GOOD SOFTWARE MANAGEMENT
GUIDELINES FOR DEVELOPERS

Geoff Halprin
The SysAdmin Group Pty Ltd
(C) Copyright 1995-1999, The SysAdmin Group. All Rights Reserved.

Release: 02.12 (1999-07-16)

1 INTRODUCTION

Software development projects need to be pursued with the whole of the product life cycle in mind. This whole-of-life-cycle approach is necessary because, as has been well documented, 60% or more of the cost of software accrues from the in-service support and operation of these systems.

There is, thus, substantial value to be gained by having developers include system support requirements within the overall requirements for a development project. Clients generally do not specify these non-functional requirements, however, as they are neither aware of, nor focussed on such concerns. Moreover, support requirements are, by their nature, generic, and a single set of operational support specifications for an environment will yield a better return-on-investment than individual per-application specifications.

This document is intended as a series of guidelines which represent the experience of a number of years of supporting both commercial and custom software applications across a wide variety of platforms.

These guidelines are not intended to suspend commonsense and good judgement. The applicability of each point detailed below needs to be addressed on a case-by-case basis.

1.1 SCOPE

This document is a definition of the minimum requirements from developers with respect to the installation and operational management of custom-developed software applications for deployment into a distributed computing environment.

This document is specifically tailored towards the UNIX and the Solaris operating system, although the concepts are readily transportable to other versions of UNIX and other operating systems.

1.2 AUDIENCE

It is assumed that the reader has a basic knowledge of UNIX, and some exposure to the principles of UNIX systems administration.

2 RATIONALE

The basic problem with most software products is that they make the very wrong assumption that they have the entire target host(s) to themselves, and, thus, do not have to share resources and can make large assumptions about their environment. (This false assumption can, in part, be attributed to the typical development environment where this is more likely to be the case.)

Modern reality is that very few hosts are dedicated to a single application. Even on the rare occasions that this may be true, these hosts are still the subject of ongoing change as they are tailored to the unique and continually changing environment of the customer.

Moreover, all hosts are also subject to upgrades and changes of infrastructure such as operating system releases, system patch levels, new product installations, shifting data locations and other work to maintain the stability, robustness and workload balance and to improve availability. Thus, even an intimate familiarity with the target host environment on the part of the developer is of limited value and relevance.

It is thus essential that all applications are "well behaved" with respect to how they interact with other applications and the shared resources of the host, network and environment into which they are to be deployed.

3 Principles for Good Support

1. **Application Isolation**. It is important to segregate application data and files from system data and files (and other applications). All hosts are subject to regular system upgrades and patching, and so the more intertwined the application is with the system (with respect to file placement), the more likely it is to fail after such an upgrade.

2. **No Magic Numbers**. Applications must not embed any value that might be subject to change. This includes things such as; host IP addresses, Oracle installation directory, location of the application, PATH components, usernames, etc. These should all be isolated into a single environment script. See section 4.2 for details.

3. **Data Abstraction**. Every application consists of a number of distinct types of program and data. These have different characteristics in terms of growth, security and access behaviour. It is important for the tuning and capacity planning of a system that these data types are distinguished between, and that the system administrator has control over where this data is placed.

4. **Least Privilege**. It is important that applications use appropriate system features to protect access to the application data and, equally importantly, access to the operating system (and hence other data) from within the application.

5. **Good Documentation**. It is important that all applications are well documented in terms of their ongoing management. This includes installation instructions, and user, data, log file and availability management procedures.

6. **Consistency**. Consistency is the first cousin of predictability. Predictability is essential to effective troubleshooting and maintenance. Eg. Programs should produce error and diagnostic messages that are consistent in their format, so that automated tools can process these messages.

7. **Programs Should Handle Failure Gracefully**. Programs must not assume they are receiving valid data where they do not control the source of that data. All input data must checked before being accepted. This is especially true of data from a network source, such as an upstream application. Where a problem is encountered, a meaningful error message should result.

4 REQUIREMENTS

4.1 PARTITIONING OF DISTINCT FILE TYPES (DATA ABSTRACTION)

The application must make clear distinction between each of the types of files presented below.

