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 methodology for analyzing the benefits and costs of automation, and hence for determining if automation will indeed reduce the cost of IT operations. Our analysis provides a quantitative framework that captures several traditional rules of thumb: 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 [8, 4, 16]. 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*. This message has appeal since automation provides a way to reduce labor costs and error rates as well as increase the uniformity with which IT operations are performed.

After working with corporate customers, service delivery personnel, and product development groups, we have come to question the widely held belief that automation of IT systems 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 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 increases other costs, such as those for maintaining the automation infrastructure, adapting inputs to structured formats required by automation, and handling automation failures. When these extra costs outweigh the benefits of automation, we have a situation described by human factors experts as an *irony of automation*—a case where automation intended to reduce cost has ironically ended up increasing it [1].

To prevent these ironies of automation, we must take a holistic view when adding automation to an IT system. This requires a technique for methodically exposing the hidden costs of automation, and an analysis that weighs these costs against the benefits of automation. The approach proposed in this paper is based on explicit representations of IT operational processes and the changes to those processes induced by automation. We illustrate our process-based approach using a running example of automated software distribution. We draw on data collected from several real data centers to help illuminate the impact of automation and the corresponding 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

We begin our discussion of the hidden costs of automation by laying out a methodical approach to exposing them. Throughout, 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 in-

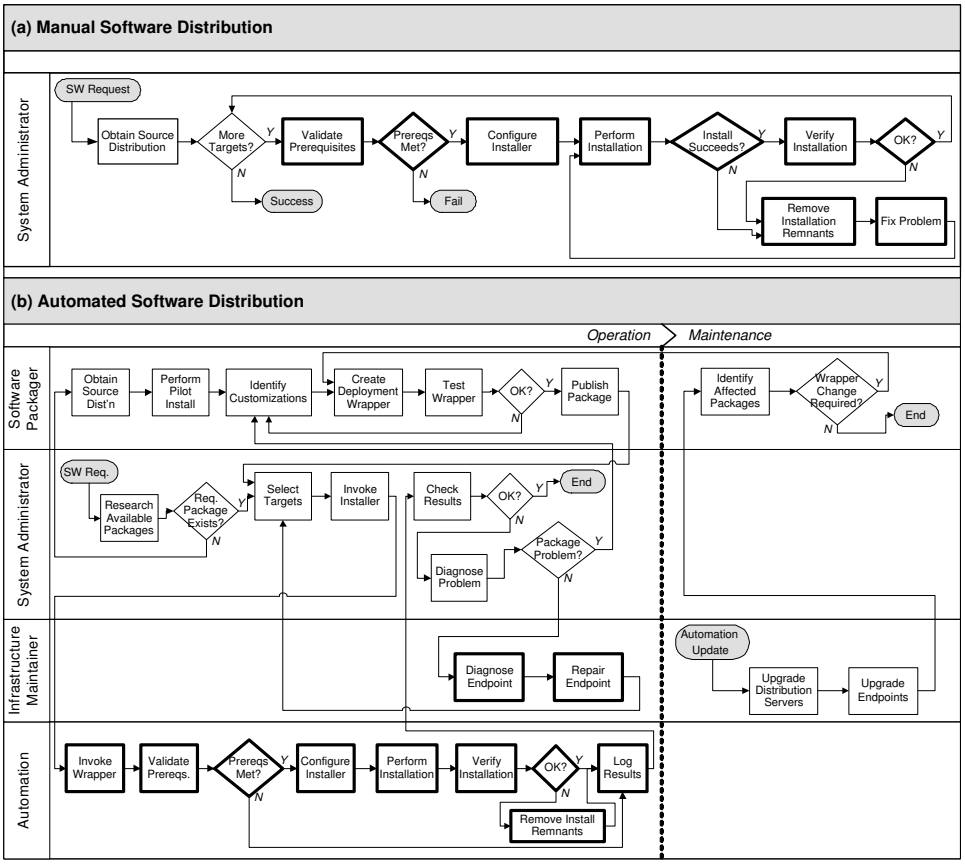

Figure 1: ***Manual and automated processes for software distribution.*** *Boxes with heavy lines indicate process steps that contribute to variable (per-target) costs, as described in Section 3.*

volves 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.

Our approach is based on the explicit representation of the *processes* followed by system administrators (SAs). These processes may be formal, *e.g.* derived from ITIL best practices [13], or informal, representing the ad-hoc methods used in practice. Regardless of their source, the first step is to document the processes as they exist before automation. Our approach accomplishes this with "swim-lane" diagrams—annotated flowcharts that allocate process activities across *roles* (represented as rows) and *phases* (represented as columns). 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.

Figure 1(a) shows the "swim-lane" representation for the manual version of our example software distribution process. In the data centers we studied, the SA responds to a request to distribute software as follows: (1) the SA obtains the necessary software resources; (2) for each server, the SA repeatedly does the following—(2a) checks prerequisites such as the operating system release level, memory requirements, and dependencies on other packages; (2b) configures the installer, which requires that the SA determine the values of various parameters such as the server's IP address and features to be installed; and (2c) 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 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 copied to what directories, pre-requisites, and the setting of configuration parameters).

Now suppose that we automate the process in Figure 1(a) so as to reduce the work done by the SA. That is, in the normal case, the SA selects a software package, and the software distribution infrastructure handles the other parts of the process flow in Figure 1(a). Have we simplified IT operations?

No. In fact, we may have made IT operations *more*

complicated. To understand why, we turn to our process-driven analysis, and update our process diagram with the changes introduced by the automation. In the software distribution case, the first update is simple: we move the automated parts of Figure 1(a) from the System Administrator role to a new Automation role. But that change is not the only impact of the automation. For one thing, 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. We use the term *software package* to refer to these structured inputs. These inputs are typically expressed in a formal structure, which means that their creation requires extra effort for package design, implementation, and testing. Last, when errors occur in the automated case, they happen on a much larger scale than for a manual approach, and hence additional processes and tools are required to recover from them.

