RC23928 (W0604-003) April 3, 2006
Computer Science

IBM Research Report

Automating System Administration: Landscape, Approaches, and Costs

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

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

Automating System Administration: Landscape, Approaches, and Costs

Aaron B. Brown, Joseph L. Hellerstein, and Alexander Keller

IBM Corporation

Abstract

Automation is essential for efficient system administration. This chapter addresses the following questins: What to automate? How to automate? And, when to automate? The "what" question is addressed by studying what system administrators do and therefore where automation provides value. The "how" question is about technologies for automation. We discuss three approaches—rule-based systems, control theoretic approaches, and workflow-based automation. The "when" is ultimately a business question that should be based on a full understanding of the costs and benefits of automation. For this, we provide a framework for doing such an analysis and apply it to software distribution.

Key words: Automation, System Administration, Scenarios, Strategies, Process Models, ITIL, workflow construction, Cost-Benefit Analysis

1 Introduction

The cost of systems administration in Information Technology (IT) systems often exceeds the cost of hardware and software. Our belief is that automating system administration can reduce these costs and increase the business value provided by IT.

Making progress with automating system administration requires addressing three questions. What to automate? How should this automation be done? When does the automation provide business value?

What to automate is often answered in a simplistic way — everything! The problem here is that automation requires investment and inevitably causes some disruption in the "as-is" IT environment. As a result, it is important to target aspects of System Administration where automation provides the most

value. We believe that a process-based perspective provides the kind of broad context in which such judgments can be made.

How to automate system administration can be approached in many ways. Among these are rule-based techniques, control theory, and automated workflow construction. These approaches provide different kinds of benefits, and, in many ways, address different aspects of the automation challenge.

When to automate is ultimately a business decision that should be based on a full understanding of the costs and benefits. The traditional perspective has been that automation is always advantageous. However, it is important to look at the full costs imposed by automation. For example, automating software distribution requires that: (a) the distribution infrastructure be installed and maintained; (b) software packages be prepared in the format required by the distribution infrastructure; and (c) additional tools be provided to handle problems with packages that are deployed because of the large scale of the impact of these problems. While automation often provides a net benefit despite these costs, we have observed cases in which these costs exceed the benefits.

The remainder of this chapter provides more detail on the questions of what, how, and when to automate.

2 What System Administrators Do

We start our consideration of automating system administration by examining what system administrators do, as these tasks are the starting points for building automation.

System administrators are a central part of IT operations. Anyone who has owned a personal computer knows some elements of system administration. Software must be installed, patched, and upgraded. Important data must be backed up, and occasionally restored. And, when a problem occurs, time must be spent on problem isolation, identification, and resolution.

In many ways, system administrators perform the same activities as individuals. There are, however, some important differences. The first is scale. Administrators are typically responsible for tens to hundreds of machines. This scale means that management tools are essential to deal with repetitive tasks such as software installation.

A second important difference is that system administrators are usually responsible for mission critical machines and applications. Seconds of downtime at a financial institution can mean millions of dollars of lost revenue. Failures

in control systems for trains and aviation can cost lives. These considerations place tremendous emphasis on the speed and accuracy with which administrators perform their jobs.

In large data centers, system administration is typically structured by technology. Examples of these technology areas are servers, databases, storage, and networks. Administrators will typically train and develop specialized expertise in one technology area—for example a database administrator (DBA) becomes an expert in the details of database installation, tuning, configuration, diagnosis, and maintenance. However, the different technology areas often contain similar tasks, such as diagnosis and capacity planning, so the techniques developed in one area will often translate relatively easily to others. Even so, the tools used to accomplish these tasks in the separate areas remain different and specialized.

The remainder of this section is divided into four parts. The first two parts describe system administration in the words of experts. We focus on two technology areas—database administration and network administration. The third part of the section is a broader look at system administration based on a USENIX/SAGE survey. The section concludes with a discussion of best practices.

2.1 Database Administration

We begin our look at expert system administration in the area of database administration. The material in this section is based on a database administration course [8].

The job of a database administrator (DBA) is largely driven by the needs of the organization. But at the heart of this job is a focus on managing data integrity, access, performance, and security/privacy. Typical tasks performed by a DBA might include:

- designing and creating new databases
- configuring existing databases to optimize performance, for example by manipulating index structures
- managing database storage to ensure enough room for stored data
- diagnosing problems such as "stale" database data and deadlocks
- ensuring data is replicated to on-line or off-line backup nodes
- federating multiple independent database instances into a single virtual database
- interfacing with other administrators to troubleshoot problems that extend beyond the database (e.g., storage or networking issues)

- generating reports based on database contents and rolling up those reports across multiple database systems

The tools that a DBA uses to perform tasks like these vary greatly depending on the scale and needs of the organization. Small organizations and departments may use simple tools such as Paradox, Visual FoxPro, or Microsoft Access to maintain data. Larger organizations and governments with corporate-wide data requirements demand industrial-strength database tools such as IBM DB2, Oracle database, Microsoft SQL Server, Ingres, etc. And for organizations with multiple databases (often a situation that occurs after acquisitions, or when independent departments are transitioned to a centralized system), additional tools, often called "middleware", will be necessary to federate and integrate the existing databases across the corporation. Middleware poses its own administration considerations for installation, configuration, optimization, and maintenance.

In addition to the core tools—the database engine(s) and middleware—DBAs use many specialized administration tools. These may be scripts developed by the DBA for specific situations, or they may be vendor-supplied tools for monitoring, administration, and reporting.

As we look toward automation of a DBA's system administration responsibilities, we must recognize the challenges faced by a DBA. First is the integration of large numbers of tools. It should be clear from the discussion above that anything larger than a small department environment will involve multiple database engines, possibly middleware, and a mix of home-grown and vendor-supplied tools. Second is the need to "roll-up" data in large environments, e.g., integrating department reports into corporate-wide reports. Part of the challenge of the roll-up is the need for processes to scrub data to ensure correctness and consistency. The challenge of providing roll-ups creates a tension between the small scale systems with ease of entry and the large scale systems that provide robustness and scalability.

Finally, DBAs do not operate in a vacuum. There is considerable interaction with other administration "towers", such as network and storage management. The following from [8] provides an illustrative example:

> Unfortunately, one situation that can occur more often than planned, or more accurately, more than it should, is when the database does not update despite the careful steps taken or the time involved in trying to accomplish a successful update [...]
>
> The first step is to investigate the problem to learn the true "age" of the data. Next, the DBA would attempt to update a recent set of data to determine what may have occurred. It could have been a malfunction that day, or the evening the update was attempted. Assuming the situation does not

improve by this upload, the next step is to contact the network administrator about possible server malfunctions, changes in standard record settings, or other changes that might affect the upload of data.

Occasionally, the network server record lock settings were changed to "disallow" any upload to a network server over a certain limit [...] If the DBA is lucky, or has positive karma due for collection, all that may be required is for the network administrator to reset the record lock setting at a higher level. Updates can then be repeated and will result in current data within the business unit database for purposes of management reporting.

However, on a bad day, the DBA may learn from the network administrator that the server was or is malfunctioning, or worse, crashed at the precise time of the attempted update [...] In this situation the investigation must retrace the steps of data accumulation to determine the validity of the dataset for backup and experimental upload (network administrators at ringside) to watch for any type of malfunction.

In environments such as the foregoing, automation usually proceeds in an incremental manner, starting with special-purpose scripts that automate specific processes around the sets of existing tools, to generalizations of those scripts into new management tools, to rich automation frameworks that integrate tools to close the loop from detecting problems to reacting to them automatically.

