IBM Research Report

Reducing the Cost of IT Operations --
Is Automation Always the Answer?

Aaron B. Brown, Joseph L. Hellerstein
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Reducing the Cost of IT Operations—Is Automation Always the Answer?

Aaron B. Brown and Joseph L. Hellerstein
IBM Thomas J. Watson Research Center
Hawthorne, New York, 10532
`{abbrown,hellers}@us.ibm.com`

Abstract

The high cost of IT operations has led to an intense focus on the automation of processes for IT service delivery. We take the heretical position that automation does not necessarily reduce the cost of operations since: (1) additional effort is required to deploy and maintain the automation infrastructure; (2) using the automation infrastructure requires the development of structured inputs that have up-front costs for design, implementation, and testing that are not required for a manual process; and (3) detecting and recovering from errors in an automated process is considerably more complicated than for a manual process. Our studies of several data centers suggest that the up-front costs mentioned in (2) are of particular concern since many processes have a limited lifetime (e.g., 25% of the packages constructed for software distribution were installed on fewer than 15 servers). We describe a process-based cost-benefit analysis that aids in determining if automation will indeed reduce costs. Our analysis indicates that automating a process is beneficial if the process has a sufficiently long lifetime, if it is relatively easy to automate (i.e., can readily be generalized from a manual process), and if there is a large cost reduction (or leverage) provided by each automated execution of the process compared to a manual invocation.

1 Introduction

The cost of information technology (IT) operations dwarfs the cost of hardware and software, often accounting for 50% to 80% of IT budgets [7, 3, 14]. IBM, HP, and others have announced initiatives to address this problem. Heeding the call in the 7th HotOS for "futz-free" systems, academics have tackled the problem as well, focusing in particular on error recovery and problem determination. All of these initiatives have a common message: *salvation through automation*.

After working with corporate customers, service delivery personnel, and product development groups, we have come to question the widely held belief that automation always reduces costs. In fact, our claim is that automation can *increase* cost if it is applied without a holistic view of the processes used to deliver IT services. This surprising conclusion derives from the hidden costs

of automation, costs that become apparent when automation is viewed holistically. While automation may reduce the cost of certain operational processes, it can add additional cost in other processes due to the need to support the automation infrastructure, adapt inputs to structured formats required by automation, and handle automation failures.

In this paper, we use the example of automating software distribution to illustrate the hidden costs of automation. We draw on data collected from several real data centers to help illuminate these costs, and to give an example of how a cost-benefit analysis can be used to determine when automation should and should not be applied. Finally, we broaden our analysis into a general discussion of the trade-offs between manual and automated processes and offer guidance on the best ways to apply automation.

2 Hidden costs of Automation

Throughout this paper, we use software distribution to server machines as a running example since the proper management of server software is a critical part of operating a data center. Our discussion applies to software package management on centrally-administered collections of desktop machines as well. Software distribution involves the selection of software components and their installation on target machines. We use the term "package" to refer to the collection of software resources to install and the step-by-step procedure (process) by which this is done.