A. APPLICATION BINARIES. [APP_BIN]

These should generally be located in a hierarchy beneath **/opt**; i.e. **/opt/APP**. This hierarchy should have at least {**bin**,**lib**,**sbin,man,doc**} subdirectories.

B. CONFIGURATION FILES. [APP_CONF]

Configuration files control the behaviour of the application. They should not require frequent changes, and should be basically static in nature.

These should generally be located in **/etc/opt/APP**.

C. PERSISTENT DATA FILES. [APP_DATA]

Persistent data files are those that must be located where they will not be accidentally erased by such events as housekeeping or system reboots. These should generally be located in **/var/opt/APP** (for shared data local to the host), or the user's home directory for personal application data.

The documentation (and installation script) must specify the largest expected size of these files (or a formula for this calculation), such that the administrator can be sure to choose or create a suitable location.

D. TRANSIENT DATA FILES. [APP_TMP, APP_BIGTMP]

Transient data files are those that are short lived (ie. usually the duration of a run, but always less than one day). These are usually stored in **/tmp**, but where there are a large number of these, and/or they are large in size, an alternative location such as **/var/spool** may be more appropriate. However, always be careful to avoid placing these files in an area which may cause data loss should it be filled by temporary files. To this end, it may appropriate to create a new area such as **/var/opt/APP/tmp**.

It should be noted that **/tmp** is usually cleaned on system boot, and also often subject to automated house-cleaning each night. Do not use **/tmp** for files or named sockets that have a life expectancy of greater than a day.

The documentation (and installation script) must specify the largest expected size of these files (or a formula for this calculation), such that the administrator can be sure to choose or create a suitable location.

E. LOCK FILES. [APP_LOCK]

Lock files should be located in **/var/spool/locks**.

F. LOG FILES. [APP_LOG]

Log files should be stored in **/var/log**, along with other system log files. See section 4.4.3 for more on log files.

The above abstractions allow system administrators to better plan for system growth, and to also tune the application and data placement strategies for better disk load balancing and data management policies.

Importantly, all these areas are clearly distinct from operating system areas, will be preserved across operating system upgrades, and can be readily migrated between hosts.

The installation process must prompt the operator during installation for the relevant directory within which to store each class of file.

This information should be stored in a central configuration file and used by the application environment script (see section 4.2 below). The suggested name for each of these is in the headings above.

It should be noted that often a host might have multiple versions of the same application loaded, such as during upgrades and acceptance testing. Following the rules in this section will greatly reduce problems associated with this.

4.2 USER ENVIRONMENT

One of the most important aspects of good programming is information hiding. Yet this very trait seems to be ignored when dealing with application-environment integration.

Applications are often migrated between hosts – from development to production being the most obvious example. It is vital that this process be as straightforward as possible. Underlying infrastructure, such as database servers, are also upgraded beneath applications. Such upgrades should not entail major re-writing of application code but should, for the most part, be invisible to the application.

The most important step towards this goal is to isolate all variables and magic numbers that relate to the host environment into the one place. That place can be an application wrapper script, a database table or a configuration file. The latter is the recommended option, as such a file is maintainable by the system administrator, without programmer intervention.

Doing this means that there is one point of maintenance when underlying software is updated or moved, vastly reducing the maintenance overheads.

This, in conjunction with the data abstraction previously described, provides the single largest improvement to system maintenance overheads.

To this end, specific requirements include the following:

1. A detailed definition of any expected user environment, including PATH and other environment variables, must be provided.

2. Allow as many aspects of behaviour as reasonable to be controlled via environment variables.

3. Provide a sample application wrapper script which sets appropriate environment controls, and then executes the application. Where there are multiple application entry points, then isolate these environmental controls into a separate script, and ensure that all entry points use this script to establish their environment.

4.3 SOFTWARE LICENCES AND ENABLER CODES

The documentation must list all software licenses and enabling codes required to ensure the correct and legal continued operations of the application and related software.

4.4 REGULAR MAINTENANCE DUTIES

4.4.1 AVAILABILITY MANAGEMENT

1. Clear instructions for the start-up and shutdown of the application, in the form of a **/etc/init.d** script must be provided.

2. A statement of any prerequisite services must be provided. (i.e. which services in **/etc/init.d** must be started prior to the application.)

