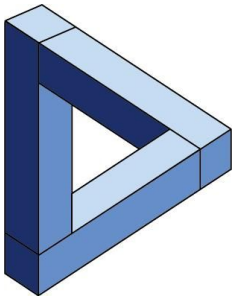


Surviving Change

Building redundancy into the one system that never has backups: the human system

Andrew Frederick Cowie - Operational Dynamics

Abstract

Ever increasing complexity means that people charged with maintaining production systems face an impossible task: they need to spend time developing new ways to manage their platform but are constantly fighting fires which prevent them from concentrating on the real challenges at hand.

In essence, the solution to the problem is a human one. These issues are not unique to systems administration, so this paper takes a far reaching approach to addressing them. Expertise from software development, military operations, the scientific revolution, and even NASA are used.

Three themes are dominant: we need a systematic way to learn from our experiences, bridging structure and flexibility is key, and above all, it is leadership which makes the difference.

Using procedures for specific events offers a breakthrough approach. A formidable tool is created when procedures are used in a way that resists bureaucracy while encouraging knowledge transfer and learning.

Author

Andrew Cowie is an operations consultant based in Sydney. His firm helps clients improve the effectiveness of their technology by focusing on people and the processes around them, driving usability, scalability and maintainability through team building, establishing procedures, and improving systems performance. You can reach him at andrew@operationaldynamics.com

Introduction

Let me tell you a story:

> *It's 3 am, and time is eternal.*[1]
>
> *At least, it would be, if your systems were up.*
>
> *But they're not, because there was runaway process, which ended up causing the machine to reboot. The database didn't come up right. And the guy who knows how to fix it is away at his Aunt Martha's, who, regrettably, lives out somewhere beyond where they ever did manage to get electricity.*
>
> *So instead, your site is off the air, and the clock is ticking. You don't know the exact figure, but you have a funny feeling that if the site isn't back online in about another 60 minutes, you'll be looking for a new job.*
>
> *In a new industry.*
>
> *No pressure, now. Not a bit.*
>
> *We've all been here. It was supposed to be your weekend off. You're the one they woke up. You're groggy from lack of sleep, you're probably hung over and in any case the other guy is the one who always took care of the database.*
>
> *All you have to do is start it.*
>
> *You thought you knew how. Sure, you'd watched over his shoulder a few times. But as you type madly at your keyboard, you realize that there's a minor detail you're missing:*
>
> *The database system password.*
>
> *Oops.*
>
> *So you struggle off in the dark, frantically searching through email archives and source code, hunting for the magic phrase. At last you find it.*
>
> Time elapsed, 20 minutes
>
> *Then you're sitting there. Typing* `sqlplus`. *It always worked for the other guy. After a while, you clue in that you need to have a bunch of environment variable set, that those users have in their .profiles. So you manually get that sorted.*
>
> *Finally, you're connected to the database. You type* `STARTUP`. *And wait.*
>
> ...35 minutes
>
> *It says there's an error.*
>
> *It says it can't find the database.*
>
> *You realize to your horror that the RAID array has gone offline, and that some command you typed when you were trying to get in caused the database instance to failover to the other cluster node and now the disk drives aren't connected to this machine anymore.*
>
> *How do you get the drives back? You've never done a cluster failover before!!*
>
> ...55 minutes

You begin looking around for something to sacrifice because you know there's not much else that's going to save you.

The point of the story is that it didn't have to be that way. The problem is that someone had the knowledge you needed. But they're away. It was a simple thing, really. All you needed was the correct sequence of commands and you would have been able to recover the database quite easily .

So often, we are reluctant to write things down. Everyone knows that programmers hate (and are bad at) documenting their own code. But for Systems Administrators, our code is not in the running applications. Our code is in the procedures and practices that we follow, not just day to day, but also in special and unusual circumstances.

This paper will describe how you can foster a culture of building procedures to record the knowledge not just behind critical events but around routine events too. The tendency of people to "wing it" *can* be overcome. Instead, you and your team can learn that the true hallmark of professionalism: knowing that we are all human and we all make human error mistakes. As such, professionalism means being respectful enough to pull the manual down off the shelf and follow the procedure for what you're trying to do, every time, so that no mistakes are made. And when inevitably some change in the underlying environment occurs and the procedure results in an error, then professionalism means not just bashing your way through a problem and then forgetting about it, but taking careful notes and ensuring the lessons you learn in solving the problem are quantified and used to update the procedure so that the same error doesn't occur in the future.

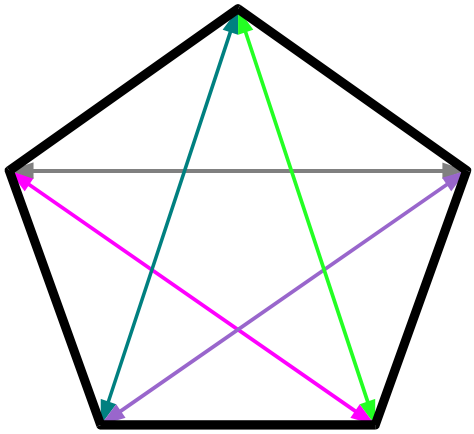
The complexity gap

A common phenomenon, not only in computing or business but throughout organized human endeavor, is a reluctance to write down the details of what we do and and how we go about doing it. The reason is simple: people fear that they will be replaced if they write down their knowledge (presumably with someone "cheaper"). I would not be an astute student of Dilbert if I didn't acknowledge that this sometimes happens – but there is a difference in our world: when was the last time you saw a sysadmin with spare time? As we grow in experience, work should be getting easier, not harder. But just about every operations guy I talk to is exhausted, stressed, and, frankly, bored. How come?

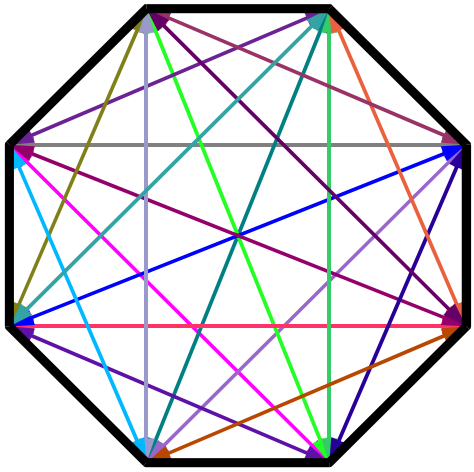
Modern IT environments are some of the most complex systems ever created. They involve layers upon layers which all have to work seamlessly together. Consider the "simple" act of someone viewing a single web page. Even neglecting the minor miracle of getting the request from the user's browser up to their ISP then out through a labyrinth of interconnected networks stretching half way around the world, the request then has to pass through border routers, firewalls, load balancers, web servers, packet directors, applications servers which then make requests to databases, legacy systems and file servers, dynamically generate a page which is then sent back through the entire chain all the way back to Joe user.

