Funding Flow:

The Middle Ground of Agile Budgeting

By Roland Cuellar, LitheSpeed LLC and Tom Paider, Nationwide Insurance

01

Introduction

by Evan Leybourn

The profession of management accounting has continued to evolve ever since the modern concept of budgeting was defined by James McKinsey in 1922. Each new practice is built on what has come before in order to serve the needs of the companies of the time.

In today's economy, companies are seeking agility. For almost the first time in history, customers having more information than the firms trying to sell products and services to them. In turn, companies need to adopt a new mindset; to be truly customer-centric. To put the needs of the customer above the short-term needs of the shareholder – which, in turn, ensures that the long-term needs of the shareholder are met.

These companies (which, as you are reading this, hopefully includes your company) are seeking new ways of responding to the changing needs of their customers. In technology team concepts we saw Agile emerge. In product teams, concepts like Design Thinking are becoming popular. HR and Marketing have their own responsive and agile ways of thinking and working. And finance teams are no different. Over the last 20 years, we are seeing new management accounting practices emerge to support this new mindset.

But, like all change, it takes time.

What you are reading here is a first step; a middle ground of Agile budgeting. What Roland and Tom have created here is helpful, insightful, and practical. This is just the start of your journey and I ask you to take these first steps with an open mind and a willingness to continue to evolve – just as every accountant since 1922 has done.

-Evan Leybourn, Founder, Business Agility Institute

According to the Business Agility Institute's Global Business Agility Report from 2019, there are three clear organizational predictors of business agility: flexible funding models, organizing work around value streams, and the drive for relentless improvement. Much has been written about the power of value stream management and creating a culture of continuous improvement. Less has been written about the "how" of moving to flexible funding, particularly the application of these models in companies with a long history of annual project funding cycles.

As these organizations attempt to pivot away from fixed-constraint projects towards product lines and value-streams, they find that traditional project funding models are incongruous with agile methods. Flexible funding models that allow teams to quickly pivot in reaction to internal and external feedback continue to be much sought after but seldom seen in the agile community.

We have seen a number of companies successfully move towards flexible funding models through their product-centric transformations and attempt to do away with project-based funding in favor of flexible funding. But the truth is that for most organizations, this is just a bridge too far. Such a change would involve a radical change in funding, governance, estimation, and even accounting. To be successful, an intermediate step is needed… a way that maintains some of the current financial model while allowing for the benefits of agility. In this paper, we will show how an intermediate approach that both maintains the current project-based model and a flexible agile model can co-exist.

Larger organizations will struggle with adopting an entirely new funding model in line with Agile ways of working. They have questions like:

· How do we justify project approvals and how do we budget these efforts?
· What financial inputs are required?
· How are the projects financially tracked once underway?
· How will we know what we are going to get and when?

Of course, the traditional approach most organizations use is a rigid funding model that locks in scope, timeframe, and budget in the hope of creating predictable financial outcomes. The sad truth, however, is that outcomes are seldom predictable. The original schedules are rarely hit, and substantial budget overruns are commonplace. Worse still, the business outcomes are not achieved at the level that was desired, even if most of the requirements were delivered. In fact, it is very common for the original business goals to be completely lost as the focus turns to locking down and tracking detailed scope and cost. Traditional approaches attempt to create highly precise views of delivery that are most often very inaccurate in terms of both project outcome and business outcome.

A principal flaw in this model is that we can know, a priori, exactly what the user will want and use, and that through their usage of these features, we will achieve our desired business outcomes. Even the most advanced product development organizations have a long history of significant product failures:

· Apple Newton
· Apple Lisa
· Apple eMate
· Google Nexus
· Google Plus
· Google Inbox
· Google Picassa
· Microsoft Phone
· Microsoft Zune
· Windows Me
· Microsoft Cortana
· The list goes on and on and on

If the richest and most experienced product development firms on the planet cannot reliably predict what consumers will value, and what the resulting business benefits will be, then it seems somewhat ridiculous to think that we, using our long, linear, outdated, budgeting model, can do better.

More flexible approaches are key to creating an environment where we can quickly develop, deploy, learn, and adjust in order to iterate towards a winning solution. But for most organizations, this is simply too big of a change to the status quo. It is too great a leap to go from rigid scope-bound budgets and plans to what is sometimes perceived to be an open checkbook.

So where is the middle ground?

Rather than a wholesale change of the funding model, there is a path forward that allows organizations to both maintain their project model and much of their funding model while still achieving greatly improved organizational agility.

