Best Practices in IT – Recommendations Based on Proven Experience

Originally an Internal Training Document

Created: January 2006 with revisions

Utilizing proven practices is key to efficiency, usability, extensibility, and asset protection in IT.

Contents

Preface

This document was originally created as a reference to help identify best practices that could / should be implemented in all professional services engagements. This is part of our overall Quality Management Plan. What is good for a consultant is also good for a customer, and the recommendations still apply. Everything described within this document has been put into practice by the author many times during his career and has proven to be invaluable time and time again

Why Consider Best Practices?

First and foremost, the answer is to learn from the mistakes of others. Best practices come from both the good and bad experiences of other organizations. They represent the knowledge gained from those experiences that have been proven in production environments, and provide the opportunity to improve operations at a minimal cost. And, they can help you avoid costly problems!

General Tips

1. Understand the environment, the various components involved, and the interaction between those components. Being able to quickly and accurately describe the environment to an outsider helps speed response times for requests and helps minimize errors due to an incomplete or incorrect understanding of your environment. It also provides the context necessary for someone to make recommendations on alternatives.
2. Understand the impact of downtime. Will lives be lost? What is the financial cost per hour or day? Are there regulatory or compliance issues to be concerned with? What is the PR (public relations) impact? For example, could you receive "bad press?" Could your competition use this against you?
3. Understand the impact of lost data. Can the data be reproduced by any other means (such as paper forms)? If yes, what is the time and cost of doing so? What is the impact of lost treatment data? Probably billing, but what about scheduling and a treatment plan? Could someone miss an important treatment or receive too many treatments? Would this expose your company to litigation risk? Would this be a violation of regulatory or compliance issues? (e.g., HIPAA or SOX) What is the impact of that risk?
4. Define Communication Procedures. What are the major categories of events that are anticipated for your environment (e.g., planned vs. unplanned downtime, crisis notification, routine "heads-up" type communications) and who needs to know that information? Is there an escalation procedure for critical issues? What needs to be documented in writing? What / where is the repository for written communication? Consistency is important.
5. Create an IT Risk Plan. Identify potential risks and their impact. Create a risk management plan for those that have a significant impact or are more likely to occur. Identify the triggers for executing that risk management plan and ensure that there is approval for implanting actions based on that plan. In a time of crisis this information is invaluable and relieves the burden of the decision making process on the person being asked to carry out the pre-defined plan. This could result in improved response time, reduced data loss, and better overall decisions.
6. Keep support agreements in place for key systems. Any system that is critical to production should either have a support agreement or a replacement system that can be swapped-in at a moments notice. We did work for one client who failed to do this. When the RAID controller on the Windows Server for their EDI system failed the vendor was unwilling to help without payment at the time of service. Managers at the client site were only willing to issue a purchase order. I paid for this with our corporate AMEX and we billed the client. The CIO later thanked me.

7. Focus on Staff Development. The people entrusted with supporting your production systems should have the ability and resources available to do so. Are they really familiar with the environment and tools that they need to use or support? Do they understand how various components interact (e.g., Ingres and Crystal Reports)? Do they know what to do and who to call when there are problems? Training can have a tremendous payoff when it is focused on areas that are most beneficial to your needs. Having employees document procedures and train others demonstrates their ability, provides an opportunity for them to learn more, and creates an opportunity to cross-train others in order to avoid dependence on specific individuals.

System Administration

1. There should be a representative test environment for production changes. This includes the installation of patches / hot fixes, operating system and software upgrades, tuning, configuration changes, etc. Sometimes even innocuous looking changes will have a negative impact. Being able to identify and resolve those types of issues proactively, before they become production problems is invaluable, as is having a stable test environment to validate changes and/or fixes. Please note that "representative" should really be representative from all perspectives. We once worked with a site running Windows 2000 Server using identical hardware for test and production. Everything seemed identical but the problem only presented itself on the production system. We finally identified the only difference between the systems – Windows Registry entries. The production system had about 25% more entries than the test system. This client rebuilt the production server during a weekend and resolved their problem.
2. Perform regular maintenance such as system reboots or restarting certain applications. Scheduled maintenance (weekly, monthly, or quarterly) is important. Planning for it and making others aware of it increases preparation all around and minimizes end-user frustration and problems due to unplanned events and outages.
3. Make certain that all systems are properly labeled, front and back. It is too easy to unplug something from the wrong machine when you are not absolutely certain about the system you are working on.
4. Plan on failure. Utilize redundancy whenever possible. Have spare parts when it makes sense. Evaluate support contracts to ensure that they are consistent with your business needs (e.g., night and weekend support, response time). Visually inspect the systems daily, looking for fault lights, no lights, and listening for unusual sounds.
5. Actively review available patches and drop support dates. Just because a patch is available doesn't mean that it needs to be applied, but someone should at least review the bugs it resolves to see if they might apply to your environment. Once a vendor publishes drop support dates it is important to begin considering upgrade plans. Too often organizations are forced to upgrade in a short period of time due to a new bug arising in unsupported versions of software.

