Toward Quantifying System Manageability

George Candea
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
george.candea@epfl.ch

Abstract

Manageability directly influences a system's reliability, availability, security, and safety, thus being a key ingredient of system dependability. Alas, we do not have today a good way to measure manageability or reason quantitatively about it, and this is a major hindrance in improving systems' ease of management. In this paper, we propose a manageability metric that aims to be objective, intuitive, and broadly applicable. We hope such a metric will help software developers make the right design tradeoffs and will also foster fair competition on the basis of manageability. We also offer preliminary thoughts on incorporating this metric into a benchmark.

1 Introduction

It is no longer necessary to argue that managing enterprise computing systems is complex and time-consuming, or that the cost of managing IT infrastructures far exceeds the hardware and software costs—numbers speak for themselves: IT operations account for 50%-80% of today's IT budgets [3], amounting to tens of billions of dollars yearly. Besides the bottomline, poor manageability also impacts reliability, availability, and security in harder-to-quantify ways. As human error becomes the dominant cause of unscheduled downtime [9], we desire systems that are easier, cheaper, and quicker to manage.

The greatest manageability challenge is posed by stateful systems (e.g., databases, filesystems). By contrast, stateless applications (e.g., Web servers) require little configuration, can be scaled through mere replication, and are reboot-friendly. While one administrator can manage 100s to 1000s of Web servers, it takes approximately one administrator for each TB of data in a database [6]. The number of knobs on stateful systems is overwhelming: the Oracle DBMS has 220 initialization parameters and 1,477 tables of system parameters [13], while its "Administrator's Guide" is 875 pages long [12].

There is little hope to improve manageability without a well-defined, objective way of measuring this kind of improvement; in fact, an important tenet of engineering is to always assess quantitatively progress vs. regress in the design and implementation of a system. So we need a way to measure manageability, i.e., a benchmark.

Benchmarks of various sorts (especially for performance) emerged the moment systems, due to their complexity, were no longer comparable simply based on their specifications. A simple example is a 3 GHz microprocessor that may provide less computational power than a 2 GHz one. Suitable benchmarks can easily highlight such differences, typically relying on mimicking a workload deemed representative for the system in question.

A manageability benchmark must be objective, or else comparison between systems makes little sense (for instance, a study found that Oracle 10g is more manageable than IBM's DB2 UDB v. 8.2 [17], while another study found the exact opposite [2]). Moreover, when benchmark results, such as the ones from SPEC and TPC, are audited and independently verifiable, their validity and acceptance are increased. But the key element in an objective benchmark is having a clearly defined and broadly accepted metric along with an objective methodology for conducting the benchmark. This metric is important not only for comparing systems from a purchaser's perspective, but it is also a tool for architects and developers to make tradeoffs in their design decisions.

Comparing real systems is always challenging, because their functionality and complexity can be broad and an in-depth comparison is often time-consuming. The value of specific features can sometimes be judged only by direct experience, after deployment, and detailed features often cannot even be compared directly. Nevertheless, a benchmark should provide a good balance between useful quantification and simplifying assumptions.

In this paper we introduce a manageability metric that can help engineers ascertain how close their systems are to a chosen manageability target and how to make manageability-improving tradeoffs. We also outline a methodology for evaluating system manageability, thus taking a first step toward a manageability benchmark.

2 A Manageability Metric

In most fields of engineering, having a specific property as a first-order design goal requires a suitable metric to quantitatively measure progress toward that goal, as well as evaluate how different design decisions impact the desired property. In choosing such a metric for system manageability, we aim for three properties: simplicity, intuitiveness, and wide applicability.

Before attempting to quantify manageability, it is worth deciding specifically what manageability is. The ISO-9126 standard for the evaluation of software quality defines maintainability—a "close cousin" of manageability—as a set of attributes that bear on the effort needed to make specified modifications: stability, analyzability, changeability, and testability [7]. These "ilities" aim to capture the ease with which a software system or component can be modified to adapt to a changed environment, correct faults, or improve performance. Within the framework of this definition, we consider a system's manageability to be determined by *the level of human effort required to keep that system operating at a satisfactory level*.

