SCTE CABLE-TEC EXPO FAST FORWARD 2021
VIRTUAL EXPERIENCE
OCTOBER 11-14

2021 Fall Technical Forum
SCTE • NCTA • CABLELABS

Software Reliability Engineering

Scaling the Cloud with Automation

A Technical Paper prepared for SCTE by

Brian Gray
Sr. Manager, Software Engineering
Comcast Cable
1800 Arch St., Philadelphia, PA 19103
267.634.5540
Brian_gray@comcast.com

Sriram Ramakrishnan, Principal Architect, Comcast Cable

Fei Wan, Sr. Principal Architect, Comcast Cable

Table of Contents

Title **Page Number**

List of Figures

Title **Page Number**

1. Introduction

Operations teams have long functioned under a primary mandate of assuring customers the most solid and uninterrupted experience possible. The recent advent of software reliability engineering (SRE) introduced engineers to the formalized automation of toil, leaving more time for creative problem solving of service interruptions. However, twin issues remain: how to automate a virtualized cloud environment in practice, and how to measure and prioritize repeated tasks to be automated for the greatest impact. In this paper, we offer a case study of the evolution of a complex cloud infrastructure from a state of manual deployment, scaling, failover, and upgrading, to one of push-button control and automated self-management. Driving this evolution is a pair of mathematical tools developed in Comcast's Core Application Platforms (CAP) group that use the financial concepts of "net present value"/NPV and "internal rate of return"/IRR to organize and value automation opportunities simply and objectively.

2. A Look Inside the Elements Deployment Architecture (A Starting Point)

Elements is the internal name of an asynchronous, fast, flexible push notification service which supports both individual and fan-out notifications across such diverse network destinations as outbound web hooks and mobile push notifications. The service is built as a publisher/subscriber (pub-sub) service, in which the senders of the messages, also known as 'Publishers/Producers' are decoupled from the receivers of the messages, called the 'Subscribers/Consumers'.

High Level Architecture

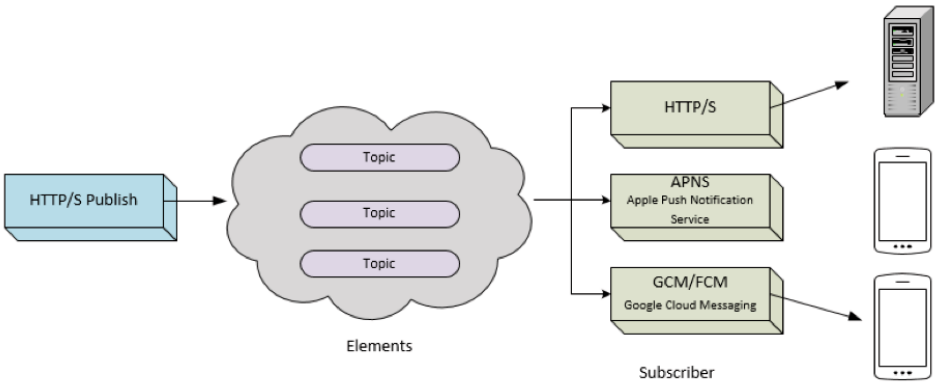

Figure 1 – Elements Architecture Layers

Complicating the equation is the complex requirement to operate across many layers of Comcast's internal network – or the "plant" – to reach destinations such as an X1 set-top box with notifications. Because we need deep access to underlying plant details, we lose out on the option of hosting this particular system in a public cloud like AWS with convenient, high-level facilities to assist our operations. Such a thing would technically be possible in a split-architecture design, using public cloud for pub-sub management, configuration, and global logic, and DirectConnect-ing into a private space that

hosts intelligent routing gateways. That solution, however, introduces many more points of failure, some of which actually lose the ability to report back that a failure happened. To compensate, you would then need **more** complexity to implement a run-time approach on a distributed trace algorithm.

2.1. Challenges

The infrastructure for the solution we maintain therefore runs entirely in our legacy private cloud environment and has many components, including Mesosphere DC/OS for container orchestration, Consul for service discovery, Couchbase for database persistence, and HAProxy for load balancing. Besides these, the system integrates with external Comcast systems for authentication, authorization, secrets management, and observability. As you can see given the complexity of this system, we see immense leverage from following the best practices for SRE. Unfortunately, when we started, we didn't have any such best practices. We were more operations-focused, which is reactive and time intensive, and not following many SRE guidelines.

