Operations Report Card
IN719 Systems Administration

Introduction

The Operations Report Card at `http://opsreportcard.com` is a list of 32 yes/no questions that help gauge the strength of an operations team's practices. Basically, the more "yes" answers, the better you're doing. In this paper we'll use the report card to help guide and set the standard for our work. ItâĂŹs not expected that you'll be able to answer every question "yes" by the end of the semester (and some don't apply to us), but you should be able to do so for many of the questions.

1 Public Facing Practices

1.1 Are user requests tracked via a ticket system?

This is so basic it pains me that I have to explain it.

Humans can't remember as well as computers. Expecting sysadmins to remember all user requests is the direct route to dropping requests.

Keeping requests in a database improves sharing within the team. It prevents two people from working on the same issue at the same time. It enables sysadmins to divide work amongst themselves. It enables passing a task from one person to another without losing history. It lets a sysadmin back-fill for one that is out, unavailable or on vacation.

It enables better time management. Every interruption a sysadmin receives sets his or her work back 7 minutes. A ticket system prevents interruptions from users that want to make requests or ask for status of requests. It lets sysadmins prioritize their work instead of responding to the loudest complainer.

It lets managers be better managers. Stalled requests become visible so that managers can intervene. It reveals trends and frequent requests so that they may be eliminated via new automation or process improvements. Micro-managers can get the information they want via software rather than bothering their sysadmins. It replaces "user whining" with evidence-based discussions: If a request really has taken too long the manager can properly address the issue; if the user only perceives the issue is taking a long time the evidence will shut them down. If they claim "it's been a problem for months but I only opened the ticket yesterday" then the manager has an... opportunity to educate the user about the non-existence of time-travel, mind-reading, and other supernatural powers.

It helps you identify systemic problems. I once had a boss query the ticket system and discover that 3 out of our 1,000 users opened 10% of all tickets. A little investigation and we were able to solve the fundamental issues causing this.

It helps users help themselves. Frequently when a user writes down what their problem is they realize what the solution is and no ticket is opened. When this doesn't happen, it helps them think through the problem so they can communicate it more clearly, which makes the actual transaction more efficient.

I spent the 1990s being the “radical” encouraging people to set up ticket systems. I spend the 200xs pleased to see it become accepted practice. We are well into the third decade. If you don’t have a way to track user requests at this point shame on you.

1.2 Are “the 3 empowering policies” defined and published?

There are three public-facing policies you must have if a sysadmin team is going to be able to get any work done. This is as much about serving customers as it is enabling team efficiency.

If you are a manager that feels your team has bad time management skills, maybe it is your fault for not having or not enforcing these policies:

- The acceptable methods for users to request help.
- The definition of “an emergency”.
- The scope of service: Who, what and where.

One document can explain all three things in less than a page. This should be made available on the department’s website or posters on the wall so that it is clearly communicated. This policy must also be backed up by management. That means they are willing to tell a user “no” when they ask for an exception. The exception process should not be a speed-bump, it should be a solid wall.

How do users get help?

An official protocol for how users are to request help enables all the benefits of the ticket system mentioned in the previous section. Without it all those benefits evaporate as users will go directly to the sysadmins who will, trying to be helpful, become interrupt-driven and ineffective.

A sysadmin must have the ability to tell users to go away when the user is not following the protocol. Without the ability to point to this policy sysadmins will either work on low priority, squeaky wheel tasks all day long, or each sysadmin will apply a different policy making the team look inconsistent, or sysadmins will communicate their frustration in unhealthy ways. Specifically, ways that are unhealthy for the user.

What is an emergency?

An official definition of an emergency enables a sysadmin to set priorities. Without this everything becomes an emergency and sysadmins become interrupt-driven and ineffective.

The policy is one way management communicates priorities to sysadmins. Otherwise sysadmins will guess and be wrong and be unfairly punished for their incorrect guesses; managers will be confounded by the “disconnect”; and users will see inconsistencies and assume favoritism, neglect, and incompetence.

This policy sets users’ expectations. Those that call everything an emergency can be corrected of their illusion.

Every organization should have a definition of an emergency or a “code red”. A newspaper’s code red is anything preventing tomorrow’s edition from being printed and loaded onto the 4am trucks. A factory’s code red is anything stalling the assembly line. A payment service’s code red is anything that is stopping the payment pipeline. Educational technology teams know that a class can’t simply be rescheduled therefore an emergency is anything preventing the proper delivery of a lesson (possibly only if the technology center was warned ahead of time). A university defines a code red as anything preventing grant proposals from being submitted in time.

A “code yellow” is anything that, if left unattended, would lead to a “code red”. For example, the payment pipeline might be functioning but the capacity forecasting sub-system is down. It is risky to take on new customers without being able to properly forecast capacity. The last estimate indicated about 2 weeks of spare capacity. Risk of a melt-down increases daily until the code yellow is resolved.

Anything else is “routine”. Fancy sites may divide routine requests into high, medium and low priorities; new service creation, provisioning of existing services, and so on. But if you have none of that, start with defining what constitutes an emergency.

What is supported?