Frankly, it's amazing that anyone expects the web to ever work. It is a testament to developers, systems administrators, database gurus, and network engineers the world over that most of the time, it does.

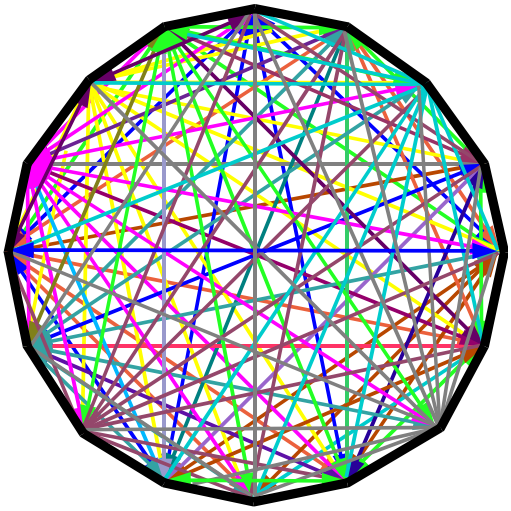
The problem, though, is that people are always making changes. Shifting business requirements, new technologies, increased demand – not to mention changes made for maintenance and upgrade reasons. Any time anyone adds a node to the system, alters the configuration of a router, makes changes to the database, or upgrades the software actually powering the application, complexity increases. In a system where each element is connected to every other element, then each time a node is added the complexity increases geometrically. We can illustrate this with a familiar example from geometry:



Pentagon: Only 5 sides, but there are actually n(n-1) = 20 interactions between the nodes. Slightly unexpected, perhaps, but still manageable.



Octagon: 8 nodes, but now 56 connections between them. Complexity increasing!



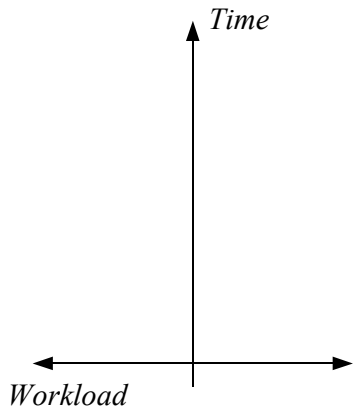
Spiraling out of control: 16 nodes but an overwhelming 240 interconnections.

As described above, the degree of interconnectedness in production IT systems is very high. Because of this, each change event, however minor, increases exponentially the complexity of the overall platform. We all know changes are happening all the time.

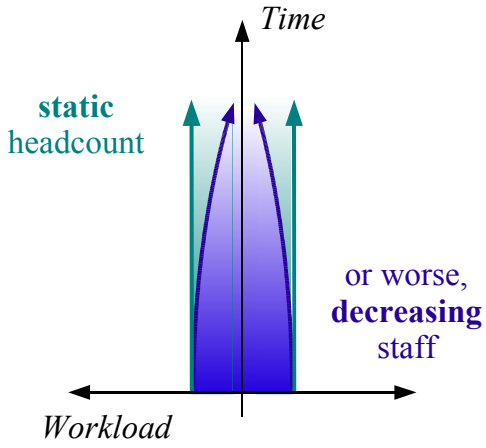
And who is it that has to look after the effects of these changes?

Us.

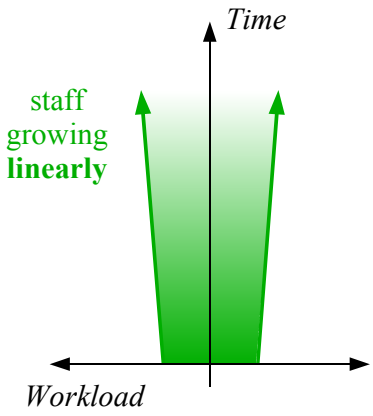
You see, demand for our time is increasing exponentially, but our resources aren't. Which means what? Consider a chart with time on the vertical axis and level of effort required shown horizontally, like this:



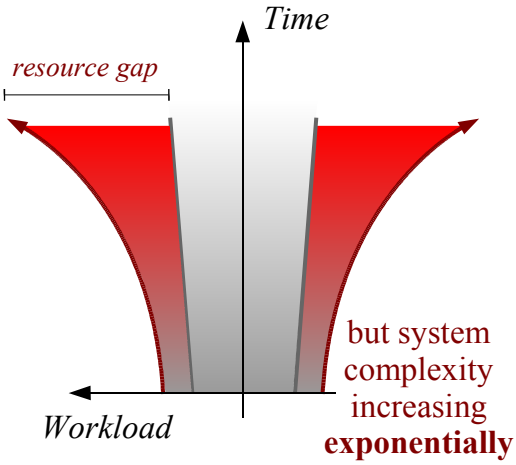
It's not like the number of people you have to get all the work done is not exactly growing. In fact, in this day and age, with the constant pressure to trim expenses and control costs, you're lucky if you're not loosing headcount as time goes on.



Even if your staff are growing linearly over time (unlikely),



it's still not enough:



Ever increasing complexity means the gap between how much time you have and how much work needs to be done to maintain the systems is constantly growing. You need to find ways to be more productive. Incremental, linear change is not sufficient.

A common problem in systems administration is that conceptually straight forward tasks end up being moderately complex in the details of their execution. Merging from CVS branches, updating Apache web server configurations, even (gasp) adding new user accounts. Because of this sensitivity to getting the fine details right, senior staff continue to be directly involved in doing these admittedly menial tasks long after they should have been devolved. As a result, these experienced people are necessary and relied upon to preform the process. So forget about vacation..

This conversation started with the fear expressed by some systems types that if they record their knowledge they will be made redundant. They like the idea of being “mission critical” because they think it makes them irreplaceable. With the differential between workload complexity and available resources constantly increasing, organizations cannot afford to be firing their best people. The challenge they have is to find ways to bridge the ever widening gap. The only way to do this is for our most experienced, senior systems administrators, systems programmers, database administrators and network engineers to work together to develop new techniques which will result in visionary breakthroughs in the level of productivity. They can't do this if they're mired in fighting fires, doing menial tasks, and constantly rushing from one emergency to another. And finally, when

you really get down to it, would you *rather* be doing silly repetitive things or doing something new and exciting which is directly aimed at improving the quality of life for you and your colleagues? [To managers, I ask: "Why do you think so many sysadmins spend so much time contributing to global open source projects?" One reason is that they're not getting any creative stimulation or challenge at work. Put in those terms, is it any surprise that people are bored, unmotivated, and ultimately not particularly inclined to stick around if contacted by a headhunter.[†]]

So how can we do better?

