced engineers from other areas. Get a pool of, say, 20 experienced engineers and perhaps 3 at a time would sit in on code reviews together with the other people working in that area.

Code reviews will help in many ways -- the engineer presenting the code will have to understand it thoroughly to present it, others will learn about it, and outside observers will provide different ways to look at the problems.

The most important thing should be the focus -- we're trying to make the code better and faster, not to make it more general, or have new features, or be more reusable, or better structured.

For complex problems, the walk-through should also include some general design review. Are these daemons really necessary? Do we really need this feature? And so on.

Fixing Bugs:

The code walk-throughs will obviously tend to turn up some bugs, so they'll serve a dual purpose.

With 500 or so priority 1 and 2 bugs, we must prioritize these as well. A bug that causes a system crash only on machines with some rare hardware configuration is properly classified priority 1, but it's probably less important than a bug in a popular program like Showcase that causes you to lose your file every tenth time, which would normally rank as priority 2. The effort involved in the fix should also be taken into account. For bugs of equal frequency of occurrence, it's probably better to fix 20 priority 2 bugs than 1 priority 1 bug if the priority 2 bugs are 20 times easier to fix.

A bunch of bugs can be eliminated by getting rid of features. Let's have the courage to cut some of the fat.

WHAT TO DO -- LONG TERM

> "Software quality is not a crime."
>
> -- Unknown (seen on a poster in building 7)

It's easy to go on forever here, but I'll try to limit it to a few key ideas. We don't have to do all these at once, but we'd better start.

Have an overall SGI software plan.

Let's get an architect, or at least a small group of highly technical people, not just managers, to agree on plans for releases. In fact, since the release is a company-wide project, there ought to be company-wide participation in the decisions of what's in a release. The group should include marketing, documentation, engineering, and management and should come up with a compromise that's reasonable to all.

In every case, some attempt must be made to check reasonableness all the way to the bottom. There's a long series of excuses, "Well, that's what my junior VPs told me.", or "That's what my directors/managers/lead engineers/engineers told me." We get killed by the optimist effect, and a disinclination to listen seriously to anyone but our direct reports. Try to imagine the guts it takes for an engineer to go to his director and say: "My manager's out of his mind -- I can't possibly do what he's promised."

Let's try to concentrate on performance and quality, not on new features, especially for the 5.1.2 release. I know from my own experience that when I write good code, I spend 10% of the time adding features, and 90% debugging and tuning them. It's the only way to make quality software. In SGI's recent releases, the opposite proportions are often the rule. It's much easier to add 100 really neat features that don't work than to speed up performance by 1%.

Aim for simplicity in design, not complexity. Make a few things work really well; don't have 1000 flaky programs.

Be willing to cut features; who's going to be more pissed off: a customer who was promised a feature that doesn't appear, or the same customer who gets the promised feature, and after months of

struggling with it, discovers he can’t make it work?

Get better agreement between the top level VPs and the lowest engineers that a given schedule is reasonable.

For new development, continue the formal design reviews and code walk-throughs. These shouldn’t just happen once in the development cycle -- things are bound to change, and code reviews can be very valuable, even for our experienced programmers.

ACKNOWLEDGEMENTS

I take full responsibility for the opinions contained herein, but I’d like to thank Mark Segal, Rosemary Chang, Mary Ann Gallager, Jackie Neider, Sharon Fischler, Henry Moreton, and Jon Livesey for suggestions and comments.