Figure 1(a) abstracts the process of software distribution carried out by System Administrators (SAs) at IBM. A request is made to distribute software to a set of servers. The SA starts by obtaining the software resources. Then, for each target, the following steps are done repeatedly. First, the software's prerequisites are checked (e.g., operating system release level, memory requirements, dependencies on other packages). Next comes configuration in which the SA determines the values of various parameters (e.g., server's IP address, features to be installed). Then, the SA performs the install, verifies the result, and handles error conditions that arise. While Figure 1(a) abstracts heavily to illustrate similarities between software installs, we underscore that a
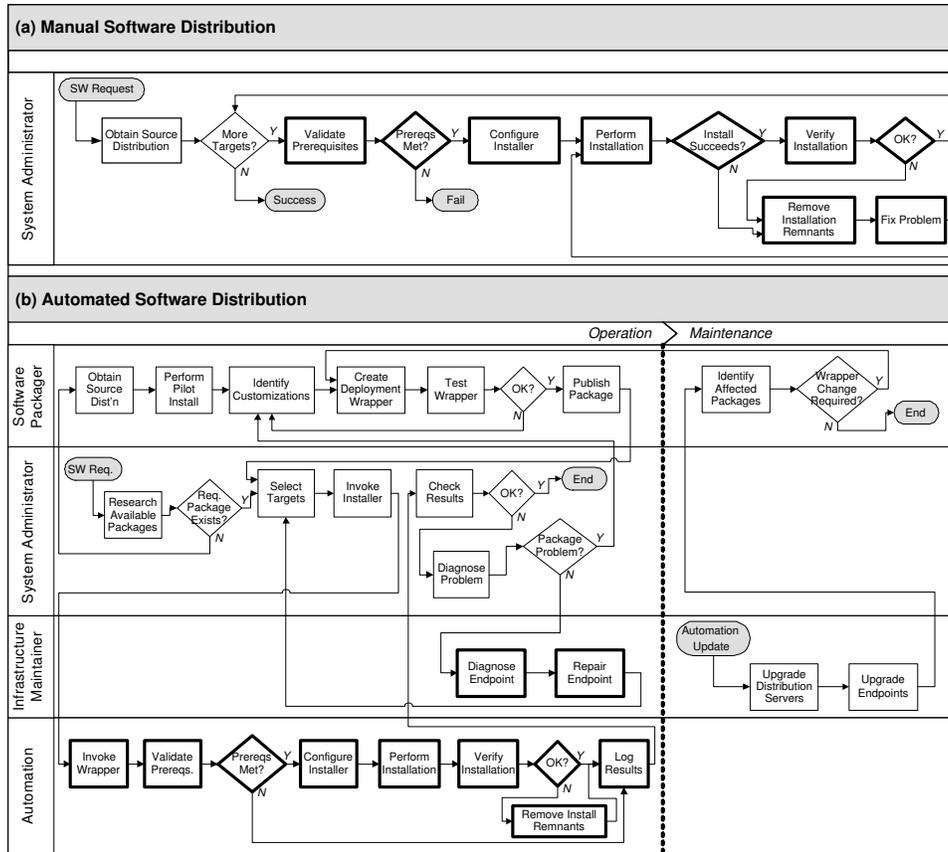
Figure 1: **Manual and automated processes for software distribution.** *Boxes with heavy lines indicate process steps that contribute to variable (per-target) costs, as described in Section 3.*

particular software install process has many steps and checks that typically make it quite different from other seemingly similar software installs (e.g., which files are copies to what directories, pre-requisites, and the setting of configuration parameters).

Now suppose that we automate the process in Figure 1(a). The System Administrator selects a separately developed package that specifies the details of the install process, and the software distribution infrastructure handles the other parts of the process flow in Figure 1(a) based on the information in the package. Have we simplified IT operations?

No. In fact, we have most likely made IT operations *more complicated*. First, the automation infrastructure is another software system that must itself be installed and maintained. (For simplicity, we assume throughout that the automation infrastructure has already been installed, but we do consider the need for periodic updates and maintenance.) Next, using the automated infrastructure requires that information be provided in a structured form (e.g., as software packages), which requires design, implementation, and testing. Last, when errors occur, they happen on a much larger scale than for a manual

approach, and hence additional processes and tools are required to recover from them.

This extra complexity manifests as additional operational processes that must be carried out to support the automation. Figure 1(b) illustrates the process changes that occur when the software distribution process in Figure 1(a) is automated (excluding the processes needed to initially build the automation infrastructure). The diagram is organized as a "swim-lane" chart, dividing process activities into two phases (operations and maintenance, the two columns) and allocating them amongst four *roles* (the swim-lanes, or rows). Roles are typically performed by people (and can be shared or consolidated); we include automation as its own role to reflect activities that have been handed over to an automated system. We see that the automation (the bottom row) has a flow almost identical to that in Figure 1(a). However, additional roles are added for care and feeding of the automation. The responsibility of the System Administrator becomes the selection of the software package, the invocation of the automation, and responding to errors that arise. Since packages must be constructed according to the requirements of the automation, there is a
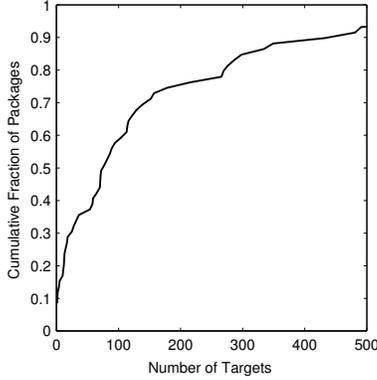
Figure 2: *Cumulative distribution of the number of targets (servers) on which a software package is installed in several data centers. A larger number of packages are installed on only a small number of targets.*

new role of Software Packager. The responsibility of the packager is to generalize what the System Administrator does in the manual process so that it can be automated. There is also a role for an Infrastructure Maintainer who handles operational issues related the software distribution system (e.g., ensuring that distribution agents are running on endpoints) and the maintenance of the automation infrastructure.