An official definition of what is supported enables sysadmins to say “no”. It should define when, where, who, and what is supported. Do you provide support after 5pm? On weekends? Do you provide desk-side visits? Home visits? Do you support anyone off the street or just people in your division? What software and hardware are supported? Is there a support life-cycle or once something is supported are you fated to support it forever? Are new technologies supported automatically or only after an official request and an official positive reply?

Without the ability to say “no”, sysadmins will support everything. An eager, helpful, sysadmin will spend countless hours trying to get an unsupportable video card to work when it would have been cheaper to have gifted him or her a supported card out of your own budget. A sysadmin, assumed lost or dead, will magically reappear having spent the day at a user’s house fixing their Internet connection. Alternatively a curmudgeonly sysadmin will tell people something isn’t supported just because they’re busy.

1.3 Does the team record monthly metrics?

You need to be data-driven when you make decisions or sway upper levels of management.

The best way to develop your metrics is to create one metric per sentence or clause of your charter.

Example: The charter for a PC deployment team might be: To provide a high-end, standardized, computer to each employee starting their first day, refreshed on a 3-year cycle, at industry leading operational cost. The metrics might be: Number of weeks since the standard configuration was updated, cost of the current standard configuration, number of new employees this month, capital expenditures, operational expenditures. How many days new employees waited for their new machine (“buckets” for prior to arrival, first day, second day, 3rd, 4th 5th and more than 5th). Age of fleet (“buckets” for <1 year old, 1-2 years old, 2-3 years old, older than 3 years). Count of deployed machines that deviated from the standard.

If you don’t have a charter, talk with your manager about writing one. Alternatively here’s some simple “starter metrics” you can adopt today:

- How many sysadmins do we have?
- How many users do we provide service to?
- How many machines do we manage?
- How much total disk space? RAM? CPU cores?
- How many “open” tickets are in our ticket system right now?
- How many new tickets were created since last month?
- Who (or what department) opened the most tickets this month?
- What was the tickets per sysadmin average last month?
- Pick 4-5 important SLAs and record how close you were to meeting them.
- How much Internet bandwidth was consumed last month?

Record these on the first of every month. Put them in a spreadsheet. You'll be able to use this at budget time or during presentations when you want to explain what your group does.

That's it. Really. Recording those once a month is the difference between starting a presentation with a simple graph showing the rate growth in the number of machines on your network versus saying, "Hi, I'm Joe and we manage... umm... a lot of machines." At budget time being able to answer these basic questions is the foundation for other questions such as "How much does the average ticket cost?" (last year's budget total / last year's total ticket quantity). If we had 100 users, how much new disk will we need? (total disk space / number of users * 100).

Ultimately collecting these metrics should be automated. Until then, generate email to yourself on the first of each month with a reminder to do it manually.

2 Modern Team Practices

2.1 Do you have a "policy and procedure" wiki?

Your team needs a wiki. On it you can document all your policies (what should be done) and procedures (how it is done).

Automation is great but before you can automate something you must be able to do it manually. Documenting the manual process for something is a precondition to automation. In the meantime it enables consistent operations across a team and it gains you the ability to delegate. If it is documented, someone other than you can do it.

The table of contents for this wiki should include common, routine tasks. A good place to start is the add/change/delete procedures that anyone on the team should be able to do, and the tasks you dislike doing and would delegate to an assistant if you had one.

Procedure list:

- When a new employee starts.
- When an employee leaves the company.
- When an employee is terminated.
- When a new machine is installed.
- When a machine is decommissioned.
- How to add/delete a person to the VPN service.
- How to change a disk in the RAID system.
- How to change the root password on all machines.

There are three categories here: Things that you want to be consistent, things that you do infrequently and don't want to have to spend time re-remembering the procedure, things you do when panicking and don't want to have to think on your feet.

These things can all be documented with simple "step-by-step" checklists.

Once documented, anyone on the team can do them. It also creates your training program for new employees. It can also be used to write the job description of that assistant you want the company to hire for you to do all your work.

Even if you aren't on a team, or there are tasks that only you do, documenting has benefits. You have to think less when you do the task. Just like the adage that "we automate because we are lazy", it is also true that "we document because we are impatient".

Many sysadamins dislike writing documentation but writing a “step-by-step” checklist isn’t that bad. Keeping it on a wiki is important: anyone can correct it and anyone can improve it.

For any task you might want to have separate “policy” and “procedure” documents. Policy is what management defines: All new users will receive a wireless mouse. Procedure is how the tasks get done: The wireless mouses are stored in the 3rd bin; charge it, and test it with the following steps, etc. Policy are only changed with management approval. Procedures are changed by the technicians, with change notifications sent to the author or other authority.

2.2 Do you have a password safe?

This shows you have a mature way to manage passwords.

There are many excellent software-based password “vaults” systems. Though an envelope in an actual locked box is often good enough.

The problem is often verification. How do you know that an evil co-worker isn’t putting the wrong password in those envelopes? If you are that paranoid have a different person verify any new passwords.

2.3 Is your team’s code kept in a source code control system?

> When installing a new machine is an API call, we’re all programmers now. -Limoncelli, on cloud computing, devops, and the importance of developer skills in system administration.