Outcome-based Funding

Organizations can continue to fund projects but they should fund business outcomes instead of detailed scope-based projects.

In this model, we would spend $XM in order to achieve some desired business outcomes such as account growth, cost reduction, compliance with a regulation, parity with a competitor, etc. In the outcome-based model, we still have a project, but the primary controls are around the business outcome, the budget, and the timeframe, not the requirements' scope. For example, we might say that we need the following:

– Reduce the cost of processing X by 15%
– To be achieved by the end of the year
– With a budget of $2.5M

Organizations do not have to abandon projects entirely to achieve business agility– they can still have projects with a defined & measurable business-outcome target, a timeframe, and a budgeted dollar amount. The important difference being we allow the requirements scope to flex to meet the business needs. This gives the flexibility to add/change/remove requirements as needed to achieve the outcome. In this model, the business outcome is "the thing"!

Paired with early delivery of functionality and an experimental mindset, we should start to see early financial results– cost savings, revenues, new users, lower dropout rates, etc. If we don't, we can learn from the results and quickly pivot to functionality and features that will provide results. And these actual results can be measured against the monies spent to date to see if the financials are still making sense.

But How Do I Estimate in this Model?

There are several ways that one could come up with plausible estimates in this model: Bottom-up and Top-Down. Let's start with Top-Down.

Top-Down Estimation of Cost

In the Top-Down estimation method, we can work backwards from a desired financial goal in order to develop a business case. Imagine that we have done the market or cost-savings analysis and some product or project stands to achieve $5M in new revenues or cost savings for us over some timeframe.

The next step would be to get a rough budget for this work that would fit within our budgetary constraints or meet some internal financial ROI hurdle. Maybe we only have a certain amount available to spend on this. Or perhaps that in order to justify the investment, we might be willing to spend up to $3M in order to get the anticipated $5M. Either way, we can get a top-down budget for this investment without having to know the detailed requirements.

But how do we know if we can achieve the outcome with this top-down imposed budget? We might need to do a quick rough estimation of some of the key high-level requirements. The trick would then be to NOT lock in those requirements but simply use them as budgetary placeholders. As we learn more, actual requirements and solutions would evolve and subsequently update the business case.

The requirements and estimates that we come up with are used simply as a gauge, not as a strict mandate. What is contracted is the outcome and the budget and the timing. We will let the requirements' specifics flex so that we can find the most impactful and economical solutions.

But How Would We Estimate the Top-Down Cost?

By now we are back to having to estimate high-level requirements. How can we do that without knowing the details? Estimating requirements in hours is immensely time consuming and often grossly inaccurate. While it does give the illusion of precision, that precision is usually false. This is why our estimates are so often off by 100% or more. But there are alternatives that are much faster and easier and that if used conservatively, may be more accurate.

Agile Teams are Fixed Cost over the Medium Term

Agile teams are cross-functional and stay together over the medium term. The ramifications of this are huge from a cost estimation standpoint! Imagine this scenario:

1. A 10-person agile team has 4 developers, 2 testers, 2 analysts, a database person, and a scrum master.
2. Some folks make more and some make less but on average, the blended cost per hour is $150 all-in.
3. Each sprint is 2 weeks long or 80 working hours

In this example, the cost of the team, per sprint, is 10 * 80 * $150 = $120,000. There are two important pieces of information here– the cost and timeframe. Every 2 weeks costs $120,000.

As we estimate work, let's get away from huge spreadsheets of people and titles and rates and allocation percentages and all of that. Instead, estimate in team-sprints.

For example, we might estimate that the candidate requirements would require 2 agile teams for 10 sprints plus or minus 2 sprints.

Cost : 2 teams * 10 sprints * $120,000/sprint = $2.4M
Time: 10 sprints * 2 weeks/sprint = 20 weeks
Variance: 2 sprints which is 4 weeks and up to $480,000

So now we have a defined business outcome, a cost, and a timeframe, and an idea of the uncertainty. This way of estimating is fast and easy and if you really do have standing teams (which you definitely should!), then you will have a pretty good idea of what each team can actually do based upon real data. Using that data and a predisposition to quick, early delivery allows the dual benefit of more accurate financial forecasting as well as quick stress-testing of how the delivered features meet the expected outcome.

An important outcome of this approach is that with a budget in place and a desired outcome expected, teams will have to choose solutions that are both impactful and relatively economical. They will be forced to consider "best bang for the buck" solutions that make financial sense.