The age of the digital sculptors

Learning from Programmers

I know, it sounds like a horrible thought. Most sysadmins I know look upon programmers as anathema. The problems that programmers face, however, are more immediate and more overt, and as such they have developed approaches to deal with them. The world of systems administration is different, you might say?

Programmers have to work together in teams. They have to communicate effectively with each other to do so. They collaborate on a common codebase. And, above all, they work to achieve the goals on time if possible.

The last time I checked, sysadmins have to do exactly the same things!

In our work, one of the clearest red flags that an operations staff isn't running well is when we hear "Hey, who changed that?" The person making the exclamation feels that they "own" that particular part of the code or systems. This is no good: while it may seem that "you know who to go to" to solve a particular problem, all this you've really achieved is that in the long run is to make that person a bottleneck. As complexity increases, they will be increasingly required to exclusively support that system. The result is an employee who can never take a vacation.

Systems administrators take it as vogue (although a good practice) to put configurations into some kind of version control system. The trouble is that a) often this is the initiative of just one person, and b) there is usually nothing that enforces or requires the use of such tools. Programmers live in an environment where they depend on the shared code repository. They rely on those tools to incorporate their work with that of their colleagues, and for their collective work to be tested and the ultimately deployed to production. Sysadmins, on the other hand, often have direct access to the production machines, and are often able to "circumvent" the formal build and deployment systems in case of an emergency. While this typically seems a good idea in the heat of the moment, a common problem is that the next time the automated systems run, these local modifications are overwritten and the system breaks again.

The key to avoiding these scenarios is ensuring you have means in place to effectively share ownership. That means mechanisms to facilitate capturing knowledge and procedures. Tools which allow ad-hoc changes to be made in emergencies but which persist the effects of those changes across normal system changes. Above all, it means ensuring that however your platform is architected, a spirit of collaboration pervades.

The key is communication. Programmers, while they make extensive use of tools which help them, ultimately cannot avoid the necessity to interact to solve problems, one human to another. Sysadmins sometimes miss this point. If you want to survive change, then figure out ways to communicate effectively with your team mates.

[†] A phenomenon known as "warm chair attrition". See [15].

Extreme Programming

Once it might have been taken as presumptuous to say so, but today, most software development projects are late, have cost overruns, fail to meet expectations, and are bug ridden. If you're *very* lucky, the product might work when delivered.

In the last five years, a set of software development methodologies have arisen which somewhat dramatically challenge the status-quo. Certainly the explosive growth in popularity and success of Open Source software developed in distributed collaborative settings shows that conventional closed in-house development is not the only way to create quality software. In terms of successful practices we can learn from programmers, through, a set of practices known as Agile Programming – the most radical of which goes by the moniker *Extreme Programming*.

Some of the practices advocated could well be of use in the systems world. The following list, written by Bruce Tate in an excellent book about the pitfalls found in Java software development (appropriately titled *Bitter Java*),[12] gives us a good chance to reflect on what we can do better as operations professionals:

- Choose simple solutions

  "Simple solutions are less likely to create antipatterns, or to mask those which already exist." [2]

  How often to we dream up horribly complex solutions then wonder why users are constantly complaining, why they are difficult to maintain and why we have no time to work on new projects?

- Ensure that the customers are on site.

  "End users provide guidance, insight, and opinions."

  Customers? Who? If we create processes that people inside our company have to follow in order to get what they need from our systems, then *they* are our customers. They are the ones that actually have to use the system, so involve them early.

- Write user stories

  If you can jot down, in just a few sentences, what the user experience is going to be interacting with your group, your systems, or some process interface you are creating, then you're on the right track. *Share* this with people – other staff on your team, your boss, and above all with the people who are going to be using it. If they get it, and if they think it's ok, then you're on the right track. If they don't, then *listen* to them.

- Divide larger projects into measured, planned, small releases

  "Smaller cycles result in user feedback, diminished risk, and adaptability to change"

  The temptation is to often accumulate a list of needed changes and then attempt to execute all of them in one single composite event. Pressures to minimize the risk of downtime exacerbate this. Unfortunately this drives up the complexity of such events, making it harder to isolate problems. If you can keep events discrete, and have good telemetry from your systems, then you can troubleshoot more effectively.

- Program in pairs

  "This practice seems wasteful but has enormous power. Pair programming improves quality and reduces tunnel vision. Antipatterns are more likely to be spotted with an extra set of eyes"

In systems work, this can dramatically increase productivity and reduce the kinds of small human errors which can wreak so much havoc on computer platforms. It gives a person working on a critical system someone to bounce ideas off of and verify the sensibility of actions. Two heads are better than one because you get different perspectives, different ideas, and different experiences looking at a problem. And, above all, it means that there is more than one person who knows what changes were made and why.

If you take anything away from this paper at all, I hope it will be the importance of passing knowledge on – and this is a hugely effective way of beginning the process.

- Code test cases before the rest of the system.

  This is a neat one. In conventional software development, normally what happens is that code is written, then tests are created, and QA is done, to see if it works. Agile programming advocates reversing this process: write the tests first. It takes a bit of getting used to, but if you work out what the interfaces to the systems should be, and how the the outputs should behave, then you are actually well placed to subsequently be able to concentrate on developing the system to satisfy these requirements. Indeed, there is a school of thought that goes so far as to say that when all the unit tests pass, then by definition the code is done.

  I think we can learn a couple of things from this. Certainly one of the biggest problems in operations is simply knowing when something is broken. If we can establish effective monitoring, telemetry and alarm event notification systems from the outset, and make maintaining and updating those systems a rigorous part of our change management, then we are a lot more likely to know when a casual, supposedly unrelated change causes problems. This doesn't have to be expensive – there are some excellent open source projects which can do the job really well.[20],[21] Of course, if you can predict in advance that things are going to fail, then you're really cooking. Integrate QA's staging and testing systems with the production monitoring defences and you can get there.

  The other area this leads us to is being able to measure the impact of our changes, which will be discussed in a later section.

- Do not use overtime.

  "Overtime increases available hours and reduces clear thinking – with predictable results"

  Any questions?

  Seriously: take care of yourself and your people. Don't burn the midnight oil just because you think it's the macho thing to do. If you're tired, you're more likely to make mistakes. People need a mental break from work – especially from high pressure environments such as in IT. At the end of the day if you continually drive yourself or your people into overtime, you will lose the extra productivity through fatigue, and lower effectiveness.

None of these ideas are particularly new; in fact, the discipline of programming has been well established for over 30 years. Part of the problem is that many of us have: either forgotten what we learned in school (brushed aside by day to day work pressures), or, we picked up programming [or systems] on our own and never had an opportunity to study and learn from the early pioneers and masters of our field.[3] Continually striving to learn new things not just in our own narrow specialties, but across a broad range of disciplines, is ultimately one of the best tools we have to prepare ourselves for change .