We’re all programmers now. Programmers use source code control.

Things to put in your repository: Your scripts, programs, configuration files, documentation, and just about anything. If you aren’t sure, the answer is “yes”.

Keeping your configuration files in source code control starts out feeling like a luxury and ends up becoming a lifesaver.

Anything is better than nothing. Use what your developers use. No developers? Learn Git, Mercurial, or even Subversion. Desperate for a quick way to save configuration files history? `http://www.nightcoder.com/code/xed` (It is a wrapper that calls $EDITOR.)

2.4 Does your team use a bug-tracking system for their own code?

Bug-tracking systems are different than ticket systems. If you have only occasional bugs (maybe your group doesn’t write a lot of code) then filing help tickets for yourself is sufficient.

However if your team is serious about writing code, start a separate bug-tracking system. Bug-tracking systems have a different workflow than request ticket systems. Ticket systems are a communication tool between you and your users. Bug-tracking systems track the bug lifecycle (report, verify, assign, fix, close, verify).

2.5 In your bugs/tickets, does stability have a higher priority than new features?

Adding new features is more fun than fixing bugs. Sadly we can’t be fun here.

Mature teams prioritize bugs this way:

- security (highest)
- stability

- bugs
- performance
- new features (lowest)

You have to fix stability before you add new features. Security issues are high priority stability issues.

One of the principles promoted by Mark Burgess is "seek stability, then new features". Some changes we make add features while others improve stability. The order should be feature, stability, feature, stability, feature, stability. Not feature, feature, feature, OMG!, stability, stability, stability. Make things stable before proceeding to the next awesome feature.

Doctors understand this. In a hospital emergency room a patient is stabilized first. You don't help someone recover from the flu if they are bleeding to death.

The priority of "performance bugs" is up for debate. In some places performance is the same as stability.

2.6 Does your team write "design docs?"

Good sysadmin teams "think before they do." On a larger team it is important to communicate what you are about to do, or what you have done.

A design doc is a standardized format for proposing new things or describing current things. It should be short, 1-2 pages, but can be very long when the need arises.

Create a template and use it all over the place. The section headings might include: Overview, Goals, Non-Goals, Background, Proposed Solution, Alternatives Considered, Security, Disaster Recovery, Cost.

This format can be used to write a 20-page plan for how to restructure your network when you want to get buy-in from many people. It can be a 5-page document of how a prototype was built so everyone can see your results and give feedback before you build the real thing. It can be used for a half-page memo that explains the names you plan on using for a new directory tree on the file server (in which case, you probably don't need most of the headings). Use it to document a system after it was built for use as a reference by others on your team. Heck, use it to describe the team cookout you are planning.

The point is that your team has a mechanism for thinking before doing, a way to communicate plans beyond talking in hallways and chat-room, and a system that leaves behind artifacts that others can use to understand how or why something was done.

This format works when seeking feedback whether you want serious critiques or just "warn me if this will conflict with something you are about to do".

Your design doc format might have more headings or fewer, some may be optional, others may be required. Be flexible. A small project should only require a few headings. A huge project should require more of the headings. Be flexible. Have a "short form" and "long form". Having a standard template that people can use as a starting point avoids the "blank page syndrome".

2.7 Do you have a "post-mortem" process?

After a failure do you write up what happened so you can learn from it or do you just hope nobody notices and that it will all go away?

A good post-mortem (PM) includes a timeline of what happened, who was affected, what was done to fix it, how was business affected, and a list of proposed solutions to prevent this problem from happening again. Each proposal should be filed as a bug or ticket so they can be tracked to completion.

Doing PMs consistently builds a more stable environment. After each outage come up with at least one preventative measure. Can your monitoring system detect the situation so you know about it before users do? Can you detect precursors to the problem? Often systems have a way to run a battery of tests on new configurations before they are adopted ("pre-submit scripts" in source code repositories, for example). Are there tests you can add that will detect the typo that created the outage?

A post-mortem is not about blaming and shaming. In a good sysadmin culture you are comfortable with putting your name in the "what went wrong" section. You are taking a leadership role by educating people so they don't make the same mistake.

If your management uses PMs to find who to punish, they don't understand that operations isn't about doing things perfectly; it is about doing things better and better every day. Any manager that fires a person because of a non-malicious outage is going to run their company into the ground.

The PM should be published for all to see. You may be embarrassed and concerned that you are "airing your team's dirty laundry" but the truth is that if you consistently do this your users will respect you more. Transparency breeds trust.

Of course, to really develop confidence all those bugs and tickets filed as a result need to actually get worked on.

3 Operational Practices

3.1 Does each service have an OpsDoc?

Your DNS server dies. You rebuild it because you know how. Cool, right? You need to compile the newest version of BIND and install it, and you do it because you know how, right? When the monitoring system reports that error that happens now and then, you know how to fix it, right? You know how to do all this. Why write anything down?

Herer's why:

Will you remember how to do these things 6 months from now? I find myself having to re-invent a process from scratch if I haven't done it in a few months (or sometimes just a few days!). Not only do I re-invent the process, I repeat all my old mistakes and learn from them again. What a waste of time.