We can compare this rough estimate against our top-down budget and see if we are in the ballpark. This estimate might not be perfect, but given the rate at which our current methods miss the mark on budgets, it is probably no worse but is a lot faster and easier to perform.

If the fast and easy top-down approach will not work in your organization, then there is the less desirable, but effective bottom-up approach.

Bottom-Up Estimation

In a bottom-up approach, we can develop more detailed requirements that we think will be needed in order to achieve the outcome. We can then estimate the requirements in the usual way. But the estimates are used only to get ballpark cost and duration numbers, not as a commitment to deliver those particular detailed requirements. Some of the 'requirements' may be throw-away! In this model, the requirements are nothing more than our current ideas on how to solve the problem. We may - and probably should - have new and better ideas once we start really getting into the project and begin to learn more. We can use our current best guess at the requirements to get the numbers we need and then basically throw the requirements away– or at least allow them to change as we learn more. Working this way, we get a cost estimate and a schedule estimate. We still have our targeted business outcome, but we are not going to measure progress as a function of delivering requirements. We are going to measure progress as a function of delivering incremental business results. But more on that later.

Why do we prefer top-down estimation? Because it is significantly faster and easier and in most cases, will provide the necessary level of control. Development of detailed requirements and estimates, only to then possibly abandon those requirements once we see that they are not achieving the outcome is potentially a huge waste of time and money. And given that our ability to accurately estimate is bad at best, the outcome is likely to be unsatisfying.

We Do This All the Time!

Think of a simple grocery shopping example. You are throwing a party and you need to estimate the food costs. You might budget for certain snacks, drinks, entrees, desserts, etc, etc. And you might use specific numbers to help you come up with more accurate estimates. For example, you might budget for 10 bottles of wine at $39.99 each and so on and so forth. These specifics all add up to an overall budget that you decide you can live with. You get to the store and find that some wines which are just as good happen to be on sale and that they cost less. You then find that something else you wanted is out of stock forcing you to buy something that costs a bit more. We are all totally fine with this approach and we use it every day to manage our own money. The goal is not to walk out of the store with your exact shopping list, the goal is to have a well- stocked party that is within your budget. It would be ridiculous to rigidly stick to the shopping list in the face of the new and more accurate information that you receive when you arrive at the grocery store.

We can do the same thing with requirements. We can estimate requirements for budgetary purposes to help us establish an overall cost target, but it may turn out later that another requirement that we didn't anticipate can get us a better outcome for less money, and that some requirements we thought we needed might not be necessary at all. So, as long as we hit our business outcome, we should be good. In fact, I'd be very inclined to say that we have a much better chance of actually achieving the outcome using this model and we can often do it for less money.

Too Much Focus on What It Costs and Not Enough on What It's Worth!

Here is a common pattern that we see frequently. There is a huge amount of time and effort that goes into coming up with the precise cost of a project to within some ridiculous percentage of accuracy but not nearly the same level of effort in what the project is worth. It is common to see fuzzy business upsides, hockey stick projections of growth, financial benefits that are not planned to materialize for years, sloppy consumer behavior studies, and dubious economics at best. Large sums are often spent simply because an important stakeholder is demanding some capability and not because it makes good financial sense. Also, because our financial planning cycle is so long, we justify the lack of rigor in the upside by saying "we don't have time" and "we know our business and we know what our customers want". However, we can probably agree that given the huge number of projects that fail to meet their stated objectives, we obviously don't know our customers much at all. At least not to the level that we are able to monetize their behaviors. In our experience, we need to focus much more on the value of the work, why we are doing it, and how we will measure the outcome in real terms. Basically, there is way too much focus on the "I" part of the ROI and not nearly enough on the "R". In the end, a highly precise investment estimate divided by a fuzzy return is still fuzzy at best. The result is poor investment decisions that utilize enormous sums of money and tie up our limited resources only to result in mediocre levels of business improvement.