3. A script to interrogate the current system status must be provided. It should return at least a status of "available" (operating normally), and "not available" (indicating some form of error that may need investigation). This script should provide a silent option that returns an exit code of zero for fully operational, and non-zero to indicate an error.

4. The application should use SYSLOG to log application start-up and shutdown messages, and in any case must log these messages by some time-stamped mechanism.

4.4.2 USER MANAGEMENT

1. Clear instructions for the addition of a new user to the application must be provided. Where multiple classes of user exist, then the instructions should provide information on the management of each of these, and the authority that is required to manage each level of user.

2. Clear instructions for the disabling of a user must be provided.

3. Clear instructions for the deletion of a user must be provided.

4.4.3 LOG FILE MANAGEMENT

1. Clear instructions on how to and how frequently to roll over application log files must be provided. If particular services are required to be shutdown to achieve this function correctly, then this must be stated.

   The preferred mechanism for managing log files is to close and re-open the log files upon receipt of the HUP signal.

4.4.4 DATA MANAGEMENT

1. Clear instructions for performing a correct backup of a stable snapshot of the application and its data must be provided. If particular services are required to be shutdown to achieve this function correctly, then this must be stated.

2. Clear instructions for a full restore from backup of the application and its data to a previous stable state must be provided.

3. Clear instructions for a partial restore from backup of the application data from a previous stable state must be provided. Such types of partial restore as are appropriate to the application should be specified.

4. Clear instructions for the roll-forward and roll-back recovery of application data must be provided.

5. Clear instructions pertaining to the frequency with which, and how to perform data archiving, data purging and space compaction, as is appropriate for the application.

4.5 INSTALLATION INSTRUCTIONS

1. Where reasonable, make use of the operating system software package sub-system (eg. Solaris' **pkgadd(1M)** and associated utilities). This will "register" it on the system, and make the application visible to standard utilities.

   It is essential that at a minimum, a skeleton package be defined which includes version control information.

2. Do not write install scripts which run as *root*. If certain installation actions must be performed with *root* privileges, these must be isolated into a separate script that can be run independently of the main installation script. This script must be available for direct code inspection prior to its execution.

3. Do not modify system files without specific warnings for each such file. Often these system configuration files have already been customised for other applications, and so an install script cannot assume a particular state or content for these files. Where appropriate, **patch** (GNU) format change files or other relative updates should be used in preference to direct replacement of these files.

4. The installation script must be able to be re-run after a partially completed install. Alternatively, an installation back-out script must be provided to return the system to its original state. These must do extensive error checking to ensure the success of each step.

5. You must provide a separate post-installation verification script. This script will perform all reasonable checks to ensure that the application is correctly installed, including relevant file and directory permissions, and necessary additions to system files.

6. For database applications, no installation scripts should run as the database owner (e.g. *oracle*). Those scripts that do (such as to create a new database user to own the application or to create a separate database instance) must be isolated and well documented. (See note 2 above.)

7. A separate script must exist to create/recreate database tables, indices, and set/check appropriate permissions on database objects. This is vital for correct data restores.

8. The installation process must create full and extensive logs of all activities. It must inform the operator as to the location of this log upon exit.

9. Release notes detailing changes and implementation procedures should accompany new releases of an application.

In short, all actions performed by installation scripts must be documented and logged.

4.6 NETWORK PROTOCOLS AND PORTS

The documentation must list all protocols used by the application components, either by well-known protocol name (eg. FTP, HTTP), or by port and protocol (eg. 139/TCP). This list must include the direction that any connection is established for connection-oriented protocols such as TCP, and the direction of datagrams for connection-less protocols such as UDP. A tabular form of this information is preferable. See below for examples.

It is recommended that the **/etc/services** and **/etc/rpc** files (or equivalent maps) are updated to reflect any TCP and UDP ports, and RPC program numbers used. Don't assume a particular port will be available for use. Use the service name (as recorded in **/etc/services** instead).

This service name must be set in an environment variable, and not hard-coded. It is likely that at some point multiple versions of an application will reside on the one host, and this will greatly enhance portability and debugging.

Similarly, all host names must not be specified directly, but via an environment variable that is defined *in one place* (ie. APP_CONF).