2.2 Network Administration

Next, we look at another technology area for system administration: network administration. The material in this section is based on a course on network administration [36]. We have included quotes as appropriate.

Network administrators are responsible for the performance, reliability, and scalability of corporate networks. Achieving these goals requires a substantial understanding of the business for which these services are being delivered as well as the nature and trends in network technologies. In particular, in today's network-connected, web-facing world, network administrators have gone from supporting the back-office to enabling the online front-office, ensuring the network can deliver the performance and reliability to provide customer information, sales, support, and even B2B commerce online via the Internet. The network administrator's job has become a critical function in the revenue stream for many businesses.

Typical tasks performed by a network administrator might include:

- designing, deploying, and redesigning new networks and network interconnections

- deploying new devices onto a network, which requires a good understanding of the network configuration, the device requirements, and considerations for loads imposed by network traffic
- setting and enforcing security policies for network-connected elements
- monitoring network performance and reconfiguring network elements to improve performance
- managing network-sourced events
- tracking the configuration of a network and the inventory of devices attached to it
- detecting failures, diagnosing their root causes, and taking corrective actions such as reconfiguring the network around a failed component

Network administrators make use of a variety of tools to perform these tasks. Some are as simple as the TCP `ping`, `traceroute` and `netstat` commands, which provide simple diagnostics. Administrators responsible for large corporate networks frequently make use of network management systems with sophisticated capabilities for filtering, eventing, and visualization. Examples of such systems are Hewlett Packard's OpenView product and IBM's NetView product. Administrators of all networks will occasionally have to use low-level debugging tools such as packet sniffers and protocol decoders to solve compatibility and performance problems. No matter the scale of the network, tools are critical given the sizes of modern networks and the volumes of data that move across them. To quote from the advice for network administrators in [36], "A thorough inventory and knowledge of the tools at your disposal will make the difference between being productive and wasting time. A good network administrator will constantly seek out ways to perform daily tasks in a more productive way."

As we look toward automation of a network administrator's activities, there are two key aspects to consider. First is to provide more productive tools and to integrate existing tools. Here, automation must help administrators become more productive by absorbing the burden of monitoring, event management, and network performance optimization. Again this kind of automation progresses incrementally, starting with better scripts to integrate existing tools into situation-specific problem solvers, moving to integrated tools that, for example, merge monitoring with policy-driven response for specific situations, and culminating in integrated frameworks that close the loop and take over management of entire portions of a corporate network from the human administrator's hands.

The second key aspect of automating a network administrator's activities is to improve the task of problem diagnosis. Here, we refer to both proactive maintenance and reactive problem-solving. In the words of [36]:

> Resolving network, computer, and user related issues require the bulk of

> an administrator's time. Eventually this burden can be lessened through finding permanent resolutions to common problems and by taking opportunities to educate the end user. Some reoccurring tasks may be automated to provide the administrator with more time [...] However, insuring the proper operation of the network will preempt many problems before the users notice them.

Finally, it is important to point out that for any system administration role, automation can only go so far. There always is a human aspect of system administration that automation will not replace; in the case of network administration, again in the words of [36], this comes in many forms.

> Apart from the software and hardware, often the most difficult challenge for a network administrator is juggling the human aspects of the job. The desired result is always a productive network user, not necessarily just a working network. To the extent possible, the administrator should attempt to address each user's individual needs and preferences. This also includes dealing with the issues that arise out of supporting user skill levels ranging from beginner to knowledgeable. People can be the hardest and yet most rewarding part of the job.
>
> As problems are addressed, the solutions will be documented and the users updated. Keeping an open dialogue between the administrator and users is critical to efficiently resolving issues.

2.3 The Broader View

Now that we have looked at a few examples of system administration, next we move up from the details of individual perspectives to a survey of administrators conducted by SAGE, the Special Interest Group of the USENIX Association focusing on system administration. Other studies support these broad conclusions, such as [4].

The SAGE study [13] covered 128 respondants from 20 countries, with average experience of 7 years, 77% with college degrees, and another 20% having some college. The survey focused on server administration. Within this group, each administrator supported 500 users, 10 – 20 servers, and works in a group of 2 to 5 people. As reported in the survey, administrators have an "atypical day" at least once a week.

With this background, we use the survey results to characterize how administrators spend their time, with an eye to assessing the opportunity for automation. This is done along two different dimensions. The first is *what* is being administered.

- 20% miscellaneous
- 12% application software
- 12% email
- 9% operating systems
- 7% hardware
- 6% utilities
- 5% user environment

There is a large spread of adminstrative targets, and the reason for the large fraction of miscellaneous administration may well be due to the need to deal with several problems at once. These data suggest that to improve a system administrator's job, automation must be broad. Automation that targets a single domain such as hardware will be useful, but will not solve the end-to-end problems that system administrators typically face. Instead, we should consider automation that takes an end-to-end approach, cutting across domains as needed to solve the kinds of problems that occur together. We will address this topic in Section 4 in our discussion of process-based automation.

Another way to view administrators' time is to divide it by the kind of activity. The survey results report:

- 11% meetings
- 11% communicating
- 9% configuring
- 8% installing
- 8% "doing"
- 7% answering questions
- 7% debugging

We see that at least 29% of the administrator's job involves working with others. Sometimes, this is working with colleagues to solve a problem. But there is also substantial time in reporting to management on progress in resolving an urgent matter and dealing with dissatisfied users. About 32% of the administrator's job involves direct operations on the IT environment. This is the most obvious target for automation, although it is clear that automation will need to address some of the communication issues as well (e.g., via better reporting).

The survey goes on to report some other facts of interest. For example, fire fighting only takes 5% of the time. And, there is little time spent on scheduling, planning, and designing. The latter, it is noted, may well be reflected in the time spent on meetings and communicating. We can learn from these facts that automation must address the entire lifecycle of administration, not just the (admittedly intense) periods of high-pressure problem-solving.

The high pressure and broad demands of system administration may raise

serious questions about job satisfaction. Yet, 99% of those surveyed said they would do it again.

2.4 The Promise of Best Practices and Automation

One insight from the foregoing discussion is that today systems administration is more of a craft than a well-desciplined profession. Part of the reason for this is that rapid changes in IT make it difficult to have re-usable best practices, at least for many technical details.

There is a body of best practices for service delivery that is gaining increasing acceptance, especially in Europe. Referred to as the Information Technology Infrastructure Library (ITIL), these best practices encompass configuration, incident, problem management, change management, and many other processes that are central to systems administration [18]. Unfortunately, ITIL provides only high level guidance. For example, the ITIL process for change management has activities for "authorizing change", "assign priority", and "schedule change". There are few specifics about the criteria used for making decisions, the information needed to apply these criteria, or the tools required to collect this information.

That said, ITIL provides an important perspective on systems administration that helps lay out an automation roadmap. The key element of the ITIL perspective is its focus on *process*. ITIL describes end-to-end activities that cut across the traditional system administration disciplines, and suggests how different "towers" like network, storage, and application/database management come together to design infrastructure, optimize behavior, or solve cross-cutting problems faced by users. ITIL thus provides a philosophy that can guide us to the ultimate goals of automation, where end-to-end, closed-loop activities are subsumed entirely by automation, the system administrators can step out, and thus the costs of delivering IT services are reduced significantly.