The unit of measure we employ is the "management unit" (MU), a generic unit similar to the "international unit" used in pharmacology[1]. Business persons, however, think of poor manageability as a liability that increases administration costs and thus total cost of ownership; said differently, those with decision power think of manageability in monetary terms. While using dollars as a unit of measure offers the benefit of clarity and objectivity, it lacks universality. For instance, the administration cost of a mail server is typically lower in India or Bangladesh than in the US or Switzerland, due to wage differences (hence the frequent outsourcing of such IT services). Lack of universality would reduce our ability to use the metric for absolute scoring of a system's manageability in a reasonably time-invariant way.

In quantifying manageability, we build upon the premise that the root cause of management difficulties lies in *exposed complexity*. Internal complexity in modern software systems is inevitable, but exposing it to the administrators can be avoided. In a system with externally-visible complexity, there are many steps required to achieve a given task, and some of these steps take very long; at each step, something might go wrong, requiring the administrator to make a decision on the fly. Frequent decision points also offer opportunities for mistakes, the prevention of which requires highly-trained professionals and large budgets. High-level management tasks cannot be performed atomically and, when something goes wrong at an intermediate step, the system can enter an unusable state. The manageability metric should therefore encourage the simplification of system management paradigms.

[1]The international unit (IU) is used to characterize the biological effect of a substance, such as vitamin E. To define an IU for a given substance, the International Committee on Biological Standardization provides a reference preparation of the substance in question, arbitrarily sets the number of IUs contained in that preparation, and specifies a biological procedure to compare other preparations of that substance to their reference preparation.

We can think of system management as a collection of tasks the administrators have to perform to keep a system running in good condition: deployment, configuration, upgrading, tuning, backup, failure recovery, etc. We can approximate *complexity* of a management task by the number of discrete, atomic steps ($Steps_{\mathrm{i}}$) required to complete $Task_{\mathrm{i}}$; the larger $Steps_{\mathrm{i}}$, the more inter-step intervals, hence the greater the opportunity for an administrator to make a mistake. An operating system install, for example, entails $Steps_{\mathrm{install}}$ in the tens or hundreds.

The step is a unit of atomic management work. This means that, if a step fails in the middle, whatever has been done as part of that step can be easily, cleanly, and predictably undone. The precise definition of a task, on the other hand, is not directly relevant to the proposed metric, so we can think of it as a mere collection of steps.

The metric must also account for the duration of management operations, because, the longer they take, the greater the opportunity for unrelated failures that occur in between steps to impact atomicity and integrity of the overall task: power failures, administrator distractions, etc. We therefore add to the metric the notion of *efficiency* of management operations, which is approximated by the time $Time_{\mathrm{i}}$ the system takes to complete $Task_{\mathrm{i}}$. A trouble-free installation of an OS today would take on the order of $Time_{\mathrm{install}} \approx$ 1-3 hours.

If N_{i} represents the number of times $Task_{\mathrm{i}}$ is performed during a time interval $TotalTime_{\mathrm{eval}}$ (e.g., one year) and $N_{\mathrm{total}} = N_1 + ... + N_{\mathrm{n}}$, then $Weight_{\mathrm{i}} = N_{\mathrm{i}}/N_{\mathrm{total}}$ is $Task_{\mathrm{i}}$'s relative weight of occurrence during the system's lifetime. Surveys [1, 8, 10] or empirical studies can provide realistic values for $Weight_{\mathrm{i}}$. The equivalent of getting such weights was successfully done for workloads used in performance benchmarks, like SPEC, TPC, and Linpack.

We express manageability in MUs with the following formula; higher values indicate better manageability:

$$Manageability = \frac{TotalTime_{\mathrm{eval}}}{\sum_{\mathrm{i}=1}^{n} Weight_{\mathrm{i}} \times Time_{\mathrm{i}} \times Steps_{\mathrm{i}}}$$

This formula expresses the fact that manageability is reduced proportionally to how long the management tasks take and to how many atomic steps are involved in each such task. The fewer steps there are, the lower the exposed complexity of the system; the faster the management tasks can be completed, the lower the likelihood of trouble. The less management a system requires (i.e., the longer $TotalTime_{\mathrm{eval}}$ for the same N_{total}), the easier it is to manage; equivalently, the less the system needs to be managed, the better. These are rules-of-thumb known to every sysadmin, but sadly not obvious to every programmer.