How Do We Manage Changing Scope

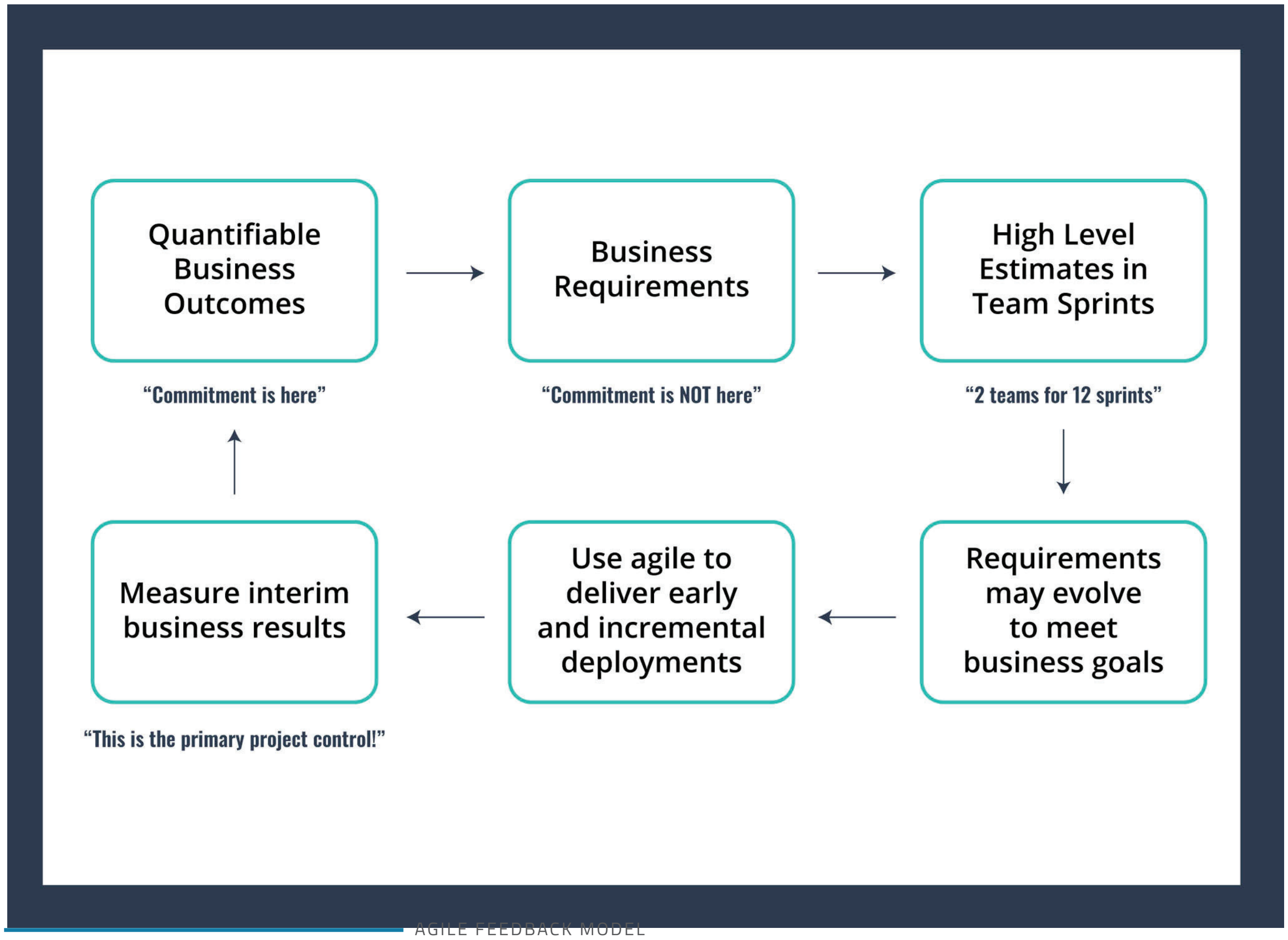
Either way you go, top-down or bottom-up, our model calls for scope to be defined more loosely. If a new requirement comes in that helps us to achieve the agreed upon and funded business outcome, then it is fair play. And if a requirement comes in that does not directly tie to the business outcome, then it is not in scope. And even if a requirement or request might help to achieve the outcome it may still be rejected if there is a simpler, cheaper way to achieve the outcome. This sort of thinking can greatly cut costs since most projects have many requirements that are hitching a free ride and do not clearly tie to measurable business outcomes.

In this model, requirements will evolve and change. In fact, that would be evidence that the process is working! We should change our requirements as we learn more about how the customer behaves, about the economics of this situation, and about the outcomes of our technical solutions. Changes in requirements do not break our governance/control model because our model isn't based on requirements– it is based on outcomes. The change management issue then becomes one of communications and alignment and transparency. Various PMO meetings, customer meetings, backlog prioritizations, team planning meetings, backlog reports, and other avenues exist for maintaining communications and managing change so that we can stay aligned. Alignment is around the goal and we need to be flexible to meet that goal.