Will you remember how to do these things when the pressure is on? My memory works worse during an emergency.

What about when you aren't around? How can you take a relaxing vacation if you feel burdened? You can't complain to be over-worked and unable to share your work with others if you haven't created a way to share the workload.

What about new people on the team? Should they learn how to do these things by watching you do it or can they learn on their own? If they can learn on their own and only bother you when they get stuck it saves you time and makes you look less like the information hording curmudgeon that you don't want to be. In fact, it makes people feel welcome and included if their new team has these kind of tasks documented.

How can your manager promote you or put you on a new and more interesting project if you are the only person with certain knowledge?

Each service should have certain things documented. If each service documents them the same way, people get used to it and can find what they need easier. I make a sub-wiki (or a mini-web site, or a Google Sites "Site") for each service:

Each of these has the same 7 tabs: (some may be blank)

1. Overview: Overview of the service: what is it, why do we have it, who are the primary contacts, how to report bugs, links to design docs and other relevant information.

2. Build: How to build the software that makes the service. Where to download it from, where the source code repository is, steps for building and making a package or other distribution mechanisms. If it is software that you modify in any way (open source project you contribute to or a local project) include instructions for how a new developer gets started. Ideally the end result is a package that can be copied to other machines for installation.
3. Deploy: How to deploy the software. How to build a server from scratch: RAM/disk requirements, OS version and configuration, what packages to install, and so on. If this is automated with a configuration management tool like cfengine/puppet/chef (and it should be), then say so.
4. Common Tasks: Step-by-step instructions for common things like provisioning (add/change/delete), common problems and their solutions, and so on.
5. Pager Playbook: A list of every alert your monitoring system may generate for this service and a step-by-step "what do to when..." for each of them.
6. DR: Disaster Recovery Plans and procedure. If a service machine died how would you fail-over to the hot/cold spare?
7. SLA: Service Level Agreement. The (social or real) contract you make with your customers. Typically things like Uptime Goal (how many 9s), RPO (Recovery Point Objective) and RTO (Recovery Time Objective).

If this is something being developed in-house, the 8th tab would be information for the team: how to set up a development environment, how to do integration testing, how to do release engineering, and other tips that developers will need. For example one project I'm on has a page that describes the exact steps for adding a new RPC to the system.

Be a hero and create the template for the rest of your team to use. Document a basic service like DNS to get started. Then do this for a bigger service. Create the skeleton so others can use it as a template and just fill in the missing pieces. Get in the habit of starting a new opsdoc any time you begin a new project.

3.2 Does each service have appropriate monitoring?

> It isn't a service if it isn't monitored. If there is no monitoring then you're just running software.
> -Limoncelli

The monitoring should be based on the SLA from the OpsDoc. If you don't have an SLA, simple "up/down" alerting is the minimum.

Don't forget to update the Pager Playbook.

3.3 Do you have a pager rotation schedule?

Do you have a pager rotation schedule or are you a sucker that is simply on-call forever?

An on-call rotation schedule documents who is "carrying the pager" (or responsible for alerts and emergencies) at which times.

You might literally "pass the pager", handing it to the next person periodically, or everyone might have their own pager and your monitoring system consults a schedule to determine who to page. It is best to have a generic email address that goes to the current person so that customers don't need to know the schedule.

A rotation schedule can be simple or complex. 1 week out of n (for a team of n people) makes sense if there are few alerts. For more complex situations splitting the day into three 8-hour shifts makes sense. "Follow the sun" support usually schedules those 8-hour shifts such that a global team always has a shift during daylight hours. You might take a week of 8-hour shifts each n weeks if your team has 3n people. The variations are endless.

This schedule serves many people: You, your customers, management and HR.

It serves you well because it lets you plan for a life outside of work. I put the highest priority on having a good work-life balance. If you don't have a good work-life balance, and you don't have a rotation schedule, physician heal thy self.

The rotation improves service to customers because it takes the "chaotic panic of trying to find a sysadmin" and makes it easy and predictable.

It serves management because it gives them confidence that the next emergency won't happen while everyone is "away".

It serves HR since, of course, your company gives compensation time or pay as required by law. If your schedule is in machine-readable format, a simple script can read it to generate reports for the payroll department.

If you think you don't have a schedule then it is "24x7x365" and you are a sucker. (But that doesn't mean you can answer "yes" for this question.)

3.4 Do you have separate development, QA, and production systems?

Developers do their work on their development servers. When they think it is done packages are built and installed on the QA system. If QA and UAT (User Acceptance Testing) approves, the same packages are used to install the software on the production systems.

This is Sysadmin 101, right?

Then why do I constantly meet sysadmins whose management won't let them do this? If your management says "it costs too much to have a second machine" they're beyond hope. QA isn't expensive. You know what is expensive? Downtime.

Experimental changes on the live server isn't just bad, in SOX environments it is illegal. Letting developers develop on the live servers is right out!

The QA system need not be as expensive as their live counterpart. They don't have to be as powerful as the live system, they can have less RAM and disk and CPU horsepower. They can be virtual machines sharing one big physical machine.

Obviously if scaling and response time are important it is more likely you'll need a QA system that more closely resembles the live system.