We believe that installations will move through several phases in their quest to reduce the burden on systems administrators and hence the cost of delivering IT services. In our view, these steps are:

(1) **Environment-independent automation:** execution of repetitive tasks without system-dependent data within one tower, for example static scripts or response-file-driven installations.
(2) **Environment-dependent automation:** taking actions based on configuration, events, and other factors that require collecting data from systems. This level of automation is often called "closed-loop" automation, though here it is still restricted to one discipline or tower.
(3) **Process-based automation:** Automation of interrelated activities in

best practices for IT service delivery. Initially, this tends to be open-loop automation that mimics acitivities done by administrators such as the steps taken in a software install. Later on, automation is extended to closed-loop control of systems such as problem detection and resolution.

(4) **Business level automation:** Automation of IT systems based on business policies, priorities, and processes. This is an extension of process-based automation where the automation is aware of the business-level impact of various IT activities and how those IT activities fit into higher-level business processes (for example, insurance claim processing). It extends the closed-loop automation described in (3) to incorporate business insights.

Business-level automation is a lofty goal, but the state of the art in 2006 is still far from reaching it outside very narrowly-constrained domains. And, at the other extreme, environment-independent automation is already well-established through ad-hoc mechanisms like scripts and response files. Thus in the remainder of our discussion we will focus on how to achieve environment-dependent and process-based automation. Section 3 describes a strategy for reaching process-based automation, and follows that with a description of some automation techniques that allow for environment-dependent automation and provide a stepping stone to higher levels of automation.

3 How to Automate

This section addresses the question of how to automate system administration tasks.

3.1 Automation Strategies

We start our discussion of automating IT service delivery by outlining a best-practice approach to process-based automation (cf. [7]). We have developed this approach based on our experiences with automating change management, along with insight we have distilled from interactions and engagements with service delivery personnel. It provides a roadmap for achieving process-level automation.

Our approach comprises six steps for transforming existing IT service delivery processes with automation or for introducing new automated process into an IT environment. The first step is to

(1) identify best practice processes for the domain to be automated.

To identify these best practices we turn to the IT Infrastructure Library (ITIL), a widely used process-based approach to IT service management. ITIL comprises several disciplines such as infrastructure management, application management, service support and delivery. The ITIL Service Support discipline [18] defines the Service Desk as the focal point for interactions between a service provider and its customers. To be effective, the Service Desk needs to be closely tied into roughly a dozen IT support processes that address the lifecycle of a service. Some examples of IT service support processes for which ITIL provides best practices are: Configuration Management, Change Management, Release Management, Incident Management, Problem Management, Service Level Management, and Capacity Management.

ITIL provides a set of process domains and best practices within each domain, but it does not provide guidance as to which domain should be an organization's initial target for automation. Typically this choice will be governed by the cost of existing activities.

After identifying the best practices, the next step is to

(2) establish the scope of applicability for the automation.

Most ITIL best practices cover a broad range of activities. As such, they are difficult to automate completely. To bound the scope of automation, it is best to target a particular subdomain (e.g., technology area). For example, Change Management applies changes in database systems, storage systems, and operating systems.

We expect that Change Management will be an early focus of automation. It is our further expectation that initial efforts to automate Change Management will concentrate on high frequency requests, such as security patches. Success here will provide a proof point and template for automating other areas of Change Management. And, once Change Management is automated to an acceptable degree, then other best practices can be automated as well, such as Configuration Management. As with Change Management, we expect that the automation of Configuration Management will be approached incrementally. This might be structured in terms of configuration of servers, software licenses, documents, networks, storage, and various other components.

The next step in our roadmap follows the ITIL-based approach:

(3) identify delegation opportunities.

Each ITIL best practice defines (explicitly or implicitly) a process flow consisting of multiple activities linked in a workflow. Some of these activities will be amenable to automation, such as deploying a change in Change Management. These activities can be *delegated* to an automated system or tool. Other ac-

tivities will not be easy to automate, such as obtaining change approvals from Change Advisory Boards. Thus, an analysis is needed to determine what can be automated and at what cost. Sometimes, automation drives changes in IT proceses. For example, if an automated Change Management system is trusted enough, change approvals might be handled by the Change Management System automatically.

The benefit of explicitly considering delegation is that it brings the rigor of the best-practice process framework to the task of scoping the automation effort. The best practice defines the set of needed functionality, and the delegation analysis explicitly surfaces the decision of whether each piece of functionality is better handled manually or by automation. Using this framework helps prevent situations like the one discussed later in Section 4 of this chapter, where the cost of an automation process outweighs its benefits in certain situations.

With the delegated activities identified, the fourth step in our automation approach is to

(4) identify links between delegated activities and external activities, processes, and data sources.

These links define the control and data interfaces to the automation. They may also induce new requirements on data types/formats and APIs for external tools. An example in Change Management is the use of configuration data. If Change Management is automated but Configuration Management remains manual, the Configuration Management Database (CMDB) may need to be enhanced with additional APIs to allow for programmatic access from the automated Change Management activities. Moreover, automation often creates **induced processes**, additional activities that are included in support of the automation. Examples of induced processes in automated software distribution are the processes for maintaining the software distribution infrastructure, error recovery, and preparation of the software packages.

The latter point motivates the following step:

(5) Identify, design, and document induced processes needed to interface with or maintain the automation.

This step surfaces many implications and costs of automation and provides a way to do cost/benefit tradeoffs for proposed automation.

The last step in our automation approach is to

(6) implement automation for the process flow and the delegated activities.

Implementing the process flow is best done using a workflow system to au-

tomatically coordinate the best-practice process's activities and the flow of information between them. Using a workflow system brings the additional advantage that it can easily integrate automated and manual activities.

For the delegated activities, additional automation implementation considerations are required. This is a non-trivial task that draws on the information gleaned in the earlier steps. It uses the best practice identified in (1) and the scoping in (2) to define the activities' functionality. The delegation choices in (3) scope the implementation work, while the interfaces and links to induced process identified in (4) and (5) define needed APIs, connections with external tools and data sources, and user interfaces. Finally, the actual work of the activity needs to be implemented directly, either in new code or by using existing tools.

In some cases, step (6) may also involve a recursive application of the entire methodology described here. This is typically the case when a delegated ITIL activity involves a complex flow of work that amounts to a process in its own right. In these cases, that activity sub-process may need to be decomposed into a set of subactivities; scoped as in steps (2) and (3), linked with external entities as in (4), and may induce extra sub-process as in (5). The sub-process may in turn also need to be implemented in a workflow engine, albeit at a lower level than the top-level best practice process flow.

3.2 Automation Technologies

This section introduces the most common technologies for automation. Considered here are rules, feedback control techniques, and workflows technology.

3.2.1 Rule-based Techniques

Rule-based systems provide a condition-action approach to automation. An example of a rule is

- Rule: If there is a `SlowResponse` event from system ?S1 at location ?L1 within 1 minute of another `SlowResponse` event from system ?S2 $\neq$?S1 at location ?L1 and there is no `SlowResponse` event from system S3 at location ?L2 $\neq$?L1, then alert the Network Manager for location ?L1.

In general, rules are if-then statements. The if-portion, or left-hand side, describes a situation. The then-portion, or right hand side, specifies actions to take when the situation arises.

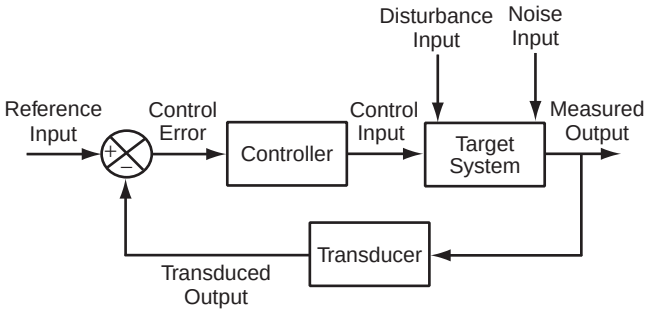

Fig. 1. *Block diagram of a feedback control system. The reference input is the desired value of the system's measured output. The controller adjusts the setting of control input to the target system so that its measured output is equal to the reference input. The transducer represents effects such as units conversions and delays.*

As noted in [35], there is a great deal of work in using rule-based techniques for root cause analysis (RCA). [27] describes a rule-based expert system for automating the operation of an IBM mainframe, including diagnosing various kinds of problems. [15] describes algorithms for identifying causes of event storms. [19] describe the application of an expert system shell for telecommunication network alarm correlation.