From inspection, it is apparent that the collection of processes in Figure 1(b) is much more complicated than the single process in Figure 1(a). Clearly, such additional complexity is unjustified if we are installing a single package on a single server. This raises the following question—at what point does automation stop adding cost and instead start reducing cost?

3   To Automate or Not To Automate

To answer this question, we first characterize activities within a process by whether they are used for setup (the outer part of a loop) or per-instance (the inner part of the loop). Boxes with heavy outlines in Figure 1 indicate the per-instance activities. Note that in Figure 1(b), most of the per-instance activities are done by the automation. We refer to the setup or up-front costs as **fixed costs**, and the per-instance cost as **variable costs**.

A rule-of-thumb for answering the question above is that automation is desirable if the variable cost of the automated process is smaller than the variable cost of the manual process. *But this is wrong.*

One reason why this is wrong is that we cannot ignore fixed costs for automating processes with a limited lifetime. IT operations has many examples of such limited lifetime processes. Indeed, experience with trying to capture processes in "correlation rules" used to respond to events (e.g., [9, 4]) has shown that rules (and hence processes) change frequently because of changes in data

center policies or endpoint characteristics.

Our running example of software distribution is another illustration of limited lifetime processes. As indicated before, a software package describes a process for a specific install. The fixed cost of building a package must be amortized across the number of targets to which it is distributed. Figure 2 plots the cumulative fraction of the number of targets of a software package based on data collected from a several data centers. We see that a large fraction of the packages are distributed to a small number of targets, with 25% of the packages going to fewer than 15 targets.

There is a second reason why the focus on variable costs is not sufficient. It is because the focus is on the variable costs of *successful* results. From Figure 1(b), we see that more sophistication and people are required to address error recovery for automated software distribution than for the manual process. Using the same data from which Figure 2 is extracted, we determined that 19% of the requested target installs fail. Furthermore, at least 7% of the installs fail due to issues related to configuration of the automation infrastructure, a consideration that does not exist if a manual process is used.

The foregoing motivates our development of a simple quantitative cost-benefit model to determine when to automate a process. Let C_f^m be the fixed cost for the manual process and C_v^m be its variable cost. We use N to denote lifetime of the process (e.g., a package is distributed to N targets). Then, the total cost of the manual process is

$$C^m = C_f^m + N C_v^m$$

Similarly, there are fixed and variable costs for the automated process. However, we observe from Figure 1(a) and Figure 1(b) that the fixed costs of the manual process are included in the fixed cost of the automated process. We use C_f^a to denote the *additional* fixed costs required by the automated process, and we use C_v^a to denote the variable cost of the automated process. Then, the total cost of the automated process is

$$C^a = C_f^m + C_f^a + N C_v^a$$

We can make some qualitative statements about these costs. In general, we expect that $C_v^m > C_v^a$; otherwise there is little point in considering automation. Also, we expect that $C_v^m \leq C_f^a$ since careful design and testing are required to build automation, which requires performing the manual process one or more times. Substituting into the above equations and solving for N, we can find the crossover point where automation becomes economical. That is, where $C^a < C^m$.
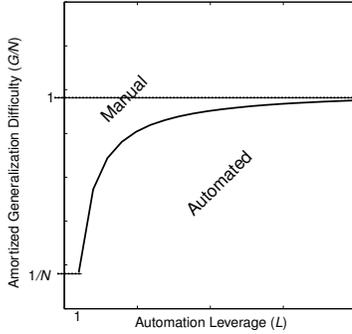
$$N > \frac{C_f^a}{C_v^m - C_v^a}.$$