But if there is no fixed scope, then how do we measure progress? What is the key control?

The Business Outcome is the Key Control!

In this model, commitment is at the business outcome level, not at the requirement level. The project is funded to achieve an outcome and the requirements are allowed to flex in order to make that achievement possible. The Product Manager, Project Manager, Architect, and other key parties are now on the hook to discover solutions that are both effective and economical. Business outcomes should be measured at regular intervals to determine if the project is meeting its business goals. Wait! Measure regular business outcomes? That's too late! The project will be over before we can measure against the controls, right? Wrong! Not if you are correctly using an agile feedback model.


Quantifiable Business Outcomes
"Commitment is here"
Business Requirements
"Commitment is NOT here"
High Level Estimates in Team Sprints
"2 teams for 12 sprints"
Requirements may evolve to meet business goals
Use agile to deliver early and incremental deployments
Measure interim business results
"This is the primary project control!"

AGILE FEEDBACK MODEL

Measuring Progress

In order to measure progress towards the business outcome, we must demand interim releases to production that allow us to get objective data on the performance of this project! The only measure that matters is actual incremental business results. This flexible funding model is predicated on getting real feedback and adjusting the product requirements in order to achieve the business outcome. If we don't deliver something, then we don't get the feedback, and we can't make the pivots necessary to be successful. ***In exchange for flexible funding, leaders need to demand early and frequent delivery, and demand measurable operational data that can be used to assess the business outcome and drive financial performance. Without this, you have nothing more than a waterfall project with a blank check!*** And here is where finance and accounting can be drivers of business agility! By demanding early and frequent operational data that can be used to justify continued spending, the business achieves financials results sooner and gets massively more visibility into the viability of projects than it ever did before in its old 'plan the work and work the plan' model.

Monetize at the Feature Level

In order to make the effective trade-off decisions, our business partners will need to get much better at understanding the economics of the features that they are asking us to build. By this, we mean that they need to understand that some features are both high value and low cost and therefore they are clearly the economic winners. Other features are low value and high cost and are economic losers. Business is typically and traditionally of the "I want it all" mindset and so we have our teams spend huge amounts of time and money building too many features that turn out to be economic losers. By packaging up the economic losers with the winners into the same project, we lose insight into where the real value is. We should insist that the business get much better at monetizing at the feature level. The WSJF (weighted shortest job first) technique can help here. WSJF is a technique that allows the business to rank or compare the relative ROI of each feature against each other so that we can see which features appear to be the economic winners. Once we have down-selected to what appears to be the most impactful features, we can do some basic estimation in terms of team-sprints, the business-upside, and most importantly, how we intend to operationally measure the business upside. If we cannot operationally measure the desired outcome in some way, then the feature should probably be cast aside.

What About CapEx?

Traditionally, capitalization and expense were managed using waterfall phases with the early phases being expensed, later dev-test phases being capitalized, and any work done after deployment being expensed once again. In agile, we are performing analysis, design, development, test, and fix all simultaneously. Obviously, a phased approach will not work here. But we can manage capex using other means. The most obvious, and perhaps even more accurate than the traditional approach, is to use the actual work items or tasks themselves. Using an agile project management tool that tracks stories and tasks, we can say that new feature/functionality story development is capitalized, defect stories are expense, testing tasks for new functionality are capitalized, etc. In this way, we can measure the actual amount of work going towards expense versus capital.

In some cases, the organization may have capital versus expense targets or ratios that they need to maintain. In these cases, we could work backwards and give our product owners "budgets" for capital and for expense. If this were needed, we could say that, for example, 40% of the backlog items can be expensed but the rest must be capitalizable for example. In this way, we are engineering a financial outcome by prioritizing our backlog in line with capex guidelines.

But things get more complex after we have an initial deployment to production of an MVP or other minimal solution. If we continue to add on to this minimal solution, is it capital or expense? If the add-on work offers new or enhanced functionality then it probably should be capitalized. If new work is maintenance/defect-fix then it should be expensed. But some firms have more rigid internal rules around this such as "any work done to support the release after deployment is expensed" or "a new software version release is synonymous with a project". In these cases, we may need to work with accounting to design smarter (and more accurate) rules that are still in alignment with the Generally Accepted Accounting Principles (GAAP). In these cases, it can be helpful to put more fine-grained definitions around the word 'release'. For example, if we are deploying new code to production every few weeks, is each of those new deployments a new release? Deployment and release do not have to necessarily be synonymous. For example, one could define a new version release to specifically be the launch of major new functionality vs "patches" which might only serve to deploy smaller changes. Both result in the deployment of new code to production but they could be treated differently for accounting purposes. It is not uncommon for some firms to have major new-functionality releases that are separate from smaller patch deployments. In this way, we can treat each separately from an accounting perspective if needed.