3.5 Do roll-outs to many machines have a "canary process?"

Suppose you have to roll out a change to 500 machines. Maybe it is a new kernel. Maybe it is just a small bug-fix.

Do you roll it out to all 500? No. You roll it out to a small number of machines and test to see if there are problems. No problems? Roll out to more machines. Then more and more until you are done.

These early machines are called "canaries".

> The classic example of animals serving as sentinels is the canary in the coal mine. Well into the 20th century, coal miners in the United Kingdom and the United States brought canaries into coal mines as an early-warning signal for toxic gases including methane and carbon monoxide. The birds, being more sensitive, would become sick before the miners, who would then have a chance to escape or put on protective respirators. Source: Wikipedia

Here are some canary techniques:

One, Some, Many: Do one machine (maybe your own desktop), do some machines (maybe your co-workers), do many machines (larger and larger groups until done.) Any single failure means you stop the upgrade, roll back the change, and don't continue until the problem is fixed.

Cluster Canary: Upgrade 1 machine, then 1% of all machines, then 1 machine per second until all machines are done. (Typical at Google and sites with large clusters)

This procedure can be done manually but if you use a configuration management system, the ability to do canaries should be "baked in" to the system.

4 Fleet Management Processes

4.1 Is there a database of all machines?

Every site should know what machines it has. The database should store at least some basic attributes: OS, RAM, disk size, IP address, owner/funder, who to notify about maintenance, and so on.

Having a database of all machines enables automation across all your machines. Being able to run a command on precisely the machines with a certain configuration is key to many common procedures.

This data should be automatically collected, though a very small site can make due with a spreadsheet or wiki page.

Having an inventory like this lets you make decisions based on data and helps you prevent problems.

I know a small university in New Jersey that could have prevented a major failure if they had better inventory: They tried to upgrade all of its PCs to the latest version of Microsoft Office. The executive council was excited that there would finally be a day when version incompatibilities didn't make every interaction an exercise in frustration. Plus look at all these new features! Oh, how enthusiasm turned into resentment as the project collapsed. It turned out that one third of the machines on campus didn't have enough RAM or disk space. Random segments of the university couldn't get any work done due to botched upgrades. The executive council was not only upset but is now very risk adverse. It will be a long time before any new upgrades will happen. All of this would have been prevented if a good asset management system was in place. A simple query would have produced a list of machines that needed upgrades. Budgeting and work estimates could have been provided as part of the upgrade program.

4.2 Is OS installation automated?

Automated OS installations are faster, more consistent, and let the users do one more task so you don't have to.

If OS installation is automated then all machines start out the same. Fighting entropy is difficult enough. If each machine is hand-crafted, it is impossible.

If you install the OS manually, you are wasting your time twice: Once when doing the installation and again every time you debug an issue that would have been prevented by having consistently configured machines.

If two people install OSs manually, half are wrong but you don't know which half. Both may claim they use the same procedure but I assure you they are not. Put each in a different room and have them write down their procedure. Now show each sysadmin the other person's list. There will be a fistfight.

Users see inconsistency as incompetence. If new machines always arrive with a setting that isn't to their liking they know how to change that setting and are happy. If half the time that setting is one way and half the time it is another way, they lose confidence in the system administrators. What bozos are installing this stuff?

If you can re-install the OS automatically, so can the users. Now you have one less thing to do. Automation that saves you time is great. Automation that lets other people do a task is even better.

Not being able to easily wipe and reload a machine is a security issue. A machine should be wiped and reloaded when a "hand me down" computer moves from one user to another. If this process isn't "friction free" there is temptation to "save time" by not doing it.

4.3 Is there a database of all machines?

Every site should know what machines it has. The database should store at least some basic attributes: OS, RAM, disk size, IP address, owner/funder, who to notify about maintenance, and so on.

Having a database of all machines enables automation across all your machines. Being able to run a command on precisely the machines with a certain configuration is key to many common procedures.

This data should be automatically collected, though a very small site can make due with a spreadsheet or wiki page.

Having an inventory like this lets you make decisions based on data and helps you prevent problems.

I know a small university in New Jersey that could have prevented a major failure if they had better inventory: They tried to upgrade all of its PCs to the latest version of Microsoft Office. The executive council was excited that there would finally be a day when version incompatibilities didn't make every interaction an exercise in frustration. Plus look at all these new features! Oh, how enthusiasm turned into resentment as the project collapsed. It turned out that one third of the machines on campus didn't have enough RAM or disk space. Random segments of the university couldn't get any work done due to botched upgrades. The executive council was not only upset but is now very risk adverse. It will be a long time before any new upgrades will happen. All of this would have been prevented if a good asset management system was in place. A simple query would have produced a list of machines that needed upgrades. Budgeting and work estimates could have been provided as part of the upgrade program.

4.4 Is OS installation automated?

Automated OS installations are faster, more consistent, and let the users do one more task so you don't have to.

If OS installation is automated then all machines start out the same. Fighting entropy is difficult enough. If each machine is hand-crafted, it is impossible.

If you install the OS manually, you are wasting your time twice: Once when doing the installation and again every time you debug an issue that would have been prevented by having consistently configured machines.