Feel the Burn

One of the major operational challenges faced by most organizations is that programmers, sysadmins, database people and network engineers are absolutely horrible at communicating with each other. How often have you been on-call all weekend trying to deal with the systems melting down because of some bug in the application code? The part that I love watching is the scene Monday morning:

> *Chief programmer: "So how was your weekend?"*
>
> *Sysadmin: "Ugnnnn."*
>
> *Chief programmer: "Oh, that doesn't sound to good. Did something happen?*
>
> *Sysadmin: "Yeah – the lousy code that **you** wrote caused the Live systems to crash every 30 minutes **all** weekend. My team and I haven't had any sleep in days."*
>
> *Chief programmer: "Really, wow. That's too bad. I was out partying all night Saturday, then slept it off lying on the beach Sunday."*
>
> *[Spectators stand back as sysadmin launches herself for chief programmer's throat]*

The trouble this scenario alludes to is that the programmers had no stake in the running of the production systems. To be blunt, they didn't feel the pain of operators. What incentive could they possibly have to make sure their code [changes] are truly going to stand up to the demands of production load? This goes further than the conventional "oh, well, if we *really* have to, we can put a programmer on-call too." No, in order to avoid this scenario in the first place, the programmers and the sysadmins (and indeed database administrators, network operations staff, and even the marketing group, account reps and product development people) all must have an involvement in the successful running of the company's technology platform – and to achieve this means forming teams that are cross disciplinary and building a culture in which people support each other.

Which brings us to leadership and team building.

The Profession of Arms

Learning from Soldiers

Unfortunately, what most of us know about the military comes from what we see in the movies. Lots of people yelling at each other. The hero dodging hails of bullets. Good entertainment. Regrettably, Hollywood's rendition of the Profession of Arms misses out on a few things. For the purposes of this discussion, I'd like to bring two to your attention.

Teach them and they will come

One aspect of a profession is that it holds a specialized sector of knowledge that is not held or practiced by other members of society. The military arts, by their vary nature, are a specialty that you can't just go to the local convenience store and pick up a book to learn. Rather, through the centuries, armies the world over have become adept at something that most people don't recognize them for: *teaching*.

The army cannot just hire Colonels or Regimental Sergeants-Major off the street. Instead, they have to grow them. Within their own system they have to give the education, training, and experience necessary to enable young Lieutenants and Privates to someday become seasoned officers and soldiers.

What does that have to do with systems administrators? Let me ask you this: Is Systems

Administration a profession or a trade? In the end it depends what definition of “profession” we use, but the fact that engineers, surgeons, lawyers, are all members of established professions, whereas physicists and chemists have long been trying to gain recognition professionals and failing. The difference? The pursuit of science, despite its noble cause and worthy outcomes, is, in and of itself, not directly aimed at benefiting and servicing *society*. If you're a sysadmin, ask yourself: am I acting as a professional? Are my actions helping the company towards its objectives, or am I simply serving myself and my own interests?

While the barrier-to-entry to becoming a systems administrator is dramatically lower than becoming a soldier (indeed, install Linux on your personal machine and you've no choice but to start down the road of learning these skills), there's a long jump between someone who is “god in their own universe” (their desktop) and someone capable of holding the responsibility for maintaining and scaling your platform and architecting its future.

> *But my platform is different. It's not like anyone else's. It's totally custom.*

Of course it is – that's the point. Just like the army can't hire a colonel off the street, you can't hire someone who is already an expert in your systems – you have to grow them, *teach* them yourself.

Systems people are very much in a position of serving others. They share an arcane knowledge. You need a means to pass on that knowledge. They, as professionals, need to be a part of the solution.

Decision Making Under Stress

I love it when people tell me that “they could never handle life in the military – its too disciplined”. Ha. When was the last time you saw someone tell their CEO “no”? The private sector is far more autocratic than a combat unit ever is. Without a doubt soldiers need to come together and when the pressure is on their trust in their team and in their leadership is what allows them to react, but believe me, when the consequences involve lives people are very deliberate about making sure that the right decisions are being taken. Sadly, in the corporate world, too often it is whim which governs.

What's interesting is that the skills necessary for the combat arms are in many ways the same skills you need to be effective in IT operations. Decisiveness. Ability to handle time pressure and high stress. Intense knowledge of all aspects of not just one's specialty but how it fits into the broader picture. And the strength of character to take responsibility for what happens, and not just pass the buck.

The one constant that leaders face in the field is incomplete information. While history and large scale maps seem to make it obvious what so-in-so should have done, the reality on the ground is very different. You have to make decisions *now*, without the benefit of perfect knowledge. You can only see to the next ridge line. You hear reports on the radio. It's dark, raining, and cold. And your decisions could have the ultimate consequences.

Having to make decisions under stress with incomplete information is actually a hallmark of the IT production environment systems people face every day. While we like to think that we can reason things through and come up with the best possible solutions, two things conspire against this.

First is time available. Sometimes we simply don't have the time to research all the options. Perhaps it's a startup company and despite the insane hours everyone is working, everything is being done on a crash-priority basis and you simply have to make the best decision you can at the time. Even in calmer circumstances, decisions with far reaching consequences need to be taken. Database table space allocations is a classic example. Despite the injunction to “optimize later”, taking a production database off line to redo the allocations for greater space efficiency, optimize extent growth parameters, and to try to improve performance is almost impossible. Redoing the database's underlying structure implies a) having somewhere of sufficient size to copy it and b) an opportunity

to actually take the production system off line for as long as the restructuring will take. In today's 24/7 operations environment? Almost impossible. So, despite the fact that you won't know how best to structure the system until you have a realistic notion of the what kind of load will be placed on it, you will nevertheless have to make decisions when first constructing the platform that will be with you for a long time to come.

The other factor forcing our hand is changing requirements. It doesn't matter if something made perfect sense 6 months ago – if the organization's business objectives have shifted, then the IT supporting the organization needs to shift as well.

Top Gun

Studying the decision making processes of fighter pilots engaged in dog fights, US Air Force Colonel John Boyd observed that there was a clear sequence which pilots go through when faced with complex, high stress, uncertain (not to mention dangerous!) environments. The Boyd Cycle, or "OODA loop" as it is known, is:

1. **O**bserve – an event is noted (a pilot sees a fighter, sonar reports a new contact, a tank commander unexpectedly sees a tank in a wood line)
2. **O**rient – the decision maker fits this new fact into their view of the situation around them (the fighter must be an enemy – it was unexpected as it is diving from the direction of the sun, the tank is recognized to be an *enemy* tank – and not in a place where the enemy was expected to be!)
3. **D**ecide – a course of action has to be chosen. What are you going to do? (Decide to evade? Decide to stop the advance you were on and turn to face this new threat?)
4. **A**ct – actually *do* something as a result of the decision that was made (Break formation and dive for safety! Signal the other tanks in your squadron and order them to turn, face, and charge!)