The reason we use task-level timing ($Task_{\mathrm{i}}$) instead of step-level is because steps are typically system-specific (e.g., different in MS Exchange vs. Sendmail). At the task level, however, we can reasonably expect to identify a substantial set of tasks that are common to most systems within a class (e.g., all mail servers, or all database servers). Thus, the metric can be used to compare systems from within a class to each other, without having to dive too deeply into system details.

While the complexity and efficiency measures are objective, their relative importance depends on the administrator's opinion: an improvement in complexity may be valued more than an improvement in efficiency or vice-versa. In a standalone metric, one could capture this differentiated weighting in a coefficient α and replace $Steps_{\mathrm{i}}$ in the formula above with ${Steps_{\mathrm{i}}}^{\alpha}$. The metric can then be used for designing systems targeted at a specific administrators audience defined by an α derived from specific numerical examples, such as "halving the number of steps in doing backup would triple the manageability of my deployment." However, the subjectivity of α makes it unsuitable for a broadly applicable metric.

The formula assumes each step is atomic, i.e., that the system cannot be left in an inconsistent state except at the boundary between two steps. We are not aware of a way to automatically verify a vendor's atomicity claims, so we must rely on the vendor to provide for each action in a step (i.e., for each "substep") a compensating action that can undo its effects. E.g., if a UNIX user group is created, then there should be a way to remove that user group. Some actions are not undoable (e.g., deleting a disk partition), in which case they must each be a single step, not contained in a more complex step. Atomicity of steps can be inspected and certified by benchmarking bodies like the TPC or SPEC, or we can rely on online reputation systems (like the ones used by eBay, Amazon, and shopping search engines) to uncover mistakes in vendors' specifications.

There are a number of other system attributes, not included in the proposed metric, that have a more or less direct impact on manageability: system size, number of nodes, complexity of interactions and dependencies between components, security requirements, volume of data handled, etc. There is an inherent tradeoff in how precise a metric is vs. how approachable and adoptable it is; we see this tradeoff at work in all widely used benchmarks. For this reason, we excluded attributes whose absence simplifies the metric substantially while still keeping it accurate to a first degree of approximation.

The role of a manageability metric is not only to measure, but also to guide developers in making day-to-day choices, which is why we prefer an intuitive, easy-to-remember formula over a complex one. By contrast, a more precise formula would include the probability distributions of $Steps_{\mathrm{i}}$ and $Time_{\mathrm{i}}$, to account for multi-step tasks that encounter failures and cause the administrator to branch to a different sequence of steps (in the formula we implicitly use averages of those distributions). In this same vein, we do not account for partial ordering constraints between steps—while these do hurt system manageability (e.g., by offering the opportunity to do steps out of order), they complicate the formula. Such complexity would sabotage the ease of adopting our proposed manageability metric.

3 Metric + Workload = Benchmark

A metric on its own can be used by development organizations in their quest to improve their products' manageability. However, to compare different systems to each other, as may need to be done in a purchasing decision, requires a benchmark. This requires pairing the manageability metric with representative management workloads. As long as we aim for workloads that are specific to a class of systems (e.g., a typical database management workload, a smartphone management workload), defining them can be done based on original studies, published surveys [1, 8, 10] or best practices documents.

The workload description consists of a set of management operations along with the weights describing their relative frequency of occurrence. These workloads would be chosen and published by industry-wide bodies, in the style of TPC and SPEC.

The choice of $TotalTime_{\mathrm{eval}}$ depends on the type of system. For enterprise systems, it is usually three years, as this is the typical cycle for provisioning and replacing systems. For consumer electronics or other types of software, shorter survey periods may be more representative (e.g., one year for smartphones). If we break down the lifetime of a system into segments of length corresponding to the survey duration, then the weights of the management operations should be approximately identical across the different segments. Choosing a $TotalTime_{\mathrm{eval}}$ for the benchmark does not imply that the user of the benchmark must wait that long to evaluate a system, rather it means that the chosen workload describes what is expected to happen in that time interval. The time segments between management operations are irrelevant when computing manageability.