If two people install OSs manually, half are wrong but you don't know which half. Both may claim they use the same procedure but I assure you they are not. Put each in a different room and have them write down their procedure. Now show each sysadmin the other person's list. There will be a fistfight.

Users see inconsistency as incompetence. If new machines always arrive with a setting that isn't to their liking they know how to change that setting and are happy. If half the time that setting is one way and half the time it is another way, they lose confidence in the system administrators. What bozos are installing this stuff?

If you can re-install the OS automatically, so can the users. Now you have one less thing to do. Automation that saves you time is great. Automation that lets other people do a task is even better.

Not being able to easily wipe and reload a machine is a security issue. A machine should be wiped and reloaded when a “hand me down” computer moves from one user to another. If this process isn’t “friction free” there is temptation to “save time” by not doing it.

4.5 Can you automatically patch software across your entire fleet?

If OS installation is automated then all machines start out the same. If patching is automated then all machines stay current. Consistency is a good thing.

Security updates are very important because the reliability of your systems requires them. Non-security related updates are important because the reliability of your system requires them and because it brings new features to your customers. Withholding new patches is like a parent withholding love. Who raised you?

Application patching is just as critical as patching OSs. Users don’t make the distinction between “OS” and “application”, especially if an application is installed widely. The bad guys that write malware don’t make a distinction either.

I wish banks had to publish their patching process so I could decide where to keep my money.

The alternative to automation is visiting each machine one at a time. This annoys users, wastes their time, and it a stupid use of your time. With the proliferation of laptops it isn’t even reasonable to think you can visit every machine.

When possible, updates should happen silently. If they require a reboot or other interruptions, users should have the ability to delay the update. However, there should be a limit; maybe 2 weeks. However the deadline should be adjustable so that emergency security fixes can happen sooner.

4.6 Do you have a PC refresh policy?

If you don’t have a policy about when PC will be replaced, they’ll never be replaced.

[By “PC” I mean the laptop and desktops that people use, not the servers.]

In the server room there is usually more thought about when each device gets replaced. Your PC environment generally needs some kind of repeatable, cyclic, process so that it stays fresh. Without it things get old and unsupportable, or people get upgrades as a status symbol and it becomes political. With a good policy things get better and more cost effective.

A certain fraction of your fleet should be old; that’s just economical. However, extremely old machines are more expensive to maintain than to replace. It is a waste of your time to produce a work-around so that new software works on underpowered machines. It is a waste of your users’ time to wait for a slow computer. It is bad time management and bad for productivity to have seriously old machines.

Companies often get into this situation. Sometimes they “save money” by not upgrading machines but it doesn’t save money to have employees with tools that don’t work well. Sometimes they just don’t realize that computers don’t last forever.

If you don’t have a policy, here’s a simple one you can start with: All computers are on a 3-year depreciation schedule. Every year the budget will include funds to replace 1/3rd of all machines. On the first day of each quarter enough machines will be ordered to replace the 9 percent oldest machines in the fleet.

CFOs like this because they like predictability. At one company the CFO was quite excited when I gave her control over which months the upgrades would happen. We agreed that 1/4 of the upgrades would happen each quarter; and she could pick which month that happened. She could even split it into individual monthly batches.

Instead of coming to the CFO to beg for new desktops now and then, it was a regular, scheduled activity. Less pain for everyone.

ProTip: At some companies servers are on a different depreciation schedule: they are designed to last longer and are on a 4-year depreciation schedule. On the other hand, their cost is amortized over all their users and therefore you can justify a 2-year schedule.

5 Disaster Preparation Practices

5.1 Can your servers keep operating even if 1 disk dies?

It used to be that if there was one broken component in a computer, you had an outage. In fact, one component failure equaled one outage. One disk dies and you spend the day replacing it and restoring data from the backup tapes. Too bad if you had hoped to get some work done, too bad if you planned to be at the company picnic that day. One failed disk ruins your whole day; just like nuclear war.

Today things are different. We build “survivable systems”. If a disk is part of a mirrored pair then one of those disks can fail and there is no outage. There is only an outage if it’s mirrored partner also breaks. Statistically that gives you hours, possibly days, to replace the broken disk before a user-visible outage would happen. Rushing to replace a disk is a better use of your time than spending a day restoring data from tape.

When you do this you have decoupled “component failure” from “outage”. Life is better.

RAID used to be expensive and rare. A luxury for the rich. Now it is common, inexpensive, and often free (when done in software). Did I say common? I meant mandatory. Spending a day restoring data off tape isn’t just a sign of bad planning, it’s bad time management. It is a waste of your time to spend the day consoling a distraught user who has lost years, months, or even just hours, of work. It isn’t heroic. It is bad system administration. Let’s not forget the bigger waste of time for your users, possibly hundreds of them, waiting for their data to be restored off a backup tape. Disk failures are not rare. Why did you build a system that assumed they are?

The MTBF of a typical server drive is 1.5 million hours. If you have 1,000 disks, expect a failure every 2 months. If you have 10,000 disks expect a failure every week. Are you really planning to spend an entire day restoring data from tape that often?