If you're having trouble understanding, try imaging a game of chess. Each move presents a new situation, and as the game progresses both players iterate through this cycle. Whomever can do it faster and better will gain an advantage – they will "get inside" the other's decision cycle, and ultimately the other will find their pieces out of place and their actions irrelevant. Checkmate.

Believe it or not, the Boyd Cycle heralded a revolution in military affairs.[4] It revealed that there was a common thread throughout combat decision making. Regardless of it being combat on the ground, in the air, or at sea; irrespective of whether it was small groups of soldiers or large fleets of capital ships: uncertainty manifests itself as the difference between information and intelligence. In terms of the commander of an army formation on the ground, *information does not become intelligence until it has an impact on that commander's decision making.* Gathering endless information only has value if it can be collated, sorted, organized, and presented in a fashion that either allows it to fit into our existing understanding of the situation *or forces us to change our orientation.*

Suddenly, we now have a bridge between the world of military operations and that of IT systems operations. If there is a thing we all face in our hyperkenetic age,[5] it is information overload. We are bombarded with advertisements, commercials, emails all demanding our attention. We glance at newspapers, television, and online sources and yet despite more information being available to the average person than at any time previously in history, we feel overwhelmed. If anything it is even more difficult than in the past find to find the information we need to make the right choices.

In that context, then, we can understand that a piece of data being out there, somewhere, does not have any value unless we are cognisant of it, and it can become a factor in our decision making. It gives us strong clues about what we need to do to manage our systems in the face of change.

A final lesson we can draw from the OODA loop is that the examples I used to illustrate it all revolved around surprise. *I was surprised to see that tank – an enemy one at that! - off to my left where I didn't expect one to be* or even *Check? I didn't expect that Knight to jump there. Now what do I do?* The unexpected can and will happen at every turn. No matter how much preparation has been done, no matter how thorough your plans, the impact of real time running will present challenges you didn't expect. In order to cope with the inevitable, you need to be flexible – in your outlook, in your procedures, and in your teamwork.

Leadership in Action

In a fantastic book called *Hope is not a Method*,[8] Gordon Sullivan (a former Chief of Staff of the US Army) and Michael Harper extended the the Boyd Cycle and presented it in terms that make sense in the business world. Someone in a position of responsibility faced with high pressure, high criticality decisions needs to iterate through these steps, which they called the "Leadership Action Cycle":

1. **O**bserve – asking these questions: *"What is happening? What is not happening?"*, the leader looks both outside the organization, and looks within at strengths, weaknesses, basic competencies, cultural tendencies and needs.
2. **R**eflect – *"What can I do to influence the action?"* Interpret the information gathered. Identify threats and deduce opportunities, alternatives, options, and courses of action. This is where objectives are established, uncertainties are separated from relative certainties, and risk is assessed.
3. **D**ecide – determine tasks, roles of key people, constraints, limits and measurable standards for success.
4. **A**ct – the organization begins to execute the leader's decisions. Pilot projects can be useful to permit early learnings to be fed back into the process. The leadership must be personally involved, especially in the change process – the leader's sponsorship and involvement demonstrate the importance of change and reinforce the need for participation by managers at all levels.
5. **L**earn – relate the outcomes of decision and action to the environment and to future action. Leader and organization modify their behavior to be more effective, adjusting and refocusing by asking *"If we knew then what we know now, what would we do differently?"* [6]

In terms of the challenges placed on us as systems administrators in a high pressure world, the things that stand out from their work are the need to be able to measure, and the need to for leadership to be directly involved. These are the subjects of the next two sections.

Above all, one of the things the US Army learned in the wake of their defeat in Vietnam[7] was the importance of honest appraisal of what happened, thorough consideration of what lessons were learned, and then **integrating those lessons into their training and doctrine**.[5]

The Art and Science of Measurement

We've come a long way since the Renaissance, right?

How many times have you seen this one?

> *Fred is trying to get anonymous FTP running. He's tries connecting, and it doesn't work. So all at once he changes:*
>
> *a) the anonymous section of the ftp daemon's configuration,*
>
> *b) the port-binding section of the config file (because he thinks "that might also be the problem"),*
>
> *c) the host's networking setup, and*
>
> *d) the password file, ("just in case").*

Anyone care to bet whether this joker will solve his problem?

Even assuming one of these changes makes a difference, how is this person going to have any idea which thing he changed actually made it work? And there's an even worse eventuality: changing 14 things at once may have resulted in a fix for the acute problem he was suffering, but how much would you like to bet that he broke something else? Since he's not looking there just at the moment, he won't notice – but his users certainly will.

In the end, it's all boils down to the scientific method. That's something we were supposed to have learned about 400 years ago, but somehow we keep forgetting about it.

In order to get anywhere when trying to troubleshoot problems, we have to be both methodical and disciplined. It goes like this: form a hypothesis of what is wrong, estimate what you can measure to find out precisely what the problem is, then change things, *one at a time*, and look at the measurements to determine whether or not your change influenced the problem you're having.

Although this is fairly straight forward for one person, and while many say "but of course I'm methodical when I solve problems," it is remarkable how often we observe people "fixing" systems changing by altering many configuration elements at once. The real problem, however, becomes apparent when scaled up to team size. In a crisis, even assuming your people are are thinking straight, too often they are acting as individuals. The sum of their actions is the same chaos as before – only this time individual staff aren't necessarily aware of what the others are doing – with the result that their actions will inevitably collide and make things worse. This is what you have to beat if you want to survive change.

There is, regrettably, no quick fix for this. It is a management and leadership issue to ensure you have effective communications practices in your operations teams. But it does underscore the importance of developing common approaches to solving problems, and just as importantly, having mechanisms to document your experiences so you can apply what you have learned to future situations.

A Struggle for Balance

One of the most significant trends in management literature in recent years has been the "Balanced Scorecard",[7] a way of tying broad strategy with specific initiatives. It helps individuals understand how their actions contribute to driving the organization's success, and helps them stay focused on those goals. It is built on the premise that "you can't manage what you can't measure, and you can't measure what you can't describe".[8] While the first half is common to cost accounting the world over, the second part represents a breakthrough in management thinking. It suggests that many

things are difficult to express and as a result, obvious quantitative measures may not be readily available.