Figure 3: *Preference regions for automated and manual processes. Automated processes are preferred if there is a larger leverage for automation and/or if there is a smaller (amortized) difficulty of generalizing the manual procedure to an automated procedure (G/N).*

This inequality provides insights into the importance of considering when to automate a process. IBM internal studies of software distribution have found that C_f^a can exceed 100 hours for complex packages. Our intuition based on a review of these data is that for complex installs, C_v^m is in the range of 10 to 20 hours, and C_v^a is in the range of 1 to 5 hours (mostly because of error recovery). Assuming that salaries are the same for all the staff involved, these numbers indicate that there should be approximately 5 to 20 targets for a automated software distribution to be cost effective. In terms of the data in Figure 2, these numbers mean that from 15% to 30% of the installs should not have been automated.

The foregoing cost models can be generalized further to obtain a broader understanding of the trade-off between manual and automated processes. In essence, this is a trade-off between the leverage provided by automation versus the difficulty of generalizing a manual process to an automated process.

Leverage L describes the factor by which the variable costs are reduced by using automation. That is, $L = \frac{C_v^m}{C_v^a} \geq 1$.

The generalization difficulty G relates to the challenges involved with designing, implementing, and testing automated versions of manual processes. Quantitatively, G is computed as the ratio between the fixed cost of automation and the variable cost of the manual process: $G = \frac{C_f^a}{C_v^m} \geq 1$. The intuition behind G is that, to construct an automated process, it is necessary to perform the manual process at least once. Any work beyond that test invocation of the manual process will result in a larger G. Substituting and solving, we find that

$$\frac{G}{N} = 1 - \frac{1}{L}$$

We refer to G/N as the amortized difficulty of generalization since the generalization difficulty is spread across N invocations of the automated process.

Figure 3 plots G/N versus L. We see that the vertical axis (G/N) ranges from $1/N$ to 1 since $G \geq 1$ and $G \leq N$. The latter constraint arises because there is little point in constructing automation that is G times more costly than a manual process if the process will only be invoked $N < G$ times. The figure identifies regions in the $(L, G/N)$ space in which manual and automated processes are preferred. We see that if automation leverage is large, then an automated process is cost effective even if amortized generalization difficulty is close to 1. Conversely, if amortized generalization difficulty is small (close to $1/N$), then an automated process is cost effective even if automation leverage is only slightly more than 1. Last, having a longer process lifetime N means that G/N is smaller and hence makes an automated process more desirable.

This analysis suggests three approaches to reducing the cost of IT operations through automation: reduce the generalization difficulty G, increase the automation leverage L, and increase the process lifetime N. In the case of software distribution, the most effective options are to increase N and to reduce G. We can increase N by making the IT environment more uniform in terms of the types of hardware and software so that the same package can be distributed to more targets. However, the dangers here are twofold. First, increasing N has the risk of increasing the impact of automation failures, causing a commensurate decrease in L. Second, attempts to increase homogeneity may encounter resistance—ignoring a lesson learned from the transition from mainframes to client-server systems in the late 1980s, which was in large part driven by the desire of departments to have more control over their computing environments and hence a need for more diversity.

To reduce cost by reducing G, one approach is to adopt the concept of mass customization developed in the manufacturing industry (e.g., [8]). This means designing components and processes so as to facilitate customization. In terms of software distribution, this might mean developing re-usable components for software packages. Mass customization can also be achieved by having target systems that automatically discover their configuration parameters (e.g., from a registry at a well known address). This would mean that many differences between packages would be eliminated, reducing G and potentially leading to consolidation of package versions, also increasing N.

4   Related Work

The automation of IT operations has been a focus of attention for the last two decades [9], with on-going development of new technologies [4, 17, 1] and dozens of automation related products on the market [16]. More

recently, there has been interest in process automation through workflow based solutions [5, 15, 12]. However, none of these efforts address the question of when automation reduces cost. There has been considerable interest in manufacturing in business cases for automation [11, 2, 6], and even an occasional study that addresses automation of IT operations [10, 13]. However, these efforts only consider the automation infrastructure, not whether a particular process with a limited lifetime should be automated.