To give an example, let's suppose we find out that HAProxy CPU usage is ridiculously high and that it's affecting the performance of our application. The extant tools we have available are Ansible – a simple IT automation facility – configured via "playbooks", and the HAProxy configuration schema. One option, then, is to check the config and verify if the value for nbproc is set > 1, where nbproc is an HAProxy configuration key that allows the proxy to spin up multiple processes. To change this config, since it's an emergency, we would create a one-time playbook with this HAProxy config and push the changes to our proxy servers. Once the change has been pushed, we would do a manual rolling restart of our proxy servers.

Obviously, there are multiple things wrong here. This change leads to inconsistent configuration, having no testing or automation. Also, this one-time run causes changes not to be source-controlled in GitHub, team-members might not be aware of this change, and there is no source-of-truth for the configuration. Beyond all these issues, time is critical: time we did spend on manual changes or figuring out who made the change. For all teams, but especially small teams like ours, time is the most valuable resource, and we really need to perform efficiently.

Another instance is a routine scaling of the database processing instances in our Couchbase infrastructure. If we wanted to increase the size of our cluster, we used Terraform (TF) to create the virtual machines (VMs) from the legacy cloud console and performed the configuration for adding this new Couchbase node to the cluster. Obviously, many things are wrong here, as well: No automation, lots of opportunities for failures, and no consistent way to scale the infrastructure.

In summary, these are the challenges we faced, which ultimately affected us as a team, and applications we maintain:

- Lack of automation
- Lack of Testing
- Inconsistent Configuration
- One-Off runs with Ansible
- No continuous integration & continuous deployment (CI/CD)

3. The Elements End State

3.1. Improved Configuration Management

The Elements end state continues to leverage Ansible as the configuration management tool. It's a useful tool that allows the declarative specification of your Infrastructure as Code (IaC), and having Ansible enables us to make changes to the configuration in a more consistent fashion. But we still run into the issue of one-time runs. Teams were making changes to the infrastructure by making one-time runs through Ansible. There are couple of issues with this approach:

- Changes are not shared with the other team members, and others might not know what has changed in the infrastructure.
- There is no single source of truth.

We really wanted to avoid this scenario and the only way to achieve that was to ensure teams follow the guidelines in making Ansible changes:

- All Ansible playbook changes needs to be in source control.
- Changes to playbooks must be done through Pull Requests (PRs) and can be merged only once review is completed by peers.

The obvious downside is changes might take time to be deployed, and reviews might take longer. We tried to reduce review times by pairing once a week (preferably after scrum) and approving the PR's. Remember, this process takes time and discipline among team members. But, in the long term, this helped us to have a predictable and consistent infrastructure that's easier to maintain.

3.2. Infrastructure Testing

Consider our scenario where we need to scale our Couchbase cluster. We don't have any consistent way to test our infrastructure, and we need one to test our Terraform, Packer and Ansible playbooks. Terratest is a Golang library that provides patterns and functions for testing the infrastructure built through Terraform. We baked Couchbase and all other standard tools into an image, which we created using Packer. Terratest can test the build of this image as well. Once the image has been built, Terraform will use those image IDs to build the Couchbase infrastructure. Ansible can be used to configure additional changes. Inspec is used to Test Ansible playbooks. Inspec is an open-source auditing and automated testing framework to describe and test for regulatory concerns, recommendations, or requirements. See Figure 1 – Elements Architecture Layers