It is human nature to shy away from the things that are difficult to conceptualize. Certainly, the trends in corporate management for the last 100 years have focused on “scientific” measurement powering production and operations analysis. For example, this focus has been almost entirely on financial aspects largely to the neglect of factors like morale and long term development. Things that are difficult to describe in conventional terms are easier to ignore, and tangible dollars and cents are as conventional as it gets. When your balance sheet has gaping holes in it and your profit n' loss is hemorrhaging cash, it is abundantly clear that action needs to be taken immediately rather than after the bankruptcy hearings! The trouble is that when an organization is in this kind of trouble, focus tends to be on things financial to the exclusion of all else. Actions taken tend to impact very negatively on the morale, well being, and motivation of the employees. While everyone pays lip service to the fact that “obviously” these factors are important to productivity, it is a rare organization which actually maintains them through tough times, *because* they are so hard to measure.

Difficulty in measurement is only half the problem. Actually it's an excuse – and a rather pathetic one at that – for absence of leadership.

Can you hear me Mission Control?

Learning from NASA

“Failure is not an option”. The movie *Apollo 13* made the phrase, and Gene Kranz (the Flight Director, played by Ed Harris), famous. Few would contest the extreme professionalism or technical caliber of the people involved in the American space program. Yet, there they were, meticulously referring to their written procedures as they went through events both critical and routine.

The attitude that pervaded the entire space program throughout those years was, “It won't fail because of me.”[9] This determination indeed led them to work insane hours, but I believe that a difference exists between what they were doing and what many of us, working the same crazy hours, do. They were focused on things that really mattered. And that was because of outstanding leadership at all levels of the program.

As we go about our daily work, how many of the tasks we spend time on are unimportant to the job of getting the organization's work done, or worse, are make-work type jobs? Without a doubt there will always be pressure from those inclined to bureaucracy and mediocrity. Such things are often cloaked in the guise of “mitigating risk” but risk is an operational quantity and is best defined by those facing the realities of their work.

Trust

At the beginning of 1969, the schedule pressure faced by those in the American space program was incredible. They had but one year left to achieve Kennedy's goal of landing on the Moon by the end of the decade. Astronauts had to master all aspects of incredibly complex missions. The staff in mission control were double and triple hatted, running flights as well as trying to prepare for subsequent missions. Leading up to the flight of Apollo 9, the first manned on orbit test of the Lunar Module craft, Commander Jim McDivitt:

> *“established another first by designating astronaut Stu Roosa as the full time representative of the crew for all topics. Given the complexity of the mission, McDivitt felt there was too great a chance for something to slip through the cracks. These steps may seem elementary, but in the rapidly moving flight program, we were hard pressed to do anything but the fundamentals for each mission. We were learning by doing, with*

little time to reflect, only to respond."[10]

While most of us in the IT world would not portray ourselves as being in something as critical as space launch operations, the tenor of the passage above is familiar territory. As we struggle to cope with the ever increasing complexity of our world, we have no choice but to strive to improve our competence not only in the technology behind computers, but also in the profession of operations.

> *"For Flight Directors and CapComs,*[†] *the principal tools used during the mission were the MCC intercom and the crew voice loops. Our common job was to listen, integrate, communicate, and act."* [11]

The similarities between Kranz's observations about the process of operations leadership and Boyd's OODA loop are striking.

On 11 April 1970, the Apollo 13 spacecraft suffered a massive internal explosion. Within minutes of the incident, the controllers in Huston had to rapidly make decisions about what path to follow to get the ship back. The decision came down to Gene Kranz and the other Flight Directors. They chose not to risk lighting the main engine, but instead to take the much longer route of looping around the Moon and then coasting back to Earth, even though at first glance there was not enough oxygen, water, or power to make it back. Kranz commented,

> *"Many people were unaware of the options, but I believed the systems controllers thought I had made the wrong decision. They favored the fastest way home, a direct abort.*
>
> *Missions run on trust. Trust allows the crew and team to make the minutes and seconds count in a crisis. In the scramble to evacuate the command module, we didn't have a chance to brief the crew or even get their opinion on the return path ... the flight dynamics controllers returned to their consoles to start developing the return trajectory plan... The systems guys would have to find a way back with what we had."*[12]

Trust doesn't just come from nothing. It builds from shared experience, from learning to rely on other people, from developing the confidence in your people's decisions. This is why rehearsals are so important, and why downsizing and high turnover can have such a devastating effect on an organization's effectiveness – when teams are constantly disrupted they can't build up the bonds of trust that allow them to operate effectively when during a crisis.

Lessons Learned

The other major element to be drawn from NASA about operations is that after an event is over, they are rigorous about analyzing what happened. They review telemetry, examine variations from their planned sequence of events and try and determine the reasons behind them. This isn't from an inflexible "figure out who to blame because no changes are allowed" perspective, but rather is an honest attempt to learn.

Carrying out these kinds of reviews imply a considerable burden on your team – but this effort is necessary if you expect to meet the challenge of supporting your organization as it grows. Indeed, your staffing level is on the mark when you have the people you need to undertake planning and review of critical activities, not just to maintain a tenuous grasp on day to day problems.

One of the best encapsulations of the demands of operations is found in the *Foundations of Mission Control*. As you read it, you'll realize that leadership is not just something which is imposed "from above" on a team of people, but is also a quality which comes from within:

[†] *CapCom:* Capsule Communicator, the one talking to the spacecraft on the air to ground radio loop, and by tradition always a fellow astronaut. *MCC:* Mission Control Center.

"**To instill within ourselves these qualities essential for professional excellence:**

Discipline: Being able to follow as well as lead, knowing that we must master ourselves before we can master our task
Competence: There being no substitute for total preparation and complete dedication, for space will not tolerate the careless or indifferent
Confidence: Believing in ourselves as well as others, knowing that we must master fear and hesitation before we can succeed.
Responsibility: Realizing that it cannot be shifted to others, for it belongs to each of us; we must answer for what we do, or fail to do.
Toughness: Taking a stand when we must, to try again, and again even if it means following a more difficult path.
Teamwork: Respecting and utilizing the ability of others, realizing that we work toward a common goal, for success depends on the efforts of all.

To always be aware that suddenly and unexpectedly we may find ourselves in a role where our performance has the ultimate consequences.

To recognize that the greatest error is not to have tried and failed, but that in trying we did not give it our best effort."[13]

These words hang on the wall of NASA Mission Control in Huston. They represent the ultimate standard in operations leadership. To us they are an inspiration and give us something to strive for.

Theory and Practice

Up to this point, we have discussed the importance of leadership, of teaching and of facilitating the growth of your own people, of measurement, and of having a well described way of capturing what you learn through experience. We've hinted that a key part of tying all this together is procedures. And while effective leadership will always be the cornerstone of organizational success, procedures can give you the lever you need to survive change.