We illustrate with a hypothetical example of a database management workload (Table 1). Say $TotalTime_{\mathrm{eval}}$ is three years and, for each $Task_{\mathrm{i}}$, the indicated $Weight_{\mathrm{i}}$ is

$\mathbf{Task_i}$	$\mathbf{Weight_i}$ over 3 years
Installation	1 / 57
Major software upgrade	3 / 57
Minor required patching	12 / 57
Migration to new hardware	1 / 57
Failure recovery (e.g., bad disk)	3 / 57
Backup (setup + validation)	6 / 57
Recovery from backups	3 / 57
Disk space increase/decrease	9 / 57
Performance tuning	9 / 57
Schema management	10 / 57

Table 1: Hypothetical management workload for the class of database systems ($N_{\text{total}} = 57$).

the relative weight of that task within this interval. Over the course of three years, one could expect the database system to be installed and suitably configured once; every year, the system undergoes a major software upgrade; once a quarter, various patches are applied; and so on.

Software is steadily becoming more of a service than an artifact: when purchasing software, we also purchase a limited right to future updates, notifications of security vulnerabilities, technical support, etc. For example, high-end storage appliances have a graphical interface for common operations; while sufficient for most provisioning and management, it does not allow for the resolution of exceptional problems. In such cases, the user submits a trouble ticket and relies on the vendor for assistance. The manageability of a particular software system can be substantially influenced by the quality of technical support, documentation, etc. Our proposed metric, however, sets out to measure solely the artifact, not the entire service package. We leave the evaluation of the artifact's ecosystem to consumer review websites and other such means of assessing end-to-end user experience.

The duration of steps can sometimes be a function of attributes not included in the metric (see §2). For instance, the duration of backup/recovery is typically highly correlated with the volume of data involved, the number of assigned IP addresses is proportional to the number of nodes, etc. Such dependencies are quite common in benchmarks, and the usual approach is to choose a few reference configurations that fix these variable attributes. For example, the TPC-H benchmark [16] has five categories corresponding to databases of 10 TB, 3 TB, 1 TB, 300 GB, and 100 GB. A similar approach can be taken for the manageability benchmark as well.

The final component of the benchmark is a load driver that implements the prescribed workload and can time the individual steps. The specifics of this driver depend on the interface of the system class under consideration.

4 Applying the Benchmark

To illustrate the use of the benchmark, we show how one might compute the manageability of the Oracle DBMS.

Consider installing the DBMS: According to the documentation [11], installing on Linux consists of pre-installation, installation, and post-installation actions; we can therefore think of installation as consisting of the three corresponding tasks. The pre-installation task consists of 75 individual steps, such as creating necessary user groups, setting various kernel parameters and using fdisk to set up partitions [11]. The installation task consists of 12 steps, such as downloading an installer, authenticating, and running the installer. Finally, post-installation consists of 28 steps, such as downloading and installing patches and running the Enterprise Manager Console.

For simplicity, let us evaluate the manageability with respect to installation only. Each of the 75+12+28=115 installation steps would be executed by the load driver and timed; the sum of these times represents $Time_{install}$, say 12 hours on a reference hardware configuration. Installation is done once during $TotalTime_{\text{eval}}$, so $Weight_{install} = 1/57$ (from Table 1). The driver then computes $\sum_{\text{i}=1}^{1} Weight_{\text{i}} \times Time_{\text{i}} \times Steps_{\text{i}} = (1/57){\times}12{\times}115 = 24$ and finally computes the overall manageability by dividing 3 years × 365 days × 24 hours = 26,280 hours by this sum, giving a manageability value of $26,280/24 = 1,095$ MUs.

How would a developer use the benchmark as a guide to improve Oracle? A first step would be to reduce the number of steps involved in performing the three tasks related to installation. Many of the pre-installation steps could be automated by atomic scripts. If all necessary files were included on the CD or in the RPM package, there would be no need to download installers from the Web, which would avoid the risk of corrupt downloads or bored administrators. If these measures halved the installation time to $Time_{\text{install}} = 6$, the manageability score of the DBMS would double to 2,190 MUs.

5 The Real World

In this section we analyze several aspects related to the use of the proposed metric in practice: how it can be used by businesses, whether to differentiate between administrators' experience levels, the effect of automation on manageability measurements, the role played by the assumption of step atomicity, and finally a discussion of visibility vs. control in manageable systems.

Business Uses. For better or worse, the key to adoption of the proposed benchmark is for it to provide an easy way to connect good or poor manageability to the financial bottomline. Ultimately, enterprises employ IT solutions in order to improve profit margins. The proposed benchmark can serve to quantitatively reason about investment in IT, such as predicting the savings that would result from using a product with better manageability. Optimizations can even be done across multiple system attributes, like performance and manageability, if suitable utility functions exist.