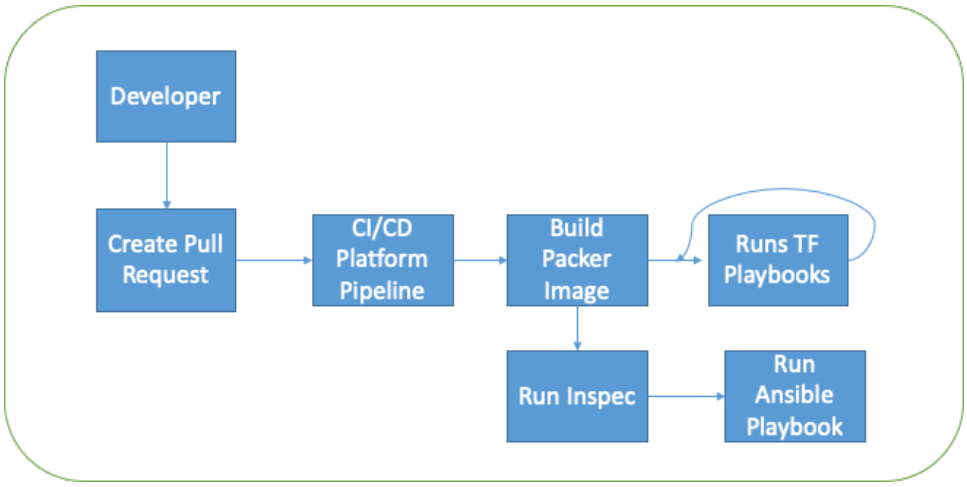

Figure 2 – PR -> Pipeline -> Playbook Process

Developing Terraform tests using Terratest requires Golang skills. This is not an insurmountable barrier, but some learning is needed. Inspec is more human-readable and might be easier for teams to learn.

Software infrastructure testing takes process, organization, and discipline, but will pay-off in long-term.

Just having testing is not enough, however; we needed an automated way to trigger these tests. That's where we decided to follow CI/CD guidelines for our infrastructure playbooks.

3.3. Continuous Integration and Deployment

This is where we really dig into the application of software development best practices to infrastructure code. These follow a couple of simple guidelines:

- All infrastructure code, including Terraform, Packer and Ansible playbooks, is in our source-control GitHub Enterprise repository.
- Any changes to the playbooks go through a PR process and, we have branch protection enabled to avoid direct merging into main. All PRs require at least 1 approving review.

It's good practice to have custom, application-specific templates for creating issues and PRs. For every PR that gets created, we have our CI pipeline that validates the syntactical correctness of the playbook, validates the format (using Terraform fmt for Terraform playbooks and Ansible lint for Ansible playbooks), and runs the test. We run automated tests in an environment that's totally separate from production for Terraform playbooks. We create all the resources (servers, load balancers, machine images) using namespaces with a unique name, to ensure that we don't accidentally overwrite any "production" resources in that environment and accidentally clash with other tests running in parallel. We ensure the tests always clean up after themselves, so we don't leave a series of zombie resources lying around untracked. Most times, we can anticipate the time taken for our tests and will stay within the default limits. But on occasion we might have to increase those timeouts.

Once PR builds are successful and review is complete, PRs can be merged into the main branch. In our case of scaling the Couchbase cluster, it's as simple as modifying the Terraform playbook to increase the number of instances. All changes get propagated or deployed to a lower environment automatically as they merge to main. But we don't do automatic deployment to production environments. We follow a

release-based, gated model to ensure we validate the infrastructure job was successful before we proceed to production.

4. Lessons Learned from the End State and the Path Forward

Configuration management, automated testing, and continuous integration are the key building blocks of a SRE approach to software reliability. The operations tasks required to ensure optimal uptime, performance, scalability, redundancy, and observability can overwhelm any team, but also can often be reduced to logical procedures: the very things where programming shines as a solution!

By leveraging a team's abilities as programmers, they can analyze a process (for example adding nodes to a database cluster), reduce that process to a task in the form of code, and have a pipeline trigger that task either in response to identified and alerted criteria, or else on demand, when a human merges a PR that changes the configuration. Thus do we address the primary culprit costing the team time and resources: toil.

5. SRE, and the Plight Thereof

As an SRE team, we spend a great deal of our capacity on automation of toil (in a rather circular definition, toil can be thought of as work that can be automated). You could say it's this very thing that most distinguishes an SRE team from a DevOps team. We see an opportunity, where we're doing work by hand, over and over, and look for ways to write code to do that work for us. Then we iterate to find the next most insistent itch to scratch.

This kind of ad-hoc, subjective process works well enough and can be great for morale. I mean, who wouldn't want to throw themself into automating away their personal bugaboo? So, let's assume we allow engineers to retain that flexibility. What else can we do to optimize our selection and evaluation of automation opportunities? Turns out we can get some help with tools invented for use in evaluating financial investments. Not only do these tools assist us in decision-making, but they can also act as indicators to report the real leading and lagging value of the SRE team to its parent organization.

6. The Financial Side

You know that thing you're an expert in? And then they make a movie about that thing, and you complain about all the ways they get it wrong? That's what this section will read like to financial experts. The situation here is simply that an engineering team with a need to formalize and quantify the priority of the automation work described above found a useful tool for doing exactly that, so – disclaimer provided – let's just dive in without checking the depth…

6.1. What Is Net Present Value?

Put simply, Net Present Value (NPV) is the amount a potential investment is worth today, including all outgoing costs, all incoming returns, and discounting each number to account for the time value of money.