Massive Changes and Upgrades

Major changes are a significant part of the life-cycle of any large site:

- software upgrades,

  ranging from major application upgrades to upgrades of the entire underlying operating systems. On servers this is a relatively contained challenge, (though one effecting critical systems) but upgrading user workstations (desktops and laptops) can be nightmare – there is huge variation in individual requirements, differing hardware, and systems being available at different times;

- launches of new versions of websites,

  which often not only includes upgrading the code for the dynamic pages, but also typically involves deploying changes to static content such as images, and frequently involves database schema changes to support the new code;

- database administration,

  such as a system wide reworking of table space configurations of a live production database, or an upgrade of the underlying database software itself;

- deploying security patches,

  once or twice, no big deal – multiple issues times thousands of machines becomes a major infrastructure management issue; and

- hardware refresh,

  such as the in-situ replacement of the entire suite of networking gear (switches, routers, firewalls, everything) at the expiry of that equipments' lease. It can be challenging to keep a production site running and connected to the Internet when you rip the switching fabric out from underneath it.

Taken as a group, these are surprisingly common situations. In fact, it gives reason to wonder why they are so often considered "unusual" or "special", when in fact much of the attention of systems people is taken by preparing and executing these tasks.

These types of events all share a number of characteristics. They are complex, involving numerous interdependent systems and often include people both internal and external to the team carrying out the procedure. They impact critical services that the organization depends on for its day to day operations. As such the event can only be allowed to disrupt those services minimally, if at all. And they simply aren't allowed to go wrong.

Because of all these factors, massive change events require considerable advance planning. The need to "get it right the first time" and the fact that numerous people need to be co-ordinated in their interactions drives the necessity to clearly document the procedures for such events.

Procedures

"A plan is simply a common basis for change"[14]

Properly created, a procedure gives context. It is perhaps a cliché, but the aim of a procedure is simply to get everyone participating in the event "on the same page". To be effective, a procedure has to meet two aims. It has to provide:

- **structure**, so that all participating can see both the overall picture and their place within it; and
- **flexibility**, so those responsible for executing the procedure can adapt to changing circumstances.

Despite the fact that these two aims are potentially in conflict, with care it is nevertheless quite possible to craft a procedure which embodies both of these aspects.

The value of being able to extract a clearly summarized, high level representation of what the process is about is often underestimated. Some organizations publish detailed and verbose procedures but they are impossible to understand because one loses the "forest for the trees". By presenting a general outline of the process, then people will be able to better understand the context of their actions. An outline summary also forms an excellent basis for discussing a proposed procedure with senior management. By focusing on overall impact, risk points, and goals to be accomplished, you help your organization's senior leadership understand the reasons for carrying out a procedure, accept any risk involved, and sign off on it. These are, after all, events which will impact the production systems which underly their business – and as an added bonus from the perspective of those in the trenches, it goes a long way to helping management understand what you're going through.

Start Small

> *Comprehensive procedures for everything may be fine and nice for NASA, but in our shop things change too fast for us to have time to stop everything and go on a process documentation binge!*

You don't have to.

While one of the reasons things are going too fast for you is a lack of concrete procedures (which is a symptom of the bigger problem – inability to cope with change because the team isn't able to effectively leverage the true talent and ability of its most experience people), dropping everything just to write procedures isn't going to solve any problems. You would be ineffective in the effort unless you and your team had developed effective means to leverage your processes.

A better alternative is at hand. A wise man once said,

> *"premature optimization is the root of all evil"*[15]

so, rather than trying to implement mad change control in one fell swoop, start small!

Pick a major change event with which you have experience and that you understand well, but for which documenting the sequence of activity is worth capturing. Learn what works for you and what doesn't. Find a style of recording the actions that need to be taken. Figure out where you're going to store your procedures. Put them under version control, and encourage others from your team to contribute.

When you run the event, take notes! If you're going to be too busy being a part of what is going to happen to take notes, then draft someone else to help. Figure out how long each step took. Did people stick to the sequence, or did they wander off in different directions? Was the plan actually sufficient to cover the event, or did you have to improvise to deal with unforeseen circumstances?

After the dust has settled, bring your team together for an after-action review. Gather feedback about what worked and what didn't. Try and agree on things to do better next time. Then **record what you have learned by updating the procedure**.

The point has been raised numerous times throughout this paper: if you don't have an institutionalized way of learning from your experiences, then each time you undertake an event it will be as if you are doing it for the first time. It is not a crime to make a error – mistakes are a part of being human. But to make the same mistake again? That is truly inexcusable.

Put another way, a classic example from operations theory is *if you haven't changed the system, and you feed it the same inputs, it is foolishness to expect different outputs!* The only way to change the system is to learn from your experiences and to find a way to incorporate those learnings so that all those around you and indeed ultimately the whole organization learns. You need a substrate in which to record your experiences. Procedures can be your medium to record your experiences and become the foundation of your operations.

Fighting the tyranny of bureaucracy

Anyone who has worked in a large organization, corporate or otherwise, has probably read the proceeding sections and thought "that's all fine and nice, but in the end all that you'll get for your trouble are documents which aren't followed." Seeing that procedures are used, seeing that they remain effective and appropriate, and seeing that they don't become out of date, is actually a leadership function. If your people are using procedures to guide them then it's pretty obvious that you as their manager need to be very confident that the procedures are helping them – and the organization – and not hindering them instead.

We've all been on the wrong end of processes, forms, and documentation requirements gone horribly wrong. We recently came across an example of a six page document (including a form with lots of really small print) specifying the required process that departments had to follow if they wished IT to *add a printer*.[14] Unbelievable, but true.

Which highlights an important point: procedures in our operations world are not about passing ISO 9000 audits. They're about capturing knowledge, giving structure to complex events, and providing a forum for incorporating learnings. Most of all, they are for *us*. If you focus recording procedures for *specific point activities* then you will create a library which will help you build your team and strengthen your effectiveness. If you instead write long laborious documents which go into great detail about how various long cycle cross department activities are to occur, I can guarantee only one thing: they won't be read and they most certainly won't be used.

Maintaining flexibility is the key – and very challenging. If your own procedures stop being a tool which enable you and instead become a burden; if you find that all you've done is fill binders on the shelf which you forget to update, then take the time to pause. Step back, and either find a way to make them more relevant, or retire them and start afresh capturing your activities and your learnings.

I'd like to close with "Andrew's Dilbert Hypothesis":

> Why is it that everyone finds *Dilbert* funny? Because we can all relate to it: humans self-organize inefficiently. Therefore, the only way to not have your work environment end up a nightmare is to actively and continually fight against senseless bureaucracy.

Only implement formal processes for things that make sense. Be agile in the use and combination of procedures. Allow common sense to prevent you from slavishly following formalities that don't make any sense. Respect the well being and best interest of your people, and finally, for those of you in management, exercise some honest-to-God leadership and overcome inertia.