The simplicity of rule-based systems offers an excellent starting point for automation. However, there are many shortcomings. Among these are:

(1) A pure rule-based system requires detailed descriptions of many situations. These descriptions are time-consuming to construct and expensive to maintain.
(2) Rule-based systems scale poorly because of potentially complex interactions between rules. Such interactions make it difficult to debug large scale rule-based systems, and create great challenges when adding new capabilities to such systems.
(3) Not all automation is easily represented as rules. Indeed, step-by-step procedures are more naturally expressed as workflows.

There are many approaches to circumventing these shortcomings. One of the most prominent examples is [22], which describes a code-book–based algorithm for RCA. The practical application of this approach relies on explicit representations of device behaviors and the use of configuration information. This has been quite effective for certain classes of problems.

3.2.2 Control Theoretic Approaches

Control theory provides a formal framework and a set of analysis/design techniques for regulating and optimizing systems. Over the last sixty years, control theory has developed a fairly simple reference architecture. This architecture

is about manipulating a target system to achieve a desired objective. The component that manipulates the target system is the controller.

As discussed in [10] and depicted in Figure 1, the essential elements of feedback control system are:

- target system, which is the computing system to be controlled.
- control input, which is a parameter that affects the behavior of the target system and can be adjusted dynamically (such as the `MaxClients` parameter in the Apache HTTP Server).
- measured output, which is a measurable characteristic of the target system such as CPU utilization and response time.
- disturbance input, which is any change that affects the way in which the control input influences the measured output of the target system (e.g., running a virus scan or a backup).
- noise input, which is any effect that changes the measured output produced by the target system. This is also called sensor noise or measurement noise.
- reference input, which is the desired value of the measured output (or transformations of them), such as CPU utilization should be 66%. Sometimes, the reference input is referred to as desired output or the setpoint.
- transducer, which transforms the measured output so that it can be compared with the reference input (e.g., smoothing stochastics of the output).
- control error, which is the difference between the reference input and the measured output (which may include noise and/or may pass through a transducer).
- controller, which determines the setting of the control input needed to achieve the reference input. The controller computes values of the control input based on current and past values of control error.

The foregoing is best understood in the context of a specific system. Consider a cluster of three Apache Web Servers. The Administrator may want these systems to run at no greater than 66% utilization so that if any one of them fails, the other two can immediately absorb the entire load. Here, the measured output is CPU utilization. The control input is the maximum number of connections that the server permits as specified by the `MaxClients` parameter. This parameter can be manipulated to adjust CPU utilization. Examples of disturbances are changes in arrival rates and shifts in the type of requests (e.g., from static to dynamic pages).

Many researchers have applied control theory to computing systems. In data networks, there has been considerable interest in applying control theory to problems of flow control, such as [21] which develops the concept of a Rate Allocating Server that regulates the flow of packets through queues. Others have applied control theory to short-term rate variations in TCP (e.g., [24]) and some have considered stochastic control [2]. More recently, there have

been detailed models of TCP developed in continuous time (using fluid flow approximations) that have produced interesting insights into the operation of buffer management schemes in routers (see [17], [16]). Control theory has also been applied to middleware to provide service differentiation and regulation of resource utilizations as well as optimization of service level objectives. Examples of service differentiation include enforcing relative delays [1], preferential caching of data [25], and limiting the impact of administrative utilities on production work [28]. Examples of regulating resource utilizations include a mixture of queueing and control theory used to regulate the Apache HTTP Server [33], regulation of the IBM Lotus Domino Server [29], and multiple-input, multiple-output control of the Apache HTTP Server (e.g., simultaneous regulation of CPU and memory resources) [9]. Examples of optimizing service level objectives include minimizing response times of the Apache Web Server [11] and balancing the load to optimize database memory management [12].

All of these examples illustrate situations where control theory-based automation was able to replace or augment manual system administration activities, particularly in the performance and resource management domains.

3.2.3 Automated workflow construction

In section 2.4, we have observed that each ITIL best practice defines (explicitly or implicitly) a process flow consisting of multiple activities linked in a workflow. Some of these activities will be amenable to automation – such as deploying a change in Change Management – whereas others will not (such as obtaining change approvals from Change Advisory Boards). On a technical level, recent efforts aim at introducing extensions for people facing activities into workflow languages [23]. The goal is to facilitate the seamless interaction between the automated activities of an IT service management process and the ones that are carried out by humans. Here, we summarize work in [5] and [20] on automating system administration using workflow.

In order to provide a foundation for process-based automation, it is important to identify the control and data interfaces between delegated activities and external activities, processes, and data sources. The automation is provided by lower-level automation workflows, which consist of atomic administration activities, such as installing a database management system, or configuring a data source in a Web Application Server. The key question is to what extent the automation workflows can be automatically generated from domain-specific knowledge, instead of being manually created and maintained. For example, many automation workflows consist of activities whose execution depends on the current state of a distributed system. These activities are often not specified in advance. Rather, they are merely implied. For example, applications must be recompiled if they use a database table whose schema is

to change. Such implicit changes are a result of various kinds of relationships, such as service dependencies and the sharing of the service provider's resources among different customers. Dependencies express compatibility requirements between the various components of which a distributed system is composed. Such requirements typically comprise software pre-requisites (components that must be present on the system for an installation to succeed), co-requisites (components that must be jointly installed) as well as ex-requisites (components that must be removed prior to installing a new component). In addition, version compatibility constraints and memory/disk space requirements need to be taken into account. All of this information is typically captured during the development and build stages of components, either by the developer, or by appropriate tooling. Dependency models, which can be specified e.g., as *Installable Unit Deployment Descriptors* [14] or *System Description Models* [26], are a key mechanism to capture this knowledge and make it available to the tools that manage the lifecycle of a distributed system.

An example of a consumer of dependency information is the *Change Management System*, whose task is to orchestrate and coordinate the deployment, installation and configuration of a distributed system. Upon receiving a request for change (RFC) from an administrator, the change management system generates a *Change Plan*. A change plan describes the partial order in which tasks need to be carried out to transition a system from a workable state into another workable state. In order to achieve this, it contains information about:

- The type of change to be carried out, e.g., install, update, configure, uninstall,
- the roles and names of the components that are subject to a change (either directly specified in the RFC, or determined by the change management system),
- the precedence and location constraints that may exist between tasks, based on component dependency information,
- an estimate of how long every task is likely to take, based on the results of previous deployments. This is needed to estimate the impact of a change in terms of downtime and monetary losses.

The change management system exploits dependency information in order to determine whether tasks required for a change must be carried out sequentially, or whether some of them can be parallelized. The existence of a dependency between two components – each representing a managed resource – in a dependency graph indicates that a precedence/location constraint must be addressed. If a precedence constraint exists between two tasks in a workflow (e.g., X must be installed before Y), they need to be carried out sequentially. This is typically indicated by the presence of a link; any task may have zero or more incoming and outgoing links. If two tasks share the same predeces-

sor and no precedence constraints exist between them, they can be executed concurrently. Typically, tasks and their constraints are grouped on a per-host basis. Grouping on a per-host basis is important because the actual deployment could happen either push-based (triggered by the provisioning system) or pull-based (deployment is initiated by the target systems). An additional advantage of grouping activities on a per-host basis is that one can carry out changes in parallel (by observing the presence of cross-system links) if they happen on different systems. Note that parallelism on a single host system is difficult to exploit with current operating systems as few multithreaded installers exist today. In addition, some operating systems require exclusive access to shared libraries during installation.