What is the time value of money then? It's a formalization of the concept that money now is worth more than the same amount of money later. Sure, $100 is literally worth more now than $100 will be in ten years, because of inflation. But in addition to that, $100 in 2021 dollars is worth more now than $100 in 2021 dollars will be in 2031, in part because the ten years in the middle give you more options for what to do with the money.

To calculate the NPV of a proposal, you first need to decide what rate to assign to this time value. This is called the discount rate. When a person wins the lottery and is given the option of taking $X now or $Y/year for 30 years, the "now" value is calculated by applying the discount rate – for Powerball, 4% – to each payment's amounts and summing them to yield an NPV.

How is the discount rate figured? The simplest way is to determine the best low-risk investment available as an alternative to whatever else you may be thinking of doing with the money. For regular people, maybe this is a high-yield money market or bond fund. For a business that tracks metrics, it could be the return on marginal advertising dollars or scaling up workforce to accommodate more service delivery. Something you have control over and could throw money at. For the purposes of this discussion, we're conflating discount rate and "cost of capital" here. For a full treatment of the difference between them, review Cost of Capital vs. Discount Rate: What's the Difference?, but very briefly and inexactly, we can imagine that cost of capital is the real-world stuff above (advertising ROI, etc.), and discount rate is what it's called when it assumes the form of a number used in a mathematical formula.

Anyway, each time money changes hands, the present value of that transaction is calculated as:

$$\frac{R_t}{(1+i)^t}$$

Figure 3 – Present Value of a Single Event

…in which R_t is the cash flow (positive or negative) at the time of the event, i is the assigned discount rate, and t is the amount of time in the future the event takes place. To get the NPV of an entire series of events including the initial expenditure, we simply sum up all discrete events:

$$\mathrm{NPV}(i, N) = \sum_{t=0}^{N} \frac{R_t}{(1+i)^t}$$

Figure 4 – Net Present Value of a Series of Events

This result is a scalar value, in dollars, representing the magnitude of value that the project will net in excess of the discount rate over its lifetime. In other words, how much better or worse this investment is than going the risk-free route.

In the case of the lottery? Look into long-term equity or debt investments and see if you can find anything that beats 4%. It is for this reason that winners are advised to take the lump sum and invest initially in an index fund while figuring out better options. Historically, the average of the market as a whole has generated ~10% returns over the long term, minus inflation. The lump sum is far greater than the true NPV of the long-term payments because the discount rate applied to the winnings is so low.

6.2. What Is Internal Rate of Return?

Internal rate of return (IRR) is a fascinating inversion of NPV, which manages to factor out the discount rate. Thus, you're not concerning yourself with inflation, the risk-free investment alternatives, or cost of capital, only the factors internal to the investment. It does this by taking the NPV formula, setting the answer to 0, and solving for the discount rate:

$$\mathrm{NPV} = \sum_{n=0}^{N} \frac{C_n}{(1+r)^n} = 0$$

Figure 5 – Internal Rate of Return Formula

The mechanics of how to solve this are not at all easy or straightforward, so we'll gloss over them in favor of just using built-in Excel formulae. For a treatment of numerical methods to solve for IRR, start here.

The result is the discount rate at which this investment would yield no NPV. How is this useful? By itself, it is not. That is to say, in order to make use of the discount rate in an NPV calculation, it needs to have a real-world investment to compare against or you're just guessing. On the other hand, when you compare the IRR to other potential investments, it makes for a wonderful way to rank them by pure return percentage internal to the investment itself. So, to start, you can say, "Who cares what my cost of capital is? Investment #5 shows the highest IRR of all my options."

When combined with cost of capital though, you can first prune out those non-starters whose IRRs do not exceed it.

6.3. Which One Is Better?

The attempt to answer this question is effectively an attempt to decrease the number of tools you have at your disposal. It can be comforting to have and know your hammer and treat everything like a nail, but you become more effective when you learn about screwdrivers as well. As for IRR and NPV, they both have their place because they both describe very different viewpoints on your investment options:

- IRR gives you the rate of return but says nothing about the scale of the investment.
- NPV gives you the total value of the investment but gives no perspective on how quickly you recoup your initial layout.