Conclusion

In 1984 Eli Goldratt wrote a book which revolutionized manufacturing. Called *The Goal*,[1] it introduced "The Theory of Constraints" which demonstrated that maximizing the productivity of each element of a system was *not* the way to maximize the output of the system as a whole. It revealed instead that there are bottlenecks in a plant and that they (and only they) directly control the flow of goods through to market. In order to optimize the plant to reach the goals of the business, the bottlenecks need to be optimized and the rest of a plant structured around them.

In our service industry world, it might be tempting to think that these ideas don't apply to us. But we and the platforms we manage represent systems. As a whole, we too are governed by the theory of constraints. I would argue that the bottleneck for all of us as systems professionals is *time*. We never have enough of it. We need to have our best, most experienced people working creatively and intelligently on developing ways to better leverage available resources – which is only way to make the radical leaps necessary to keep up with exponentially growing complexity.

They can't do that if they're borne down by the drudgery of mundane, repetitive tasks. No, the only way to free up experienced people is to not have them fighting fires, which means devolving technically precise but routine tasks to more junior staff. Developing team collaboration abilities are a key part of this, as are procedures.

You can follow this path by beginning with massive changes, events so large that you have no choice but to follow a plan. But as the process is applied to more routine activities you begin to

build up a library that helps you transition knowledge to more junior people. Redundancy is always thought of as a good thing with systems, but a word of horror when used in conjunction with staff. But the two senses of the word are different. It's not about putting people out of work - it's about empowering them to handle change, to pass on what they have learned, and to earn the freedom to be able to take that vacation without leaving your teammates in the lurch.

Procedures can be a breakthrough technique – as long as they are not used for bureaucracy – because they enable capture of both lessons learned and change history. They help senior systems administrators concentrate on making a significant impact, and they help junior staff gain confidence and experience. All together, that gives you and your team what you need to survive in a chaotic and ever changing world.

References

Notes to text

1 From “3 A.M. Eternal” by KLF - one of the first hits of the electronic music age. [4]

2 quotes in this section from Bruce Tate in [12], p 49. Further details are available online at [19].

3 Compliments to Steve Landers of Digital Smarties Pty Ltd, Western Australia who first pointed this out me. He recommends [11], a recent book of essays on the topic.

4 the doctrine that ultimately developed is called “Maneuver Warfare”. See [2],[5].

5 “Hyperkenetic present” is used by publisher Manning in describing the cover art of their books. They note that the great diversity of regional dress of even a mere 200 years ago has largely been lost, but perhaps we have gained more interesting, varied, and enriched personal lives. See colophon to [12]and [18].

6 Adapted from [8], p 50-51.

7 The worldly reader may wonder at the use of the US Army as a positive example at a time when American forces are struggling the field. That the US government did such a poor job of planning for the aftermath of their campaign in Iraq is nothing short of obscene negligence. The military, however, did its job – the actual invasion of Iraq was swift, decisive, and almost bloodless – a continued vindication of Maneuver Warfare as a planning and warfighting doctrine – the same doctrine that carried the day in Panama '89, Kuwait '91, Haiti '94, Bosnia '95 and Kosovo '99.

8 Professor Barry Bison, Chair, J. Herbert Smith Technology, Management and Entrepreneurship Center, University of New Brunswick, Fredericton, Canada, paraphrasing Kaplan and Norton [7],[13].

9 Jim Lovell, [6].

10 Gene Kranz in [10], p 249.

11 Kranz, [10], p 259.

12 Kranz, [10], p 318.

13 Kranz, [10], p 266 and 393.

14 Field Marshal Erwin Rommel, as quoted by von Mellenthin in [3]. His book is an excellent study of the imperative to give flexibility to your subordinates while binding their actions into a grand and cohesive whole.

15 widely attributed to Donald Knuth, although others in the computing pantheon seem to have presented the same idea, notably Sir Tony Hoare.

Publications

[1] Eliyahu M. Goldratt and Jeff Cox, *The Goal* (North River Press, Great Barrington, MA, 1984).

[2] William S. Lind, *Maneuver Warfare Handbook* (Westview Press, 1985).

[3] F. W. von Mellenthin, *Panzer Battles : A Study of the Employment of Armor in the Second World War* (Ballantine Books; Reissue edition 1985).

[4] KLF, "3 AM Eternal", *The White Room* (Arista, 1992).

[5] Robert Leonhard, *Art of Maneuver: Maneuver Warfare Theory and Airland Battle* (Presidio Press, Reprint edition 1994).

[6] Jim Lovell and Jeffery Kluger, *Lost Moon* (Houghton Mifflin Co, 1994).

[7] Robert S. Kaplan and David P. Norton, *The Balanced Scorecard*, (Harvard Business School Press, 1996).

[8] Gordon R. Sullivan and Michael V. Harper, *Hope is not a Method* (Broadway Books, New York, 1996).

[9] Steve Traugott and Joel Huddleston , "Bootstrapping and Infrastructure", *Proceedings of the 12th Systems Administration Conference (LISA '98)* (Boston: USENIX, 1998). pp181-196. See http://www.infrastructures.org/ for an updated version of this paper.

[10] Gene Kranz, *Failure Is Not an Option* (Berkley, New York, 2000).

[11] László Böszörményi, Jürg Gutknecht and Gustav Pomberger, *The School of Niklaus Wirth: The Art of Simplicity* (Morgan-Kaufmann, 2000).

[12] Bruce Tate, *Bitter Java* (Manning, Greenwich, CT, 2001). pp44-49.

[13] Robert S. Kaplan and David P. Norton, *The Strategy Focused Organization*. Harvard Business Press, Cambridge, 2001.

[14] "Centralized Application Systems Administration Support Procedures for the Defense Civilian Personnel Data System (DCPDS)", Department of Defense Civilian Personnel Management Service, 2002. http://www.cpms.osd.mil/vmo/documents/CentSysAdmProceduresOct2502.pdf

[15] Megan Santosus, "Here in body only" *CIO Magazine* (IDG Australia, September 2003).

[16] Albert Endres and Dieter Rombach, *A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories* (Pearson Addison Weselley, 2003).

[17] Robert S. Kaplan and David P. Norton, *Strategy Maps* (Harvard Business Press, Cambridge, 2004).

[18] "About our covers" (Manning). http://www.manning.com/about/covers.html

[19] Don Wells, et al: *Extreme Programming A Gentle Introduction*. http://www.extremeprogramming.org/

Software

[20] Nagios (formerly NetSaint) http://www.nagios.org/

[21] Orca, http://www.orcaware.com/ which is built upon Toby Oetiker's Round Robin Database and graphing tool, http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/