My rule of thumb is simple: For small servers the boot disk should be mirrrored and any disk with user data should be RAID1 or higher.

Boot disk: I recommend mirroring the boot disk of every server because it is usually impossible to rebuild from scratch. Server boot disks tend to accumulate “stuff”. Over the years many software packages may be installed. New drivers, patches and kludges may exist that aren’t documented. The configuration is more a history of the company than some well-planned design. In a perfect world none of this would be true. Every machine should be reproducible though an automated system. Alas, that is the goal but we aren’t there yet. The primary exception is clusters of homogenous machines like HPC environments or Google or Yahoo!. Alas, dear reader, I bet that isn’t your situation. In fact, I bet that even at Google and Yahoo! the Windows server that runs keycard system that lets people in and out of the buildings is exactly the worst-case scenario that needs a mirrored boot disk.

Yes, you could probably rebuild such a server in a day if you are lucky, but a RAID1 controler is less than that in salary if you work minimum wage.

User data: I recommend RAID1 or higher for user data just because it is so inexpensive that not doing it is embarrassing. By the way... you do know that RAID6 is the minimum for 2T disks and larger, right? It is professionally negligent to use RAID5 on such disks. RAID6 or RAID10 is the minimum; at least for now, but I digress.

The exceptions to all this is any place where the service can keep running if individual components die whether the redundency is at the disk level, the machine level, or the data center level. Also, data that can be reconstructed from scratch within the SLA. Here are some examples:

- The use of a fancy redundant file systems like the Google File System (GFS). GFS stores all data in at least 3 places. IBM's GPFS Native RAID (GNR) does something similar.
- "Scratch and temp space" where users know it could go away at any time.
- Video or other read-only data that could, if lost, be re-read from media.
- The data is a read-only copy of data found elsewhere. Though if you are replicating for speed, RAID5 might give you a performance boost because it uses more spindles.
- Disposable machines. For example, a static image web server or a DNS "secondary and cache"-only server that can be rebuilt quickly and automatically. If you have hundreds of them the savings from not buying RAID cards can be dramatic.

5.2 Is the network core N+1?

An outage for one person is a shame. An outage of many people is unacceptable. Just like redundant disks is now a minimum, duplicate network connectivity is, too.

Yes, it is still expected that there is one link from a workstation to the first switch, but after that everything should be N+1 redundant. At a minimum, all trunks are dual-homed. At best, any one uplink, any one card, or any one network router/switch/hub can die and packets still get through.

LANs are generally designed as follows: The laptops/desktops in an office plug into the wall jack. Those connect to "access switches" which have many ports. Those access switches have "trunks" that connect to a hierarchy of "core switches" that scoot packets to the right place, possibly to egresses (the connections to other buildings or the Internet).

My rule is simple: The core has got to be redundant. It used to be a pricey luxury for the rich. Now it is a minimum requirement.

If your site isn't configured that way, you are living in the days when computers were useful without a network.

Exception: Sites small enough that they don't have a core. Even then all trunks should be redundant and a "spares kit" should exist for each kind of hardware device.

5.3 Are your backups automated?

This question assumes you are doing backups. You are doing backups, right?

You need backups for 4 reasons:

1. Oops, I deleted a file.
2. Ooops, the hardware died.
3. Oh no, the building burned down.
4. Archives.

Each of these may require different backups methodologies.

Situation (1) is solved by snapshots in the short-term but not in the long term. Sometimes a file is deleted and needs to be restored much later. Simple snapshots will not help. RAID does not help in this situation. RAID is not a backup mechanism. If someone deletes a file by mistake, RAID will dutifully replicate that mistake to all mirrors. You will have a Redundant Array of Incorrect Data.

Situation (2) sounds like RAID will help, but remember that a double-disk failure can mean you've lost the entire RAID1 mirror or RAID5 set. RAID10 and RAID6 lose all data in a triple-disk failure. These things happen. You are one clumsy electrician away from having all disks blow up at once. Really.

Situation (3) is often called "disaster recovery". Off-site backups, whether on tape or disk, are your only hope there.

Situation (4) is often for compliance reasons. The technology to make the backup is often the same as Situation 3 but the retention time is usually different. If some other department is requiring these for compliance, they should pay for the media.

For any of these reasons the process must be automated. As the building burns down you don't want to have to inform management that the data is lost because "I was on vacation" or "I forgot".

5.4 Are your disaster recovery plans tested periodically?

The last section was a bit of a lie. There aren't 4 reasons to do backups. There are 4 reasons to do restores.

Nobody cares about backups. People only care about restores. If you can figure out how to do restores without needing to do backups first I will lobby the Nobel committee to create a prize for sysadmins just so that you can be the first to receive it.

You don't know if backups are valid until you test them. Faith-based backup systems are not good. Hope sustains us but it is not an IT "strategy".

A full test involves simulating a total failure and doing a 'full restore'.

You won't know the real amount of time a restore takes until you try it. Restores from tape often take 10x longer than doing the backup. If you can do a full backup of your payroll server in 8 hours, then you have to be prepared to not cut paychecks for 80 hours in the event of a restore from scratch. That's more than 3 days.