An easy example is a pair of investments, one of which gives you an IRR of 25% over 2 years, returning $50,000 NPV. Another sustains a 20% IRR over 10 years and yields $200,000 NPV. The first option is a higher rate of return and gives control of your original investment back sooner, allowing reinvestment in year 3. The other locks in a $200,000 NPV investment and allows you to be hands off for 10 years. Are you sure you're going to have a >20% IRR option 2 years from now? What would a likely 10-year portfolio starting with the 25% investment look like, and would it compare favorably or unfavorably with a static 20%?

So, it pays to calculate both and use your own judgement to decide which investments to fund, depending on your risk acceptance, your demand for flexibility, and your list of known options.

7. Factoring out Dollars

Now that we've covered the financials, let's throw away the dollar signs. These calculations are great mathematical tools to leverage, but we're not here to talk about money. We're here to talk about time, so what happens when we change all our units to units of time?

I don't know if you noticed, but the IRR calculation placed $0 on the left, canceling out the R_t (expressed in dollars) on the right. The resulting percentage rate is unitless, indicating that we could use anything as the original units: potatoes, rainbows… or time. If you invest 40 hours of work automating a task, so that now there are 4 hours of work per month you don't have to do anymore, the formula works exactly as if you were investing money.

IRR doesn't care about units. Time works identically to money and gives you useful, sortable results.

Bam, done. We can go home now.

7.1. What about NPV?

Now this value does have units, namely dollars. And because it does, we must re-assess the validity of one of our concepts: the time value of money. If we're to switch our units to time, what does it mean to refer to the "time value of time"? Conceptually, we need to resolve the premise that time now is more valuable than time later. Is it?

This could be argued both ways. Let's say you have 2 weeks of vacation, and further that these weeks can rollover year over year until whenever you wish to use them. Maybe there is more value in using them now; after all, you could be dead in a year. But then Universal keeps opening exciting new theme parks. What if in 2 years they add a Jurassic Park Cruise where the room stewards dress as raptors and you have a port of call on Isla Sorna, and you could have gone had you not used the 2 weeks on Orlando? Decisions, decisions.

This 100% sounds like the kind of thing I'd have written a thesis on in school, and they'd have been all, "this is interesting, but it's not economics until it's quantified", but I'm not going to do that here. All that we need to recognize here is that you're going to have to make your own decision and assign your own value to the NPV discount rate (and it's ok for that value to be negative). Just take the time to think through your situation and come up with a value that makes sense for the priorities of your organization.

Ask your executives, and you'll likely hear that they'd rather realize a return on time sooner than later, because executives tend to keep an eye on mitigating risk. A project that completes sooner decreases risk because it represents a shippable, (possibly) revenue generating asset. If it succeeds, those revenues kick in sooner, leading to them literally being worth more due to the time value of money concepts discussed above. If it fails, the company takes the loss but has stopped pouring money into development sooner. Either way, there is a strategic net benefit to software leaders to apply a positive time value of time in calculating the NPV of potential automation efforts.

Once you've decided on a reasonable discount rate, NPV can work just fine using time as its unit of measurement. Exercise care assuming this extends to rainbows.

8. The Evaluation Process

The process is quite straightforward, given a step #0: create a spreadsheet that calculates IRR and NPV based on initial investment, with slots to input expected hours returned at various dates in the next 3

years. Why 3 years? That's arbitrary. You can extend it to 5 or 8 or whatever, but in CAP's technical sphere, rare is the infrastructure script that can be expected to survive more than 3 years of re-architecture. Once you've replaced your EC2 cluster with Fargate, or started managing Kubernetes with GKE, your carefully written scripts to resize, rehydrate, or redeploy are of no use. So, given you have a spreadsheet, the process is:

1. Do a task the first time, and remember you did it
2. Notice that a task is toil, by virtue of having to do it a second time
3. Record how long it takes to do it by hand
4. Mentally break down the process into automatable chunks
5. Estimate how many dev-hours would go into coding the automation, along with how often and when this toil would have to be done in the future
6. Enter #5 into the spreadsheet (as a negative value), and #3 whenever you expect to have toil replaced by automation in the future (accounting for the fact that until the automation is complete, you'll still have to do it by hand)

That's all you need to document the opportunity, which you will do in the "Data" tab as illustrated below in Figure 7. Then whenever you get capacity to do some automating, pull up the spreadsheet, sort by IRR, NPV, and Breakeven time, and compare the top few of each to find a combination you like. Maybe you only have a couple weeks until a larger project hits, so you take a smaller automation task with a high IRR. Maybe you've got a lull and a lot of time to do something big, so you shoot for high NPV.