6. Keep support information readily available. This includes contact numbers, site / license ID, serial numbers, etc. that the vendor will ask about when a problem ticket is opened with support. It is also important to know the minimum set of diagnostic information that vendor will require. Where possible the collection of this information should be automated so that it can be easily captured when any system problems occur. This might include log files, configuration files and output from diagnostic commands.

7. Validate that your backup & recovery strategy works as planned. The representative test system is a perfect place to do this. This validation should occur at least quarterly. Take this opportunity to familiarize members of staff with the recovery of systems they do not normally support so that recovery is not dependent on specific individuals. This will also help to validate the recovery plan as someone who is less familiar with the systems will have to follow the plan to the letter.

8. Use Antivirus Software in Windows environments. While this may seem to be common sense, many sites fail to have antivirus software installed, have not configured it (at all or only minimally), use old virus definitions, or fail to actively monitor the activity of that product. This is a big risk in a Microsoft Windows environment.

9. Minimize user activity on production servers. People should only login to a production server when they have something specific to do on that system. Each "user" (generally a support resource as opposed to an end user) should have their own non-privileged account where they can perform the majority of that work. Privileged, generic accounts (such as Administrator or Ingres) should only be used when absolutely necessary.

Security / Access Control

1. Each end-user should have their own unique account(s) to access production systems. This provides control and audit access and work performed by this person. Sharing accounts at any level (e.g., using the "produser" user account for all production users) is bad practice.
2. Password aging should be implemented and enforced. All accounts should require that the passwords be changed on a regular basis (typically every 1-3 months).
3. User accounts should be disabled immediately when an employee leaves.
4. Passwords should be changed immediately after an outsider (such as a consultant) who has been given those passwords leaves.
5. Production programs should never run from personal directories. We have seen several sites where production control went out of control because one or two developers were provided unlimited ad hoc access to production databases and allowed to run production jobs from their accounts.
6. Access to production data should be through tested and verified programs and scripts ONLY. Allowing ad hoc access to manipulate production data could cause data corruption, lead to invalid reporting of results and could be in violation of data protection & Auditory regulations (e.g. HIPAA and Sarbanes-Oxley). Change control process for data "fixes" should be in place, and they should track the reason for the change, who approved it, and the "before" and "after" copies of the data. This audit information should be retained in a common repository.

Database Administration (with an Ingres focus)

1. Perform routine maintenance. This includes table modification (usermod is a good tool), statistic generation (optimizedb), system catalog modification (sysmod), etc. Even though Ingres does not require a lot of day-to-day management it is still preferable to actively manage the environment instead of just reacting to problems after a few months.
2. Perform daily backups ("checkpoints" - ckpdb). A database checkpoint is the only supported means of database backup. Offline checkpoints are always preferable over online checkpoints but it is often difficult or impossible to do since they require exclusive access to the database being checkpointed. Online checkpoints are more complex and are therefore more prone to error. The way to mitigate the risk of recovery problems is to validate the backups via recovery of checkpoints in the test environment. This should be done on a regular basis – at least once per quarter.
3. Check the status of checkpoints daily. The "infodb" command will have an entry for checkpoints. A status of '1' means that the checkpoint was successful. Below is an example of a successful online checkpoint:

```
----Checkpoint History for Journal----------------------------------------------
Date                    Ckp_sequence  First_jnl  Last_jnl  valid  mode  ----------
-------------------------------------------------------------------------------------
Wed Jan 18 03:10:02 2006    582       583        583       1     ONLINE

----Checkpoint History for Dump------------------------------------------------
Date                    Ckp_sequence  First_dmp  Last_dmp  valid  mode --------
-------------------------------------------------------------------------------------
Wed Jan 18 03:10:02 2006    582       0          0         1     ONLINE
```