These other impacts manifest as additional process changes, namely extra roles and extra operational processes to handle the additional tasks and activities induced by the automation. Figure 1(b) illustrates the end result for our software distribution example. 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 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**.

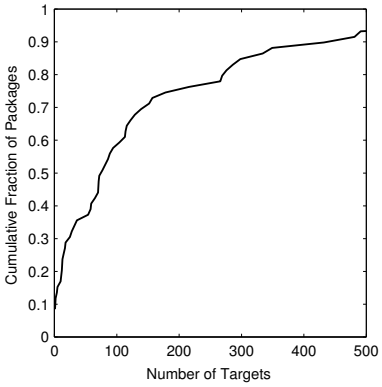

Figure 2: *Cumulative distribution of the number of targets (servers) on which a software package is installed over its lifetime in several data centers. A larger number of packages are installed on only a small number of targets.*

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., [10, 5]) has shown that rules (and hence processes) change frequently because of changes in data center policies and 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; it is only useful as long as that install and its target configuration remain current. The fixed cost of building a package must be amortized across the number of targets to which it is distributed over its lifetime. 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 over their lifetimes.

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. By considering the complete view of the automated processes in Figure 1(b), we see that more sophistication and people are required to address error recovery for automated soft-

ware distribution than for the manual process. Using the same data from which Figure 2 is extracted, we determined that 19% of the requested installs result in failure. 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. This back-of-the envelope analysis underscores the importance of considering the entire set of process changes that occur when automation is deployed, particularly the extra operational processes created to handle automation failures. It also suggests the need for a quantitative model to determine when to automate a process.

Motivated by our software distribution example, we have developed a simple version of such a model. 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

$$C^m = C_f^m + NC_v^m$$

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

$$C^a = C_f^m + C_f^a + NC_v^a$$

The costs can be obtained through billing records, as we have done at IBM. N depends on the packages being distributed and the configuration of potential targets.

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 $C^a < C^m$.

$$N > \frac{C_f^a}{C_v^m - C_v^a}.$$

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 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.

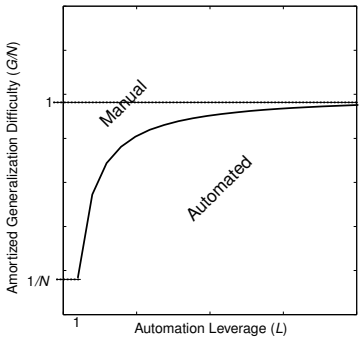

Figure 3: *Preference regions for automated and manual processes. Automated processes are preferred if there is a larger leverage for automation and/or if there is a smaller (amortized) difficulty of generalizing the manual procedure to an automated procedure (G/N).*

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 re-

gions 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 approaches 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, two issues arise. 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 greater diversity.

To reduce cost by reducing G, one approach is to adopt the concept of mass customization developed in the manufacturing industry (e.g., [9]). 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. It also implies improving the reusability of process components—for example by standardizing the manual steps used in software package installations—so that a given automation technology can be directly applied to a broader set of situations. This concept of mass-customizable automated process components represents an important area of future research.

Mass customization can also be improved at the system level 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 [10], with on-going development of new technologies [5, 19, 2] and dozens of automation related products on the market [18]. More recently, there has been interest in process automation through workflow based solutions [6, 17, 14]. 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 [12, 3, 7], and even an occasional study that addresses automation of IT operations [11, 15]. However, these efforts only consider the automation infrastructure, not whether a particular process with a limited lifetime should be automated.

5 Next Steps

One area of future work is to explore a broader range of IT processes so as to assess the generality of the automation analysis framework that we developed in the context of software distribution. Candidate processes to study include incident reporting and server configuration. The focus of these studies will be to assess (a) what automation is possible, (b) what additional processes are needed to support the automation, and (c) the fixed and variable costs associated with using automation on an on-going basis. Our current hypothesis for (b) is that additional processes are required for (1) preparing inputs, (2) invoking and monitoring the automation, (3) handling automation failures, and (4) maintaining the automation infrastructure. A particularly interesting direction will be to understand if there are any common patterns to the structure and cost of these additional processes across automation domains.

Thus far, we have discussed what automation should be done. Another consideration is the adoption of automation. Our belief is that SAs require a level of trust in the automation before the automation will be adopted. Just as with human relationships, trust is gained through a history of successful interactions. However, creating such a history is challenging because many of the technologies for IT automation are immature. As a result, care must be taken to provide incremental levels of automation that are relatively mature so that SA trust is obtained. One further consideration in gaining trust in automation is that automation cannot be a "black box" since gaining trust depends in part on SAs having a clear understanding of how the automation works.

The history of the automobile provides insight into the progression we expect for IT automation. In the early twentieth century, driving an automobile required considerable mechanical knowledge because of the need to make frequent repairs. However, today automobiles are sufficiently reliable so that most people only know that automobiles often need gasoline and occasionally need oil. For the automation of IT operation, we are at a stage similar to that of the early days of the automobile in that most computer users must also be system administrators (or have one close at hand). IT operations will have ma-

tured when operational details need not be surfaced to end users.

6 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 process-based 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] L. Bainbridge. The ironies of automation. In J. Rasmussen, K. Duncan, and J. Leplat, editors, *New Technology and Human Error*. Wiley, 1987.

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

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

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

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

[13] UK Office of Government Commerce. *Best Practice for Service Support*. IT Infrastructure Library Series. Stationery Office, 1st edition, 2000.

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

[15] 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.

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

[17] 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.

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

[19] 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.