If you are doing absolutely no tests then a little testing is better than nothing. Write a small script that randomly picks a server, then randomly picks a disk on that server, then randomly picks a file on that disk. The script should then create a ticket asking for that file to be restored (to a scratch location) as it existed 6 weeks ago. Have the script run automatically every week. This has a good chance of finding a server or disk that wasn't added to the backup schedule. Also, if you think doing these restores will be a lot of work for you, here's a secret: it won't use any of your time if your coworkers end up doing the ticket. Generate the ticket with enough random text that they don't know it is a drill.

To take this one step further, plan a "game day" where the disaster recovery plans are really put to the test. Pretend that certain people are dead and make sure the remaining people know how to fail-over services. Write scripts that document what tests will be performed. Either actually cause outages (disconnect the power or network cable) or play-act the scene: the "dead" person can proctor the test. ”Ok, now lets suppose you got paged with this message. Tell me the commands you type and the actions you take.“ Another method is to permit your CEO to walk into the data center and unplug any cable of his or her choosing.

5.5 Do machines in your data center have remote power / console access?

This needs little explanation. Remote consoles (IP-based KVM switches) are inexpensive; good servers have them built in. Remote power control isn't a luxury if the computer is more than a few miles away.

The exception to this rule is grid computing systems with hundreds or thousands of identical machines. If one fails another can take its place.

6 Security Practices

6.1 Do Desktops/laptops/servers run self-updating, silent, anti-malware software?

Viruses and malware are a fact of life. If you think bad things don't happen to good people then we all must be bad people. Every machine needs anti-malware software now.

Every malware attack means work for you: cleaning up a failed machine, recovering data, consoling users over the loss of their data. It is a waste of time for you, an inexcusable disruption to your users. It is bad time management.

Anti-malware software needs to periodically update itself. Some anti-malware software pops up a big window that says, "There's a new update! Would you like me to install it?" This is an important window because the software's logo appears in it. If users know the brand of anti-malware being used, they can recommend it to friends. It helps the vendor's stock price if everyone knows their name. It doesn't matter that 9 times out of 10 the user clicks "no". Software that silently updates itself with no animated "update" notification does not have this benefit. How irresponsible of the company to not benefit their shareholders.

If you aren't sure, I'm being sarcastic.

There are many anti-malware products. The one you install should silently update itself.

It is your job to enforce the security policy and downloading updates is part of that policy. Delegating that responsibility to a user is wrong and possibly unethical. You don't ask a pedestrian, "Should I press the brakes and not run you over?" and you don't ask a user for permission to be less secure. Yes, we used to have 300 baud modems and downloading a virus definition file would take 30 minutes. You weren't even born then. You can't use that excuse. If you were around back then (like I was) then you are old enough to know better.

For similar reasons the anti-malware software should not be easily disabled by the user. Users will disable it for the most preposterous reasons. It is common for users to disable it in an attempt to speed up their computer. I once had a user that disabled it because "it attracts viruses to my machine". You see, he explained, when it was enabled it kept popping up windows warning of viruses. When he disabled it there were no warnings. Yes, people with such a gross misunderstanding of cause and effect do exist.

Anti-malware software used to be "would be nice" but now it is a requirement. These are my personal rules:

Anti-malware scanners must run on all machines including any server that contains user-controlled data: home directories, "file shares", web site contents, FTP servers, and so on. Scanners must update automatically and silently. No user confirmation. There must be a mechanism that lets you detect it has been disabled. They should "check in" with a central server so you can see which machines are no longer being updated. Email must be scanned on the server, not on the client (or in addition to the client). Messages with malware must be dropped; messages with spam should be quarantined. You can't trust each individual machine to have the same, high-quality, up-to-date filter as you can maintain on your server. Stop the problem before it gets to the client.

6.2 Do you have a written security policy?

Looking at existing policies is a good way to get ideas. SANS has a library of samples:

`http://www.sans.org/security-resources/policies/`

It is critical to have a written security policy before you implement it.

6.3 Do you submit to periodic security audits?

This needs little explanation. If you aren't testing your security, you don't know how vulnerable you are.

6.4 Can a user's account be disabled on all systems in 1 hour?

This indicates a lot more about your team and the environment you run than just whether or not you can disable an account. It indicates the use of a global unified account system.

Having a single authentication database that all systems rely on is no longer a "would be nice". It is a "must have". If you think you don't need it until you are larger, you will find that you don't have time to install it when you are busy growing.

The best practice is to employ user account life-cycle management systems. With such a system user accounts are created, managed and controlled from pre-employment through life changes to termination and beyond. What if a user's name changes? What if someone rejoins the company? What if they rejoin the company and their name has changed? There are a lot of "edge cases" they must be able to handle.

6.5 Can you change all privileged (root) passwords in 1 hour?

This also indicates a lot more than what the question specifically asks. It indicates if your administrative access is well managed.

If you do not have this ability, create a checklist of everywhere it must be changed on a wiki page. Change the password globally by following the list, adding to it as you remember other devices. For obscure systems, document the exact command or process to change the password.

If you do have this ability, create a wiki page that documents how to activate the process (and then list all the exceptions that are still manual).