   This check should be automated and reported in real time via pager or email if a checkpoint fails, so that remedial action and a re-run can be performed as soon as possible.
4. Use journaling to capture data changes. Journaling records changes to data on a journaled table within a journaled database. This provides for the ability to have recovery up to the point of a failure (when used in conjunction with a valid checkpoint and the transaction log file) as well as the ability to view an audit trail by time, table, or Ingres user account. All tables should be journaled even if they are considered static. This ensures consistency for the entire database.
5. Review the Ingres Error Log (errlog.log) daily. The error log file is the first place that a DBA would check in the event of errors, but it is also a file that should be reviewed 2-3 times per day. This process can be automated, or if done manually should only take a few minutes. By doing so the DBA will start to recognize patterns that could identify opportunities for improvements (e.g., lock timeouts or deadlocks on a specific table or at a certain time), and will also be able to quickly identify anything that is new and different. This often allows them to proactively investigate an issue before it becomes a problem, making resolution faster and more foolproof. It is also a source of data collection for trend analysis.

6. Use a consistent and logical location naming scheme. For example, we use location names such as "ingsys" for the Ingres system location, "ingdata" for the Ingres database data locations, and "ingsort" for the sort/work areas. Whenever possible we match the OS filesystem name to the Ingres location name. This makes translation quick and easy. This standard should be consistently applied to all environments.

7. When using multiple data locations it is best to have all tables use all locations. This provides consistent data loading and load balancing. It also makes overall management of the Ingres installation and database(s) much easier.

8. Get a feeling for what is normal for your environment. Deviations from normal are generally not good. For example, if the disk utilization for the database locations sharply increases or decreases it could indicate a problem. If the CPU or memory (RAM) utilization changes significantly that could indicate a change or a problem. Knowing what the system "looks and feels like" when running normally makes it easier to diagnose abnormal behavior such as a performance problem.

9. Validate all patches / upgrades in a representative test environment before applying to production. Not all patches or upgrades are good. While there maybe improvements in some areas there might also be new problems introduced to the environment. Performance, behavior, syntax, etc. are all things that could change. It is very important to test functionality, performance, and stability before changing the production environment.

10. The owner of a production database should not be a real operating system account. Ingres will allow you to create an Ingres account that does not match an operating system account. Then a privileged Ingres account (such as "ingres") can create the database with that "fake" account as the owner using the "-u" flag on the "createdb" command. This makes it far more difficult to accidentally destroy or recover a production database.

11. Certain potentially dangerous Ingres operating system commands should be managed with 'wrapper' scripts / programs. The two commands that we typically rename are "destroydb" and "rollforwarddb". We then create a wrapper with the name of the original command. This logs actions and provides additional safeguards to protect against accidental use of these commands. It has saved a couple of clients from significant data loss and is highly recommended.

12. All secondary indexes should be created 'with persistence'. This helps avoid unintentional index loss after a table modify and does not increase overhead whatsoever within the table.

13. Key metadata should be collected on a regular basis. Collecting the following data on a regular basis and maintaining several months' worth of these collections could provide opportunities to recover data that would otherwise be extremely difficult or even impossible. At a minimum that data should include:

    a. The 'copy.in' file created by the 'copydb' command.

    b. The database statistics created by the 'statdump –o' command.

    c. The output from the 'infodb' command.

    d. The output from the 'ingprenv' command.

e. The output from the 'help table *;' and 'help index *;' commands within the SQL interface for the production database(s).

f. The output from 'select * from iitables;', 'select * from iifile_info;' and 'select * from iimulti_locations;' system catalogs from within the SQL interface for the production database(s).

g. A copy of the database configuration file (aaaaaaaa.cnf) for the production database(s) found in the primary Ingres data location directory.

14. Don't indiscriminately unload / destroy / reload a production database. There is always the chance of data loss, metadata is lost, and the ability to recover from previous checkpoints and journals is virtually lost. This should always be a last ditch approach to solving a problem.

15. Keep support information readily available. This includes contact numbers, site ID, serial numbers, etc. that the vendor will ask about when a problem ticket is opened with support.