Let's look at a sample spreadsheet. Here's what tab 1 – "Dashboard" – looks like:

Opportunity Name	IRR	NPV (hrs)	Breakeven	Idx
OpenStack Usage Accounting	-49.40%	(37.57)	Never	1
xCloud Usage Accounting	56.24%	69.35	Jan 2022	2
Big Good Project	35.71%	590.14	May 2022	3
Small Borderline Project	11.23%	0.75	Mar 2023	4
Medium Good Project	23.38%	58.42	Mar 2023	5
Big Bad Project	1.39%	(426.32)	Never	6
Doesn't Even Make Back Investment	-19.40%	(197.74)	Never	7

Figure 6 – Example IRR/NPV Dashboard

These values are calculated based on data entered in a second tab (named "Data") along the guidelines listed above. The first thing to note in the top left corner of tab 2 is the discount rate. We set this to 10% as a first guess as to how much our "time value of time" is, well… valued, and over time and experience we have had no reason to change it. It turned out our instincts about the time value of money for our team were good enough to be useful to our prioritization efforts. Notice also how the larger a project is, the greater the number of empty cells at the start to account for elapsed time spent in implementation:

Discount Rate: 10.00%	Development	Hours of Toil Avoided																			
Opportunity Name	Hours (Month 0)	Year 1												Year 2							
		1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4	5	6	7	8
OpenStack Usage Accounting	-100		6	6	6	6	6	6	6	6	6	6	6								
xCloud Usage Accounting	-100				6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
Big Good Project	-1600			75	75	75	75	75	75	75	75	75	75	75	75	75	75	75	75	75	75
Small Borderline Project	-40						8						8						8		
Medium Good Project	-240												120								
Big Bad Project	-2800													120	120	120	120	120	120	120	120
Doesn't Even Make Back Investment	-480				10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10

Figure 7 – Example IRR/NPV Data Input Tab

Something to think about here is the difference between "Big Good Project" and "Big Bad Project". The returns (120h/mo vs 75h/mo) basically scale with the investment (2800h vs 1600h), and so if one is good, it stands to reason so is the other. The difference here that sets them apart is calendar time to complete. Big Good Project completes in 2 months, whereas Big Bad Project – at less than twice the number of required hours – takes a year. In real-world terms, what we're looking at here is the difference between a team of specialists, and a team that has cross-trained. Big Good Project is done by a full team made up of engineers all qualified to work this particular effort, and so can break it up and work in parallel. Big Bad Project is executed by a team in which only one or two engineers is qualified for this work, so it takes longer. But in each case, the applicability of the technology is still valid for only 3 years from inception, so the time to develop eats into the return.

To convert the raw data entered in tab 2 into the easily readable and sortable dashboard in tab 1, we need some Excel magic. We mentioned above that Excel offers built-in IRR and NPV formulae, which means we can skip the numerical methods for calculation and do this is a way that is accessible to the average engineer. Take a look at the formula for IRR in the first row of the dashboard:

```
=IF(Data!C4 <> "", XIRR(Data!C4:AM4, Months!$A$1:$A$37), "")
```

A sharp eye will notice that there is a third tab we have not yet mentioned. This 3rd tab, named "Months", is a simple, static list by row of dates representing the first day of each of the next 36 months. This allows us to drive our calculations by date. The XIRR function supplied by Excel takes the data from the "Data" tab, applies the dates from the "Months" tab, and calculates all the discounted values, sums them up, and solves for 0. The final piece to the formula is that "IF" at the beginning that simply allows us to show an empty cell if the data is not found, rather than a confusing internal error message.

The NPV cell is extremely similar in its formula:

```
=IF(Data!C4 <> "", XNPV(Data!$B$1, IF(ISNUMBER(Data!C4:AM4),Data!C4:AM4, 0),
Months!$A$1:$A$37), "")
```

Within the same basic structure, we now use the XNPV function, which requires the discount rate (from Data!$B$1), our effort data, and the months, plus the requisite embedded IFs to account for any possible missing information.

Getting back to the dashboard as a holistic entity, the first two rows (sorted by "Index" order, the order entered in the Data tab) represent a real case documented by the process above. We had to audit our usage of cloud resources and would have had to every month for the foreseeable future. We recorded time during the task, and it came out to 6 hours. A rough estimate of the time needed to automate the task came to 2½ weeks of work for one person.