Different types of changes require different traversals through the dependency models: If a request for a new installation of a component is received, one needs to determine which components must already be present before a new component can be installed. On the other hand, a request for an update, or an uninstall of an component leads to a query to determine the components that will be impacted by the change.

Precedence constraints represent the order in which provisioning activities need to be carried out. Some of these constraints are implicit (e.g., by means of a containment hierarchy expressed as 'HasComponent' relationships between components), whereas others (typically resulting from communication dependencies such as 'uses') require an explicit representation (e.g., the fact that a database client needs to be present on a system whenever a database server located on a remote host needs to be accessed).

3.2.3.1 Example for Software Change Management The example is based on the scenario of installing and configuring a multi-machine deployment of a J2EE based enterprise application and its supporting middleware software (including IBM's HTTP Server, WebSphere Application Server, WebSphere MQ embedded messaging, DB2 UDB database and DB2 runtime client). The specific application we use is taken from the SPECjAppServer2004 enterprise application performance benchmark [34]. It is a complex, multi-tiered on-line e-Commerce application that emulates an automobile manufacturing company and its associated dealerships. SPECjAppServer2004 comprises typical manufacturing, supply chain and inventory applications that are implemented with web, EJB, messaging, and database tiers. We jointly refer to the SPECjAppServer2004 enterprise application, its data, and the underlying middleware as the SPECjAppServer2004 *solution*. The SPECjAppServer2004 solution spans an environment consisting of two systems: one system hosts the application server along with the SPECjAppServer2004 J2EE application, whereas the second system runs the DBMS that hosts the various types of SPECjAppServer2004 data (catalog, orders, pricing, user data, etc.). One of

the many challenges in provisioning such a solution consists in determining the proper order in which its components need to be deployed, installed, started and configured. For example, 'HostedBy' dependencies between the components 'SPECjAppServer2004 J2EE Application' and 'WebSphere Application Server v5.1' (WAS) state that the latter acts as a hosting environment for the former. This indicates that all the WAS components need to be operating before one can deploy the J2EE application into them.

A provisioning system supports the administrator in deploying, installing and configuring systems and applications. As mentioned above, a change plan is a procedural description of activities, each of which maps to an operation that the provisioning system exposes, preferably by means of a set of interfaces specified using the Web Services Description Language (WSDL). As these interfaces are known well in advance before a change plan is created, they can be referenced by a change plan. Every operation has a set of input parameters, for example, an operation to install a given software component on a target system requires references to the software and to the target system as input parameters. Some of the parameters may be input by a user when the change plan is executed (e.g., the hostname of the target system that will become a database server) and need to be forwarded to several activities in the change plan, whereas others are produced by prior activities.

3.2.3.2 Executing the generated Change Plan Upon receiving a newly submitted change request, the change management system needs to determine on which resources and at what time the change will be carried out. The change management system first inspects the resource pools of the provisioning system to determine which target systems are best assigned to the change by taking into account which operating system they run, what their system architecture is, and what the cost of assigning them to a change request is. Based on this information, the change management system creates a change plan, which may be composed of already existing change plans that reside in a change plan repository. Once the change plan is created, it is submitted to the workflow engine of the provisioning system. The provisioning system maps the actions defined in the change plan to operations that are understood by the target systems. Its object-oriented data model is a hierarchy of *logical devices* that correspond to the various types of managed resources (e.g., software, storage, servers, clusters, routers or switches). The methods of these types correspond to *Logical Device Operations (LDOs)* that are exposed as WSDL interfaces, which allows their inclusion in the change plan. *Automation packages* are product-specific implementations of logical devices: e.g., an automation package for the DB2 DBMS would provide scripts that implement the software.install, software.start, software.stop, etc. LDOs. An automation package consists of a set of Jython scripts, each of which implements an LDO. Every script can further embed a combination of PERL, Expect, Windows

scripting host or bash shell scripts that are executed on the remote target systems. We note that the composition pattern applies not only to workflows, but occurs in various places within the provisioning system itself to decouple a change plan from the specifics of the target systems.

The workflow engine inputs the change plan and starts each provisioning operation by directly invoking the LDOs of the provisioning system. These invocations are performed either in parallel or sequentially, according to the instructions in the change plan. A major advantage of using an embedded workflow engine is the fact that it automatically performs state-checking, i.e., it determines whether all conditions are met to move from one activity in a workflow to the next. Consequently, there is no need for additional program logic in the change plan to perform such checks. Status information is used by the workflow engine to check if the workflow constraints defined in the plan (such as deadlines) are met and to inform the change management system whether the roll-out of changes runs according to the schedule defined in the change plan.

3.2.3.3 Recursive Application of the Automation Pattern The automation described in this section essentially makes use of three levels of workflow engines: (1) top-level coordination of the Change Management workflow; (2) implementation of the generated change plan; and (3) the provisioning system where the LDOs are implemented by miniature-workflows within their defined scripts. This use of multiple levels of workflow engine illustrates a particular pattern of composition that we expect to be common in automation of best-practice IT service management processes, and recalls the discussion earlier in Section 3.1 of recursive application of the automation approach.

In particular, the delegated Change Management activity of Distribute and Install Changes involves a complex flow of work in its own right — documented in the change plan. We can see that the approach to automating the construction of the change plan follows the same pattern we used to automate change management itself, albeit at a lower level. For example, the creation of the change workflow is a lower-level analogue to using ITIL best practices to identify the Change Management process activities. The execution of change plan tasks by the provisioning system represents delegation of those tasks to that provisioning system. The provisioning system uses external interfaces and structured inputs and APIs to automate those tasks—drawing on information from the CMDB to determine available resources and invoking lower-level operations (automation packages) to effect changes in the actual IT environment. In this latter step, we again see the need to provide such resource information and control APIs in the structured, machine-readable formats needed to enable automation. The entire pattern repeats again at a lower level within the provisioning system itself, where the automation packages for detailed soft-

ware and hardware products represent best practice operations with delegated functionality and external interfaces for data and control.

One of the key benefits of this type of recursive composition of our automation approach is that it generates reusable automation assets. Namely, at each level of automation, a set of automated delegated activities is created: automated ITIL activities at the top (such as Assess Change), automated change management activities in the middle (such as Install the DB2 Database), and automated software lifecycle activities at the bottom (such as Start DB2 Control Process). While created in the context of change management, it is possible that many of these activities (particularly lower-level ones) could be reused in other automation contexts. For example, many of the same lower-level activities created here could be used for performance management in an on-demand environment to enable creating, activating, and deactivating additional database or middleware instances. It is our hope that application of this automation pattern at multiple levels will reduce the long-term costs of creating system management automation, as repeated application will build up a library of reusable automation components that can be composed together to simplify future automation efforts.