Programming

1. Test for expected error conditions and have a general trap for unknown error conditions. What type of information will you need to investigate that problem further? That information should be logged somewhere. Can you identify the execution path? Often knowing what happened prior to the problem will help identify the root cause.
2. Understand ANSI transaction semantics and good transaction design. Remember that a transaction is a "logical unit of recoverable work." It is possible to separate a business transaction into several physical transactions and still maintain transactional integrity. Also, understand the impact of errors on the state of the transaction. For example, with Ingres a deadlock will rollback the entire transaction but a lock timeout will only rollback that statement. Not understanding this will likely result in data integrity problems (as will using things like "set autocommit on").

3. Include restart logic in a program whenever possible. While that is more work, it makes it easier to resume production processing and minimizes the chances if introducing data errors.

4. Design performance and concurrency into a system. Don't just expect a program to work fine. Plan for it, prove it, and then sit back and enjoy the fruits of your labor. It is far more difficult to back-in performance and concurrency improvements into a system than it is to do things right the first time through. This implies an understanding of performance and concurrency requirements and goals for the system!

5. Use consistent datatypes. Don't mix sizes and/or types (e.g., smallint with an integer). This can cause data errors (precision, truncation, and even loss due to incorrect conversion) and also performance errors if Ingres has to cast the values to make them consistent.

6. Be generous with comments. Detailed, unambiguous comments are as important as the source code itself from a maintenance and support perspective.

7. Use Source Code (Version) Control. Code should be checked out and checked back in on a regular basis (at least daily) to minimize the chance for loss. Anything important (forms, database procedures, rules, reports, etc.) should be stored as well. Version numbers should be assigned to each object and used for tracking releases and problems (in a manifest).

8. Don't leave an open transaction when waiting for user input. Concurrency problems are easily avoided by committing a transaction, presenting the data, and then reselecting the data to validate that it has not changed prior to an update or delete operation.

9. Build tracing / debug functions into the programs. Again, assume that there will be problems someday. Make it easy to turn on and gather statistics and debugging information in an easy and non-intrusive manner.

Quality Assurance

1. Validate that a new module or executable meets the stated requirements before it is released into production. This includes functionality, look & feel, performance, and concurrency. These things should not be assumed or left to chance.
2. Perform integration and regression testing for every release. What this really means is that the number of releases will be minimized and testing effort will increase, but to the end-user this will translate to consistent performance and few problems.

3. Never make changes to a production environment "on the fly." Those changes will almost certainly be made without the benefit of QA, and therefore are more likely to have a negative impact on that environment.

Production Control / Release Management

1. Production releases should be planned events – not surprises! Changes to the production environment should not be taken lightly. As such they should be planned with notification of the release being given to the user community well in advance. Immediately following the production release additional monitoring should be performed by the production support team.
2. A manifest of version numbers for each release should be maintained. This provides the means of reproducing or restoring a previous environment for any particular point in time. This is especially helpful in the event of problems with a new release.

3. Track important objects to identify changes. If there is a particular program executable that is critical to the application system then the date stamp and file size should be tracked. If either changes then notification should be sent out. We have automated processes to do this in a Unix environment and often catch unapproved changes to a production environment.

4. Changes to the production environment should never be made without management approval. Management seldom likes surprises, and never likes negative surprises. Make them aware of the specific change, the reason for the change, and the potential impact of the change. Get their approval before making such a change.

5. All changes should be logged. Detailed information about any and all changes (reason, before value, after value, post-change assessment) should be recorded and saved for future reference.