5 Conclusions

Recapping our position, we argue against the widely-held belief that automation always reduces the high costs of IT operations. Our argument rests on three pillars:

1. Introducing automation creates extra processes to deploy and maintain that automation, as we saw in comparing manual and automated software distribution processes.

2. Automation requires structured inputs (e.g., packages for a software distribution system) that introducing extra up-front (fixed) costs for design, implementation, and testing compared to manual processes. These fixed costs are a significant consideration in IT operations since many processes have a limited lifetime (e.g., a software package is installed on only a limited number of targets). Indeed, our studies of automated software distribution in several data centers found that 25% of the software packages were installed on fewer than 15 servers.

3. Detecting and removing errors from an automated process is considerably more complicated than for a manual process. Our software distribution data suggest that errors in automation can be frequent—19% of the requested installs failed in the data centers we studied.

Given these concerns, it becomes much less clear when automation should be applied. Indeed, in our model-driven analysis of software distribution in several large data centers, we found that 15–30% of automated software installs may have been less costly if performed Manually. Given that IT operations costs dominate IT spending today, it is essential that the kind of analysis we have demonstrated here become an integral part of the decision process for investing in and deploying IT automation. We encourage the research community to focus effort on developing tools and more sophisticated techniques for performing such analyses.

References

[1] G. Candea, E. Kiciman, S. Kawamoto, and A. Fox. Autonomous recovery in componentized internet applications. *Cluster Computing Journal*, 2004.

[2] T.J. Caporello. Staying ahead in manufacturing and technology-the development of an automation cost of ownership model and examples. *IEEE International Symposium on Semiconductor Manufacturing*, 1999.

[3] D. Cappuccio, B. Keyworth, and W. Kirwin. Total Cost of Ownership: The Impact of System Management Tools. Technical report, The Gartner Group, 2002.

[4] G. Kaiser, J. Parekh, P. Gross, and G. Valetto. Kinesthetics extreme: An external infrastructure for monitoring distributed legacy systems. In *Fifth Annual International Active Middleware Workshop*, 2003.

[5] A. Keller, J.L. Hellerstein, J.L. Wolf, K.-L. Wu, and V. Krishnan. The champs system: Change management with planning and scheduling. In *IEEE/IFIP Network Operations and Management*, April 2004.

[6] N.S. Markushevich, I.C. Herejk, and R.E. Nielsen. Function requirements and cost-benefit study for distribution automation at B.C. Hydro. *IEEE International Transactions on Power Systems*, 9(2):772–781, 1994.

[7] MicroData. The Hidden Cost of Your Network. Technical report, MicroData, 2002.

[8] J.H. Mikkola and T. Skjott-Larsen. Supply-chain integration: implications for mass customization, modularizaiton and postponement strategies. *Production Planning and Control*, 15(4):352–361, 2004.

[9] K.R. Milliken, A.V. Cruise, R.L. Ennis, A.J. Finkel, J.L. Hellerstein, D.J. Loeb, D.A. Klein, M.J. Masullo, H.M. Van Woerkom, and N.B. Waite. YES/MVS and the autonomation of operations for large computer complexes. *IBM Systems Journal*, 25(2), 1986.

[10] NetOpia. netoctopus: The comprehensive system administration solution. `http://www.netopia.com/software/pdf/net0-ROI.pdf`, 2005.

[11] C.A. Niznik. Cost-benefit analysis for local integrated facsimile/data/voice packet communication networks. *IEEE Transactions on Communications*, 30(1), January 1982.

[12] Peregrine. Service center. `http://www.peregrine.com/products/servicecenter.asp`, 2005.

[13] M. H. Sherwood-Smith. Can the benefits of integrated information systems (IIS) be costed. *Internation Conference on Information Technology in the Workplace*, pages 11–18, 1991.

[14] Serenity Systems. Managed Clinent Impact on the Cost of Computing. `http://www.serenity-systems.com`, 2005.

[15] G. Valetto and G. Kaiser. A case study in software adaptation. In *WOSS '02: Proceedings of the first workshop on Self-healing systems*, pages 73–78, 2002.

[16] ComputerWorld Staff Writer. E-business buyers' guide. In `www.computerworld.com`, 2005.

[17] S. A. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie. High speed and robust event correlation. *IEEE Communications Magazine*, 34(5):82–90, 1996.