3.2.3.4 Challenges in workflow construction An important question addresses the problem whether the change management system or the provisioning system should perform error recovery for the complete change plan in case an activity fails during workflow execution or runs behind schedule. One possible strategy is not to perform error recovery or dealing with schedule overruns within the change plan itself and instead delegate this decision instead to the change management system. The reason for doing so is that in service provider environments, resources are often shared among different customers, and a change of a customer's hosted application may affect the quality of service another customer receives. Before rolling out a change for a given customer, a service provider needs to trade off the additional benefit he receives from this customer against a potential monetary loss due to the fact that an SLA with another customer may be violated because of the change. The scheduling of change plans, a core change management system function, is the result of solving an optimization problem that carefully balances the benefits it receives from servicing one customer's change request against the losses it incurs from not being able to service other customers. In an on-demand environment, the cost/profit situation may change very rapidly as many change plans are concurrently executed at any given point in time. In some cases, it may be more advantageous to carry on with a change despite its delay, whereas in other cases, aborting a change and instead servicing another newly submitted request that is more profitable may lead to a better global optimum. This big picture, however, is only available to the change management system.

Another important requirement for provisioning composed applications is the dynamic aggregation of already existing and tested change plans. For example, in the case of SPECjAppServer2004 and its underlying middleware, change plans for provisioning some of its components (such as WebSphere Application Server or the DB2 DBMS) may already exist. Those workflows need to be retrieved from a workflow repository and aggregated in a new workflow for which the activities for the remaining components have been generated. The challenge consists in automatically annotating the generated workflows with metadata so that they can be uniquely identified and evaluated with respect to their suitability for the specific request for change. The design of efficient query and retrieval mechanisms for workflows is an important prerequisite for the reuse of change plans.

4 When to Automate

Now that we have identified the opportunities for automation and discussed various approaches, we now step back and ask: when is it appropriate to deploy systems management automation? The answer is, surprisingly, not as straightforward as one might expect—in particular, automation may not always be the right choice even for costly system administration tasks.

This is not a new conclusion, though it is rarely discussed in the context of automating system administration. Indeed, in other fields practitioners have long recognized that automation can be a double-edged sword. For example, early work with aircraft autopilots illustrated the dangers of imperfect automation, where pilots would lose situational awareness and fail to respond correctly when the primitive autopilot reached the edge of its envelope and disengaged [31], causing more dangerous situations than when the autopilot was not present. There are many other well-documented cases where either failures or unexpected behavior of industrial automation caused significant problems, including in such major incidents as Three Mile Island [32, 30].

How are we to avoid, or at least minimize, these problems in automation of system administration? One solution is to make sure that the full consequences of automation are understood before deploying it, changing the decision of "when to automate?" from a simple answer of "always!" to a more considered analytic process.

In the next several subsections, we use techniques developed in [6] to consider what this analysis might look like, not to dissuade practitioners from designing and deploying automation, but to provide the full context needed to ensure that automation lives up to its promises.

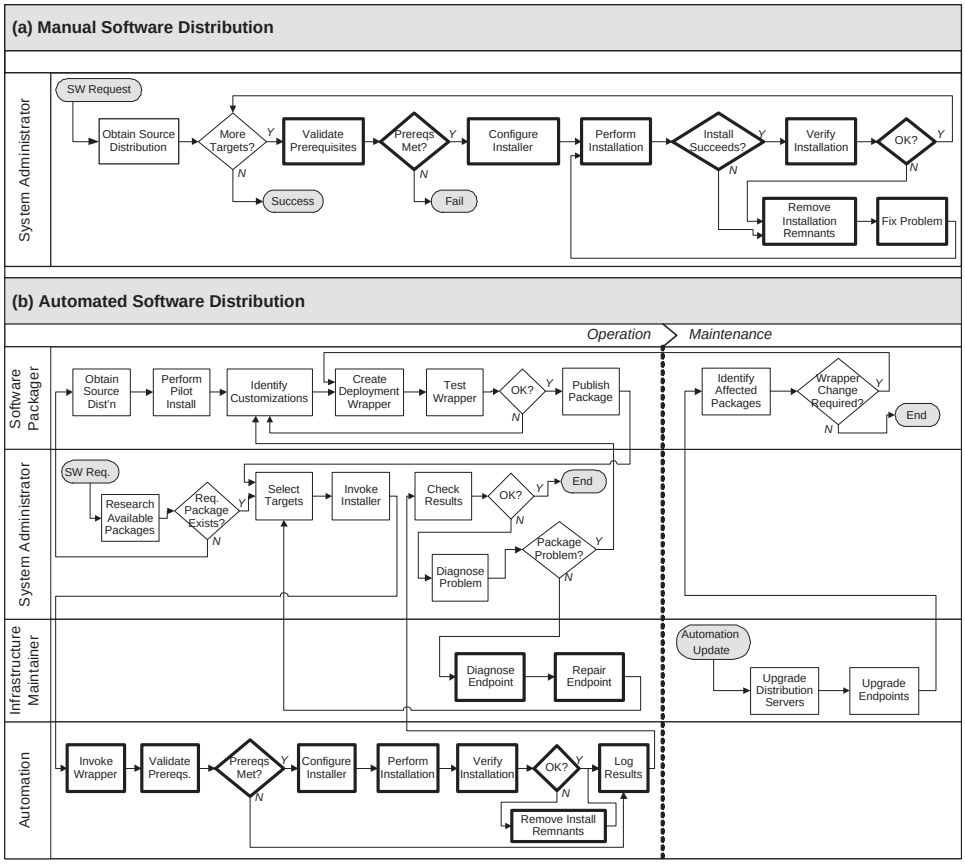

Fig. 2. ***Manual and automated processes for software distribution.*** *Boxes with heavy lines indicate process steps that contribute to variable (per-target) costs, as described in Section 4.2.*

4.1 Cost-Benefit Analysis of Automation

We begin by exploring the potential hidden costs of automation. While automation obviously can reduce cost by removing the need for manual systems management, it can also induce additional costs that may offset or even negate the savings that arise from the transfer of manual work to automated mechanisms. If we take a process-based view, as we have throughout this paper, we see that automation transforms a manual process, largely to reduce manual work, but also to introduce new process steps and roles needed to maintain, monitor, and augment the automation itself.

Figure 2 illustrates the impact of automation on an example system management process, drawn from a real production data center environment. The process here is software distribution to server machines, a critical part of operating a data center. 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. We represent the process via

"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 2(a) shows the "swim-lane" representation for the manual version of our example software distribution process, and Figure 2(b) shows the same process once automated software distribution has been introduced.

Notice that, while key parts of the process have moved from a manual role to an automated role, there are additional implications. For one thing, the automation infrastructure is another software system that must itself be installed and maintained, creating initial transition costs as well as periodic costs for update and maintenance. (For simplicity, in the figure we have assumed 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.

We have to recognize as well that the effects of induced processes—for error recovery, maintenance, and input creation/structuring—are potentially mitigated by the probability and frequency of their execution. For example, if automation failures are extremely rare, then having complex and costly recovery procedures may not be a problem. Furthermore, the entire automation framework has a certain lifetime dictated by its flexibility and generalization capability. If this lifetime is long, the costs to create and transition to the automated infrastructure may not be an issue. But even taking this into account, it is apparent from inspection that the collection of processes in Figure 2(b) is much more complicated than the single process in Figure 2(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?

The answer comes through a cost-benefit analysis, where the costs of induced process are weighted by their frequency and balanced against the benefits of automation. Our argument in this section is that such analyses are an essential part of a considered automation decision. That is, when considering

a manual task for automation, the benefits of automating that task (reduced manual effort, skill level, and error probability) need to be weighed against the costs identified above (extra work from induced process, new roles and skills needed to maintain the automation, and potential consequences from automation failure).