Consider the case of making a purchasing decision for a new DBMS at an e-commerce site. In the US, one administrator "costs" roughly \$200K/year, including salary, benefits, and office space. Say, for the sake of discussion, that the business value of an IBM DB2 system that achieves 500 tpmC on the TPC-C benchmark [15] is \$1M/year and costs \$2M to purchase and operate over three years, while an Oracle system that achieves 800 tpmC is valued at \$1.4M/year (increased throughput means a larger client population can be supported) and costs \$3M to purchase and operate.

Say the IBM system scores 8,000 MUs on the manageability benchmark and the Oracle system 4,000 MUs. At this hypothetical e-commerce business, an 8,000 MU system might be manageable by a single administrator, whereas a 4,000 MU system might require two administrators. With this information, we can now compute the profit generated by the IBM system over three years as (3 years $\times$ \$1M) $-$ (\$2M + 3 years $\times$ \$200K) = \$400K, and that generated by the Oracle system as (3 years $\times$ \$1.4M) $-$ (\$3M + 3 years $\times$ 2 $\times$ \$200K) = \$0. The outcome would argue in favor of the IBM system. When system manageability can be quantified, it can be included in actual costs and businesses can make informed decisions.

Differences in Administrator Experience. The level of experience with a certain system can vary substantially across administrators; a universal metric must be immune to such variability. It is for this reason that we chose to not include in the metric any factor that depends on the human sitting in front of the keyboard: the $Time_i$ values represent the time it takes the system (not the administrator) to complete $Task_i$. This ensures that $Time_i$ can be measured in a consistent, reproduceable manner.

The role of experience, however, plays a role in connecting a concrete number of MUs to the number of administrators required to administer the system. We expect that, over time, rules of thumb will emerge that can help businesses connect these two quantities.

Automation and Atomicity. Automation of management operations can help reduce the number of steps and the amount of time involved in those operations. Multiple steps can be aggregated into a bigger step (e.g., a script), but the aggregate step must still be atomic; techniques such as checkpointing can be used to provide undoability. The atomicity requirement prevents vendors from subverting the manageability metric through careless wrapping of multiple management steps into one big script.

Atomicity helps avoid the *automation irony* [14], an effect by which automation of frequent easy tasks causes human administrators to become less capable of handling unexpected complex tasks; this happens because automation hinders humans in constructing an accurate mental model of how the system behaves. If a non-atomic script wraps multiple steps and then fails in the middle, it will expose administrators to an unknown system state, requiring them to solve an unexpected problem. Psychologists have found that humans are bad at coping with such situations, especially when under stress [14]. If the aggregate step is atomic, however, administrators are unlikely to be exposed to unexpected complex tasks.

Management by Trial and Error. The fact that the metric requires atomic steps encourages designs with high levels of recoverability. Moreover, if management actions are easy to undo—a stronger property than mere atomicity—we can imagine administrators learning how to manage a system by routinely exploring what-if scenarios. Such a system management paradigm would lend legitimacy and safety to a practice that is already widespread. To our knowledge, this idea of managing by trial and error was first advanced by Brown and Patterson in the context of their work on system-level undo [4].

If the entire process of managing a system was transactional, i.e., all management operations had ACID semantics (atomic, consistent, isolated, and durable), then every task could have one step, and system manageability would be predominantly a function of how long it takes the system to execute management functions. This simplification relates closely to the goal of making most system failures be reboot-curable, in order for system recovery to become simple and predictable [5].

Visibility vs. Control. Advocating a manageability-centric design, that eliminates control knobs and simplifies management, does not necessarily imply eliminating administrators' visibility into the system. The issue of control is relatively orthogonal to that of visibility.

When developing a software system, the discussion of how much direct control to give administrators always comes up. This control is often desired because administrators lack visibility into the reasons for a system's behavior and want to use the control in order to gain that missing visibility. Take for instance a database whose performance has suddenly halved; this may be due to a runaway query, some other process on the same machine updating a filesystem index, the battery of a RAID controller's cache having run out and forcing all updates to be write-through, or any number of other causes. In order to diagnose the slowdown, the administrator will start "poking around" with `ps`, `vmstat`, `mdadm`, etc. This involves many steps, which offer opportunities for mistakes, and takes a long time. However, what the administrator really wants is to know why the system is slow; the control needed to remedy the situation is minimal: kill a query, reboot the machine, or replace a battery.