Project Management

1. Major "events" should be treated as projects. This means that the requirements should be well defined, a plan to achieve the stated goals created, and various supporting plans (risk management, communication, test, and implementation, fallback) created to support this effort. The level of detail and complexity will be dependent on the size and scope of the effort, but it should be a conscious decision either way.
2. Professional Project Management techniques should be utilized. For example, the Project Management Institutes' PMBOK (Project Management Book of Knowledge) defines many best practices and techniques for effective project management. These are things that will save money in the long run by helping maintain focus on the goals and constraints of the project. Please see http://www.comp-soln.com/PM_presentation.pdf and http://www.comp-soln.com/PM2_presentation.pdf for more information.
3. Understand the Business Objective of the Project. The scope and deliverables need to be in alignment with those business objects. Moreover, the Project Manager needs to create a "common vision of success" at the beginning of the project, and then continue to promote that vision throughout the project. Please see http://www.comp-soln.com/EVM_scope.pdf for our white paper on scope management.
4. Create Realistic Estimates. A work breakdown structure that defines tasks in easy to manage "chunks" (typically no less than 4 hours, and no more than two weeks) leads to better estimates. Poor estimates can lead to the perception of failure, even if everything else is being done well. Please see http://www.comp-soln.com/EVM_estimating.pdf for our white paper on estimating.
5. Create a Risk Management Plan. Projects seldom work out exactly as expected so proactively identifying risks, quantifying a risk score (probability * impact), looking at strategies to eliminate or mitigate the risk, looking at the cost of those efforts, and then defining a plan that has specific triggers and responses (and is approved by Management) is the best approach. Additionally, having a contingency reserve (budget) to address those risk events can avoid costly delays should a problem occur.
6. Create a Communication Plan. Effective communication that is timely, unambiguous, and specific will go a long way towards keeping a project moving forward and helping keep people accountable for their specific responsibilities.
7. Create a repository for information that is easily accessible. We use eRoom and Wiki-type tools (such as Basecamp) to provide easy access to project information by authorized individuals. This includes meeting minutes, old copies of project plans, status reports, risk management information, etc. There is no reason to "hide" this type of information, and later it will become a good source of information for the historical record of the project.

Production Support / Troubleshooting

1. Define acceptable Service Level Agreements (SLAs). What is an acceptable response time in the event of a problem? What is an acceptable level of downtime for an unplanned failure? What is an acceptable number of unplanned failures for a particular time frame (e.g., one or less per quarter)? Determine what is acceptable for your environment, perform a gap analysis to see what it will take to achieve that, and then estimate the level of effort and cost to achieve that particular SLA. Reasonableness is relative, but once there is agreement regarding what is reasonable and expected that information should be published and metrics should be gathered to track compliance.
2. Document Problem Events. What were the symptoms? How did they present themselves? What troubleshooting steps were taken? What was the result of each step or test? How was the problem resolved? What was the root cause of the problem? Historical information like this is priceless, especially for problems that occur infrequently.
3. Automate Production Monitoring. Automation is more efficient from a manpower perspective, is often much faster at identifying problem conditions or situations, and can proactively alert the support team to potential issues (e.g., running our of disk space, high user count, hardware errors).
4. Create a Troubleshooting Decision Tree. Make it easy for the support staff to quickly identify the appropriate next steps for resolution of the problem.
5. Create a Sense of Urgency. The support staff needs to understand the impact of downtime on the user community, and what is expected from them relative to SLAs. If there is a problem the team needs to respond accordingly, taking whatever steps are necessary to quickly resolve the problem at hand. If vendor support is required this message of importance and urgency needs to be conveyed to the vendor.

Capacity Planning & Management

1. Gather sizing metrics on a regular basis and plot that data. Look for trends and use that information to identify potential capacity issues. For example, using this approach you may be able to determine that a disk location will fill in approximately 6 months given the current rate of growth. Knowing that you can begin investigating the best way to deal with this growth and schedule corrective action before it is required.
2. Look at System Utilization relative to the System Configuration. For example, if a Windows Terminal Server systems has 16 GB of RAM but seldom utilizes more than 2 GB of the memory available, is this a problem? Is there anything that can be done to improve this? In this specific case the limitation could be the 32-bit version of Windows 2003 Server, but the questions should always be asked and investigated.

Summary

Each and every one of the best practices listed has value and adds value. Consistent use of these best practices results in higher overall quality, improved efficiency, and greater customer satisfaction. This is due to a reduction in errors and associated rework, and the minimization of unplanned service outages. As experts in our field, consultants need to ensure that we set a good example for our team and our clients, and this is one way that we can do this. The same holds true for staff at any company.

About the Author

Chip Nickolett, MBA, PMP is the President of Comprehensive Solutions, and has over 22 years of experience in the Information Technology including programming, systems analysis & design, database design & administration, and project management & implementation. He has worked for several software companies and is very familiar with the professional practices employed by those types of companies. For more information please see http://www.Comp-Soln.com/chipn.html.

Let Us Help You Succeed!

Call today to discuss ways that Comprehensive Solutions can help your organization save money and achieve better results with your IT projects. We provide the ***confidence*** that you want and deliver the ***results*** that you need.

Comprehensive Solutions
4040 N. Calhoun Road
Suite 105
Brookfield, WI 53005
U.S.A.

Phone: (262) 544-9954
Fax: (262) 544-1236