In particular, this analysis needs to consider both the changes in tasks (tasks replaced by automation and tasks induced by it) and the changes in roles and required human skills. The latter consideration is important because automation tends to create induced tasks requiring more skill than the tasks it eliminates. For example, the tasks induced by automation failures tend to require considerable problem-solving ability as well as a deep understanding of the architecture and operation of the automation technology. If an IT installation deploys automation without having an experienced administrator who can perform these failure-recovery tasks, the installation is at risk if a failure occurs. The fact that automation often increases the skill level required for operation is an example of an *irony of automation* [3]. The irony is that while automation is intended to reduce costs overall, there are some kinds of cost that increase. Many ironies of automation have been identified in industrial contexts such as plant automation and infrastructure monitoring [32]. Clearly, the ironies of automation must be taken into account when doing a cost-benefit analysis.

4.2 Example Analysis: Software Distribution for a Datacenter

To illustrate the issues associated with automation, we return to the example of software distribution. To perform our cost-benefit analysis, we start by identifying the fixed and variable costs for the process activities in Figure 2. The activities represented as boxes with heavy outlines represent variable-cost activities, as they are performed once for each machine in the data center. The other activities are fixed-cost in that they are performed once per software package.

A key concept used in our analysis is that of the **lifetime of a software package**. By the lifetime of a package, we mean the time from when the package is created to when it is retired or made obsolete by a new package. We measure this in terms of the number of target machines to which the package is distributed.

For the benefits of automation to outweigh the cost of automation, the variable costs of automation must be lower than the variable costs of the manual process, *and* the fixed cost of building a package must be amortized across the number of targets to which it is distributed over its lifetime. Using data

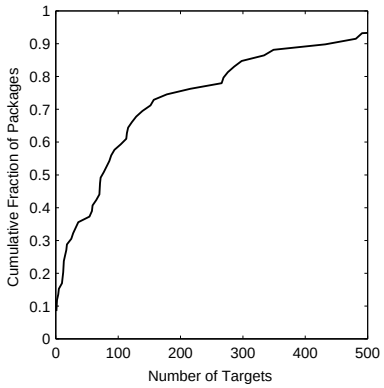

Fig. 3. *Cumulative distribution of the number of targets (servers) on which a software package is installed over its lifetime in several data centers. A larger number of packages are installed on only a small number of targets.*

from a several computer installation, Figure 3 plots the cumulative fraction of packages in terms of the lifetimes (in units of number of targets to which the package is distributed). 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.

Next, we look at the possibility of automation failure. By considering the complete view of the automated processes in Figure 2(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 3 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. Drawing on additional data from IBM internal studies of software distribution and netting out the analysis (the details can be found in [6]), we find that for complex software packages, there should be approximately 5 to 20 targets for automated software distribution to be cost effective. In terms of the data in Figure 3, these numbers mean that from 15% to 30% of the installs in the data centers we examined should not have been automated from a cost perspective.

The lesson here is clear, and goes back to the discussion at the start of this section. Even in the domain of system administration, automation can be a double-edged sword. While in these datacenters automation of software distribution provided a net cost benefit for 70 – 85% of installs, it increased costs for the remaining 15 – 30%. In this case the choice was made to deploy the automation, with the assessment that the benefit to the large fraction of installs outweighed the cost to the smaller fraction. The insight of a cost-benefit

analysis allows such decisions to be made with eyes open, and full awareness of the tradeoffs being made.

4.3 Additional Concerns: Adoption and Trust

Automation is only useful if it is used. Even the best automation—carefully designed to maximize benefit and minimize induced process and cost—can lack traction in the field. Often these situations occur when the automation fails to gain the trust of the system management staff tasked with deploying and maintaining it.

So what can automation designers do to help their automation gain trust? First, they must recognize that automation is a disruptive force for IT system managers. Automation changes the way system administrators do their jobs. It also creates uncertainity in terms of the future of the job itself. And since no automation is perfect, there is concern about the extent to which automation can be trusted to operate correctly.

Thus adoption of automation is unlikely without a history of successful use by administrators. This observation has been proven in practice many times. Useful scripts, languages, and new open-source administration tools tend to gain adoption via grass-roots movements where a few brave early adoptors build credibility for the automation technology. But, not all new automation can afford to generate a community-wide grass-roots movement behind it. And in these cases we are left with a a kind of circular logic—we cannot gain adoption without successful adoptions!

In these situations, the process-based view can help provide a transition path. From a process perspective, the transition to automation can be seen as an incremental delegation of manual process steps to an automated system. And by casting the automation into the same terms as the previously-manual process steps (for example, by reusing the same work products and tracking metrics), the automation can provide visibility into its inner workings that assuages a system administrator's distrust. Thus the same work that it takes to do the cost/benefit analysis for the first cut at an automation decision can be leveraged to plan out an automation deployment strategy that will realize the automation's full benefits.

We believe that the process-based perspective described above can provide the basis for making a business case for automation. With a full understanding of the benefits of automation as well as all the cost factors described above, we finally have a framework to answer the question of when to automate. To determine the answer we compute or estimate the benefits of the proposed automation as well as the full spectrum of costs, and compute the net benefit.

In essence, we boil the automation decision down to the bottom line, just as in a traditional business case.

The process for building an automation business case consists of 5 steps:

(1) Identify the benefit of the automation in terms of reduction in "core" manual activity, lower frequency of error (quantified in terms of the expected cost of errors, e.g., in terms of business lost during downtime), and increased accuracy of operations.
(2) Identify and elucidate the induced processes that result from delegation of the manual activity to automation, as in the software distribution example above. "Swim-lane" diagrams provide a useful construct for structuring this analysis.
(3) Identify or estimate the probability or frequency of the induced processes. For example, in software distribution we estimate how often new packages will be deployed or new versions released, as those events require running the packaging process. Often estimates can be made based on existing runtime data.
(4) Assess the possible impact of the "automation irony". Will the induced processes require additional skilled resources, or will they change the existing human roles enough that new training or hiring will be needed?
(5) Collate the collected data to sum up the potential benefits and costs, and determine whether the automation results in net benefit or net cost.

These steps are not necessarily easy to follow, and in fact in most cases a rough, order-of-magnitude analysis will have to suffice. Alternately, a sensitivity analysis can be performed. For example, if the probability of automation failure is unknown, a sensitivity analysis can be used to determine the failure rate where costs break even. Say this is determined to be one failure per week. If the automation is thought to be stable enough to survive multiple weeks between failures, then the go-ahead decision can be given.

Furthermore, even a rough business-case analysis can provide significant insight into the impact of automation, and going through the exercise will reveal a lot about the ultimate utility of the automation. The level of needed precision will also be determined by the magnitude of the automation decision: a complete analysis is almost certainly unnecessary when considering automation in the form of a few scripts, but it is mandatory when deploying automation to a datacenter with thousands of machines.

5 Conclusions

This chapter addresses the automation of system administration, a problem that is addressed as a set of three interrelated questions: what to automate, how to automate, and when to automate.

We address the "what" question by studying what system administrators do and therefore where automation provides value. An observation here is that tasks such as configuration and installation consume a substantial fraction of the time of systems adminisrators, approximately 20%. This is fortunate since it seems likely that it is technically feasible to increase the degree of automation of these activities. Unfortunately, meetings and other forms of communication consume almost of third of the time of systems administrators. Here, there seems to be much less opportunity to realize benefits from automation.

The "how" question is about technologies for automation. We discuss three approaches—rule-based systems, control theoretic approaches, and automated workflow construction. All three have been used in practice. Rules provide great flexibility in building automation, but the complexity of this approach becomes problematic as the scope of automation increases. Control theory provides a strong theoretical foundation for certain classes of automation, but it is not a universal solution. Workflow has appeal because its procedural structure is a natural way for humans to transate their activities into automation.