The capex topic can be tricky but most auditors have by now seen many of their customers go down the agile path and will have some experience in how to handle these situations. The capex problem is readily solvable if there are minds at the table that are willing to explore solutions that both meet GAAP expectations and allow us to solve the needs of our customers and business operations. A key part of the solution is likely to be an agile project management tool of some sort that we can use to tag or categorize stories and tasks so that they can receive the most appropriate accounting treatment. And the result should be more accuracy not less.

Tying It Up: It's About The Time Value of Money

It is amazing that even the largest and most sophisticated firms do not often focus sufficiently on the time value of money (TVM) with respect to project work. Everyone should understand that a dollar today is worth more than a dollar next year for a variety of reasons. And yet, remarkably few organizations force project investments to deliver dollars back to the business RIGHT NOW.

The financial investment advisory firm "The Motley Fool" explains TVM this way.

"*Time value of money is one of the most basic fundamentals in all of finance. The underlying principle is that a dollar in your hand today is worth more than a dollar you will receive in the future because a dollar in hand today can be invested to turn into more money in the future. Additionally, there is always a risk that a dollar that you are supposed to receive in the future won't actually be paid to you.*"

There are a couple of concepts here that we should explore a bit more deeply. The first is that a dollar in hand can be invested now and the second is that dollars you are supposed to receive in the future may not materialize.

We typically invest in projects in order to be paid a return. Projects are usually expected to reduce costs, increase revenues, improve efficiencies, reduce risks, etc. This means that most of our projects should be paying us back not only for the cost of the project but additional gains as well.

Now, in most organizations, the demand for projects greatly exceeds the funds and capacity available resulting in many unfunded project requests. Imagine what we could do if all of these investments could start paying us back sooner? We'd have more money and we could fund more work and get even more accomplished! The key is to force projects to start to pay us back sooner! But how can we do that?

Agile and DevOps techniques give us the tools to accelerate both deployment and payback. By focusing our efforts on a few features that we believe are economic winners, and using agile to design, delivery, and deploy those solutions quickly, we can start to reduce costs now! We can start to bring in more revenues now! This generates more money and it generates it now so that we can start to make those additional investments now, just as The Motley Fool said in the aforementioned quote.

But what if we make these deployments and we don't see the uptick in revenue or the downturn in costs? Well, this is where the second part of the TVM definition comes in. As the definition above stated: "there is always a risk that a dollar that you are supposed to receive in the future won't actually be paid to you." By using early deployments, we can start to see that some of these projects are not going to be able to pay us back! We can see sooner rather than later that the economics just aren't there. The calculated upsides were faulty and our estimated costs were low. Using agile delivery, we can see that we aren't going to get paid back and we can start to consider whether or not to kill the project.

The time value of money is one of the most basic elements of finance, and yet most organizations do not adequately focus on TVM. The result is that it is not uncommon for projects to spend extraordinary amounts of money over the course of multiple years only to deliver NO VALUE back to the business in the interim. These projects often have long payback cycles and are frequently bolstered by questionable economics whose validity cannot be measured for years since no interim deployments or payback is being required.

A more agile (and financially frugal!) organization would heavily tilt funding towards those projects that can achieve positive financial impact sooner rather than later. Focusing on the time value of money metrics will force more frequent delivery of value, lower risk, and allow more frequent measurement of project economics. Working this way would allow us to make smaller bets!

So how do we do achieve these highly desirable outcomes of smaller bets, lower risk, more frequent measure of project economics, and positive TVM?

1. By favoring projects that have a plan to achieve faster payback
2. By funding projects based upon outcomes instead of funding a set of requirements
3. By having our teams focus on economics at the feature level
4. By demanding early and frequent measurement of actual business outcomes through early and regular deployments of functionality

There is a middle ground between the current way that projects are funded and the stable long term funding that the agile community is recommending. Organizations can both keep their projects and achieve far greater business agility, if they can adopt these practices. Let's be clear that we very much favor long-term stable funding of value stream teams, but the project-outcome funding model that we outline here can be a strong intermediate step that most organizations can start to implement right now.