Another bit of information that must be considered is that cloud teams sometimes migrate from one private cloud platform to another. The old system gets named after the legacy software on which it is based. The new system uses an internally branded codename, in our case "xCloud", and with a phased "go live" schedule that deprecates the old platform as hardware is shifted over.

Now at first glance that seems like a big investment: 2½ weeks to code something that only saves 6 hours a month. Which is exactly why we do this mathematically. This example, when applied to xCloud usage happens to provide the largest return on the sheet, at an IRR of 56.24%! By anyone's standards that would be a great candidate for a project to undertake. On the other hand, if we use the same effort to automate

legacy cloud usage, we see that it's a non-starter due to the limited lifespan of the solution. Because we will only (generously) get a year of use out of the automation attributable to migrating off of the platform, we never make back the time investment.

At other places in the sheet, you can see examples of good projects, projects right on the border with NPV near 0, and a couple that are negative, but that highlight the difference between IRR and NPV. "Big Bad Project" has a positive IRR – though less than the discount rate – but a hugely negative NPV. We'd technically end up saving time on it, but at the cost of missing out on a whole lot of better options along the way. "Doesn't Even Make Back Investment" is a loser from the start with a negative IRR, though overall it loses far less than "Big Bad Project".

9. Example

Taking as an example the case of xCloud Usage Accounting, we can walk through this process:

1. The team's VP has requested an accounting of how much we are spending in operational expenses for the cloud resources we use in the Elements project. We need to break this down into categories for compute, storage, network, etc. The work is manual but assisted by an internal web interface.
2. For the second consecutive month, the same request arrives to collect usage data. The work is done manually again but flagged for automation.
3. We record that the task required 6 hours of work to complete manually.
4. The internal tool has an application programming interface (API) that provides programmatic access to the categories we need to create this monthly report. We determine that we can write a script that queries each category we need information on, merge for each component by category, and assemble the data into a simple report of use at an executive level.
5. The work is estimated to require 100 developer-hours, which we can schedule over the course of 3 months based on our capacity as a team.
6. We open the spreadsheet to the "Data" tab. -100 is entered in the column for "Development Hours" to account for the time to automate. 3 months are skipped to account for elapsed calendar time spent developing, and 6 is entered in each month thereafter to realize returns on our investment from month 4 to the end of year 3.

Flipping back to the "Dashboard" tab, the information has been instantly calculated to yield an IRR of 56.24% and a NPV of 69.35 hours.

10. Conclusion

The basic concept of using data to support business decision making is hardly groundbreaking. We use various graphs and equations in the preparation of business cases, project performance reports, and managerial accounting as just a few examples. The higher levels of corporations are filled with Masters of Business Administration (MBAs) and other people comfortable speaking in finance as a second language, but often this perspective does not filter down to those closer to everyday decision making.

This paper presented a complicated, manually-operated system of interconnected dependencies and walked through a breakdown of the automation it took to render it into a self-managing entity. We also discussed one method of leveraging established financial models, at the individual task level, to yield a simple, plug-and-play tool for prioritizing toil automation.

It could be argued that the most valuable part of all of this is establishing a pattern of discipline to collect and organize the data (steps #1-5 of "The Evaluation Process"). Go ahead and edit your user story template to add pre-filled acceptance criteria with entries for:

1. Elapsed time spent on the task
2. The automation effort estimate

…and you're most of the way there.

Abbreviations

API	Application Programming Interface
CAP	Core Application Platforms
CI/CD	Continuous Integration / Continuous Deployment
CPU	Central Processing Unit
DC/OS	Distributed Cloud Operating System
IaC	Infrastructure as Code
IRR	Internal Rate of Return
MBA	Master of Business Administration
NPV	Net Present Value
PR	Pull Request
Pub-sub	publish/subscribe
SRE	Service Reliability Engineering
TF	Terraform
VM	Virtual Machine

Bibliography & References

Cost of Capital vs. Discount Rate: What's the Difference?, Christina Majaski; https://www.investopedia.com/ask/answers/052715/what-difference-between-cost-capital-and-discount-rate.asp

Internal Rate of Return: Numerical Solution, Wikipedia; https://en.wikipedia.org/wiki/Internal_rate_of_return#Numerical_solution