The "when" question is ultimately a business question that should be based on a full understanding of the costs and benefits. The traditional perspective has been that automation is always advantageous. However, it is important to look at the full costs imposed by automation. For example, automating software distribution requires that: (a) the distribution infrastructure be installed and maintained; (b) software packages be prepared in the format required by the distribution infrastructure; and (c) additional tools be provided to handle problems with packages that are deployed because of the large scale of the impact of these problems. While automation often provides a net benefit despite these costs, we have observed cases in which these costs exceed the benefits.

As the scale of systems adminstration increases and new technologies for automation are developed, systems administrators will have even greater challenges in automating their activities.

References

[1] Tarek F. Abdelzaher and Nina Bhatti. Adaptive content delivery for Web server QoS. In *International Workshop on Quality of Service*, pages 1563–1577, London, UK, June 1999.

[2] E. Altman, T. Basar, and R. Srikant. Congestion control as a stochastic control problem with action delays. *Automatica*, 35:1936–1950, 1999.

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

[4] Rob Barrett, Paul P. Maglio, Eser Kandogan, and John Bailey. Usable autonomic computing systems: The system administrators' perspective. *Advanced Engineering Infomatics*, 19(3):213–221, July 2006.

[5] Aaron Brown, Alexankder Keller, and Joseph L Hellerstein. A model of configuration complexity and its application to a change management system. In *9th International IFIP/IEEE Symposium on Integrated Management (IM 2005)*, pages 531–644. IEEE Press, May 2005.

[6] Aaron B. Brown and Joseph L. Hellerstein. Reducing the Cost of IT Operations—Is Automation Always the Answer? In *Tenth Workshop on Hot Topics in Operating Systems (HotOS-X)*, Santa Fe, NM, 2005.

[7] Aaron B. Brown and Alexander Keller. A Best Practice Approach for Automating IT Management Processes. In *2006 IEEE/IFIP Network Operations & Management Symposium (NOMS 2006)*, Vancouver, BC, Canada, April 2006.

[8] Peg Byers. *Database Administrator: Day in the Life*. Thomson Course Technology. http://www.course.com/careers/dayinthelife/dba_jobdesc.cfm.

[9] Yixin Diao, Neha Gandhi, Joseph L Hellerstein, Sujay Parekh, and Dawn Tilbury. Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache Web server. In *IEEE/IFIP Network Operations & Management Symposium (NOMS 2002)*, pages 219–234, April 2002.

[10] Yixin Diao, Rean Griffith, Joseph L. Hellerstein, Gail Kaiser, Sujay Parekh, and Dan Phung. A control theory foundation for self-managing systems. *Journal on Selected Areas of Communications*, 23(12), 2005.

[11] Yixin Diao, Joseph L Hellerstein, and Sujay Parekh. Optimizing quality of service using fuzzy control. In *IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 2002)*, pages 42–53, 2002.

[12] Yixin Diao, Joseph L. Hellerstein, Adam Storm, Maheswaran Surendra, Sam Lightstone, Sujay Parekh, and Christian Garcia-Arellano. Using MIMO Linear Control for Load Balancing in Computing Systems. In *American Control Conference*, pages 2045–2050, June 2004.

[13] Barbara Dijker. A day in the life of systems administrators. http://www.sage.org/field/ditl.pdf.

[14] M. Vitaletti (Editor). *Installable Unit Deployment Descrip-*

tor Specification, Version 1.0. W3C Member Submission, IBM Corp., ZeroG Software, InstallShield Corp., Novell,, July 2004. http://www.w3.org/Submission/2004/SUBM-InstallableUnit-DD-20040712/.

[15] Alan Finkel, Keith Houck, and A Bouloutas. An alarm correlation system for heterogeneous networks. *Network Management And Control*, 2, 1995.

[16] C. V. Hollot, V. Misra, D. Towsley, and W. B. Gong. A control theoretic analysis of RED. In *Proceedings of IEEE INFOCOM '01*, pages 1510–1519, Anchorage, Alaska, April 2001.

[17] C. V. Hollot, V. Misra, D. Towsley, and W. B. Gong. On designing improved controllers for AQM routers supporting TCP flows. In *Proceedings of IEEE INFOCOM '01*, pages 1726–1734, Anchorage, Alaska, April 2001.

[18] IT Infrastructure Library. *ITIL Service Support, version 2.3.* Office of Government Commerce, June 2000.

[19] G. Jakobson, R. Weihmayer, and M. Weissman. A domain-oriented expert system shell for telecommunications network alarm correlation. *Network Management And Control*, 2, 1995.

[20] Alexander Keller and Remi Badonnel. Automating the Provisioning of Application Services with the BPEL4WS Workflow Language. In *15th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 2004)*, pages 15–27, Davis, CA, USA, November 2004. Springer Verlag.

[21] Srinivasan Keshav. A control-theoretic approach to flow control. In *Proceedings of ACM SIGCOMM '91*, pages 3–15, September 1991.

[22] S. Kliger, S. Yemini, Y. Yemini, D. Ohlse, and S. Stolfo. A coding approach to event correlation. In *Fourth International Symposium on Integrated Network Management*, Santa Barbara, CA, USA, 1995.

[23] M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, and D. Roller. Business process choreography in WebSphere: Combining the power of BPEL and J2EE. *IBM Systems Journal*, 43(2), 2004.

[24] K. Li, M. H. Shor, J. Walpole, C. Pu, and D. C. Steere. Modeling the effect of short-term rate variations on tcp-friendly congestion control behavior. In *Proceedings of the American Control Conference*, pages 3006–3012, 2001.

[25] Ying Lu, Avneesh Saxena, and Tarek F. Abdelzaher. Differentiated caching services: A control-theoretic approach. In *International Conference on Distributed Computing Systems*, pages 615–624, April 2001.

[26] Microsoft. Overview of the system description model. http://msdn2.microsoft.com/en-us/library/ms181772.aspx.

[27] 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 automation of operations for large computer complexes. *IBM Systems Journal*, 25(2):159–180, 1986.

[28] S. Parekh, K. Rose, J. L. Hellerstein, S. Lightstone, M. Huras, and

V. Chang. Managing the performance impact of administrative utilities. In *14th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 2003)*, pages 130–142, 2003.

[29] Sujay Parekh, Neha Gandhi, Joseph Hellerstein, Dawn Tilbury, Joseph Bigus, and T. S. Jayram. Using control theory to achieve service level objectives in performance management. *Real-time Systems Journal*, 23:127–141, 2002.

[30] Charles Perrow. *Normal Accidents: Living with High-Risk Technologies*. Perseus Books, 1990.

[31] J. Rasmussen and W. Rouse, editors. *Human Detection and Diagnosis of System Failures: Proceedings of the NATO Symposium on Human Detection and Diagnosis of System Failures*. Plenum Press, New York, 1981.

[32] James Reason. *Human Error*. Cambridge University Press, 1990.

[33] L. Sha, X. Liu, Y. Lu, and T. Abdelzaher. Queueing model based network server performance control. In *IEEE RealTime Systems Symposium*, pages 81–90, 2002.

[34] SPEC Consortium. *SPECjAppServer2004 Design Document, Version 1.01*, January 2005. http://www.specbench.org/osg/jAppServer2004/docs/DesignDocument.html.

[35] David Thoenen, Jim Riosa, and Joseph L Hellerstein. Event relationship networks: A framework for action oriented analysis in event management. In *International Symposium on Integrated Network Management*, 2001.

[36] Tony Woodall. *Network Administrator: Day in the Life*. Thomson Course Technology. http://www.course.com/careers/dayinthelife/networkadmin_jobdesc.cfm.