Where certain restrictions exist over a communications channel, such as the restriction of allowable connections based upon the class of user, then these should be stated. In general, an application will consist of a number of classes of user, each with varying privileges. These classes should all be defined, and used in this table to detail their requirements.

Eg. Text Form: An HTTP server listening on TCP/80 (203.4.203.57:80), with allowable client connections from anywhere on the Internet.

Eg. Table Form:

Destination		Source		Class
Address	**Port**	**Address**	**Port**	
203.4.203.57	TCP / 8080	*	*	User
203.4.203.57	TCP / 6666	STAFF-MACHINES	>=1024	Staff

4.7 ERROR HANDLING

The application should trap as many potential errors as possible, and must report all errors in a consistent format. This allows for both (a) more accurate comprehension by users, and (b) automated processing of error logs (such as to generate alerts).

Each error message should be uniquely identified, such as through individual numbers. Program documentation should include all possible places in the source from which a particular error can be generated. User documentation should include a list of all possible errors (in number order), and any suggestions as to the cause and resolution.

Error logging should be via the SYSLOG facility.

4.8 TECHNICAL DESCRIPTION

The developer should provide an accurate technical description of the application, to aid systems administrators in trouble-shooting, performance management, capacity planning, and disaster-recovery / business-continuity planning.

1. A list of any special application requirements:
    - special hardware requirements,
    - special environment prerequisites and assumptions,
    - Prerequisite packages / software / patches and release levels.
2. Document the expected number of processes (background and per user), swap space and memory requirements. Document any aspect that could impact performance of the application, such as during a Stress and Volume Testing (SVT).
3. An architecture diagram should be provided including process names (as seen by **ps(1)**). This should also provide information relating to scalability and replication issues.
4. A data flow diagram to assist with load balancing, etc.
    - Traffic flows between separate processes on the same host.
    - Traffic flows between separate processes over the network.
    - Transactional traffic flows.
5. A list of special configuration files and commands which define this installation.

4.9 GENERAL SECURITY CONSIDERATIONS

1. No daemon (background) processes are to run as 'root' or the database owner.

2. No world write-able files or directories are to be used. The only exceptions to this are shared directories which have the 'sticky' bit set, such as **/tmp**.

3. Use UNIX account authentication where this is appropriate.

   Where it is more appropriate to maintain a separate application level user authentication sub-system, then do so. Where you are maintaining application-level user authentication, make all efforts to keep it in sync with system password policy (ageing, minimum content, etc).

4. Use UNIX group permissions to restrict access to files where reasonable.

5. Under no circumstances are passwords to be stored in clear-text format. Avoid the transmission of passwords in clear-text across networks wherever possible.

4.10 GENERAL PROGRAMMING CONSIDERATIONS.

1. Always dynamically link libraries. Vendors often release patches relating to bugs and security, and these updates will have no effect should a program be statically linked.

   The exception to this is where a library is present on the development host but not on the production host, and this library relates to a development product (which is not present on the production platform). In this circumstance, you must statically link the relevant library. Be sure to still dynamically link all system libraries.

2. Don't assume all users are in **/etc/passwd**. The system may be running NIS or another naming service. This is also the case for **/etc/services** and **/etc/rpc**.

3. Compile the application with the symbol table intact (eg."-g" flag on most C compilers). This information is essential for debugging.

4. Be sure to comply with all relevant standards, such as RFCs.

5. Allow for Fully Qualified Domain Names (FQDNs).

6. Allow for ***very*** long pathnames and ***very*** long file names.

7. Use the UNIX SYSLOG facility to log all important errors.

8. Don't tie the application to particular userids or usernames. These should be specified by environment variables in the application environment script (and prompted for during installation).

9. Use relative symbolic links within an application tree to aid in any relocation of the application hierarchy.

5 References

1. Solaris filesystem(5) man page.
2. Solaris Application Packaging Developer's Guide.
3. "No You Can't Have Root (How Software Vendors Can Help System Administrators)," Craig Bishop. SAGE-AU'94 Conference.

Thanks to many people for feedback on the contents. Those who contributed include: Chris Green, Giles Lean, Neil Buckingham, John Wadelton, Jason Lee, Bill Kummer.

END