Administrator mistakes often result from a mismatch between the human's mental model of the system and the system's actual behavior [3]. A system with a detailed, low-level management interface is not necessarily less manageable than one with few management functions—if there is good visibility into the system, then the mental model and the actual system can be well aligned. Thus, when establishing a workload profile for the manageability benchmark, it is advisable to include tasks related to debugging and tuning. While visibility is quite difficult to quantify in a manageability metric, we believe it can be accounted for indirectly through these debugging and tuning tasks: if, in evaluating a network router, the manageability benchmark measures how long it takes to find the number of packets sent and received, then it is quite likely that vendors will opt to conveniently expose these numbers to the administrator. This improves visibility and manageability, without increasing the amount of control an administrator has over the router.

6 Conclusion

Just like security and safety, manageability is generally hard to retrofit in complex systems—it is always easier to build it in from day one. However, in the absence of means to measure manageability and to quantify the various tradeoffs, it is difficult to get the design right. We proposed a manageability metric that combines management workloads and weightings based on real world studies with direct measurement of the number of steps involved in management tasks and their duration. We believe the metric is most useful as a reasoning tool for software designers, but we can also envision it being part of a manageability benchmark. Developing this benchmark can take the software industry a step closer to a systematic approach for building systems that are more manageable and, therefore, dependable.

7 Acknowledgments

The ideas presented here are the result of discussions and debates with my colleagues at both EPFL and Aster Data Systems, as well as Aster's customers. My interest in manageability and recoverability is due in large part to Jim Gray and the Recovery-Oriented Computing project. I wish to thank the anonymous reviewers for their help in improving this paper.

References

[1] R. Barrett, E. Kandogan, P. P. Maglio, E. Haber, L. A. Takayama, and M. Prabaker. Field studies of computer system administrators: Analysis of system management tools and practices. In *ACM Conf. on Computer-Supported Cooperative Work*, 2004.

[2] J. Bloemen and G. Brunner. IBM DB2 UDB V8.2, Oracle 10g, Microsoft SQL Server 2000: A technical comparison. Business eKnowledge Solutions Gmbh, Nov. 2004.

[3] A. B. Brown and J. L. Hellerstein. Reducing the cost of IT operations – is automation always the answer? In *10^{th} Workshop on Hot Topics in Operating Systems*, 2005.

[4] A. B. Brown and D. A. Patterson. Undo for operators: Building an undoable e-mail store. In *USENIX Annual Technical Conference*, 2003.

[5] G. Candea and A. Fox. Crash-only software. In *9^{th} Workshop on Hot Topics in Operating Systems*, 2003.

[6] J. Gray. Distributed computing economics. Technical Report MSR-TR-2003-24, Microsoft Research, 2003.

[7] ISO/IEC TR 9126: Software engineering – product quality. International Organization for Standardization, 2003.

[8] W. Kakes, C. Ling, and A. Brown. What do E-mail system administrators do? http://roc.stanford.edu/retreats/summer_03/slides/bkakes.ppt, 2003.

[9] D. Oppenheimer, A. Ganapathi, and D. Patterson. Why do Internet services fail, and what can be done about it? In *Proc. 4th USENIX Symp. on Internet Technologies and Systems*, 2003.

[10] Oracle database 10g and Oracle 9i database manageability comparison. Oracle Corp., Feb 2004.

[11] Oracle. Oracle 10g installation guide for Linux x86-64. Oracle Corp., May 2006.

[12] Oracle Database 10g Release 2 administrator's guide. Oracle Corp., May 2006.

[13] Oracle Database 10g Release 2 reference. Oracle Corp., May 2006.

[14] J. Reason. *Human Error*. Cambridge University Press, 1990.

[15] The TPC-C OLTP benchmark. http://www.tpc.org/tpcc.

[16] The TPC-H decision support benchmark for ad hoc queries. http://www.tpc.org/tpch.

[17] A. Werman, C. Norris, B. Cohen, J. Becker, and S. Mints. Comparative management cost study: Oracle database 10g and IBM DB2 Universal Database 8.2. Edison Group, Nov. 2004.