Agile system dynamics (AgileSD) for fast, effective, reliable dynamic business models

Kim Warren

Dynamic Business Models (DBMs) – or "Living Business Models" – fully simulate business issues and plans, explaining past performance and projecting likely future outcomes under varying scenarios and strategies. They capture all significant interdependencies between familiar business items and indicators; customers, capacity, staff, sales, cash flow, investment, quality, and so on. And these are "white-box" models, displaying with complete transparency how every item in the system changes from period to period. This make DBMs the tool we need for the joined-up management everyone says they want.

But DBMs will only be adopted by influential leaders, and have real impact, if we can create understandable, working models with the minimum effort. Fortunately, it is now easier, faster and more reliable to build working models of any plan, function or issue than it is to attempt the same task with spreadsheet tools – and the resulting model is much more useful (see sdl.re/PowerOfModels).

This article explains the "agile" approach to creating a DBM. Working with the problem owners, you build a simple working model from the start. You then expand the model step by step until its purpose is fulfilled, matching its structure and behaviour to real-world evidence at all times. The process is fast and self-correcting, so you can work with leaders and teams to build living business models in real time.

New-comers to dynamic business modeling should find the process and tools simple enough to start learning and using this valuable skill easily and quickly. Anyone already skilled in dynamic modeling should find that AgileSD has big advantages over traditional approaches. Now that these powerful models can be readily developed, every mid- to senior-level executive should understand them, and every analyst and consultant should be able to build them.

(See an in-depth course with 'follow-me' demonstrations and working models at sdl.re/courses. A LinkedIn Pulse summary of this paper is at sdl.re/LIPagileSD, and a more extensive paper on the issue is at sdl.re/AgileArticle).

DBMs exploit the well-established science of System Dynamics (SD). Some sources imply that SD is a complex and somewhat abstract method, but its underlying principles are quite simple and practical.

Build on rock-solid principles

We can get to reliable models quickly by following those fundamental principles of SD:

1. We are trying to understand and improve ***performance over time***, whether that performance is financial or non-financial (service quality, for example). Our need may concern the long-term, sustained results of a business plan for a whole organisation or for a function, or it may concern an isolated issue, such as fixing an operational problem or successfully achieving an important change or new initiative.

2. In each period, performance is directly determined by ***quantities of Stocks*** that exist – Current customers drive this month's sales, staff numbers drive current service capacity and labour cost, and so on. So, for example …

   sales/month **=** *customers * sales/month per customer*

   or … *service hours/month = service staff * hours/month per person*

   Stocks are things that accumulate over time, like cash in a bank account. Customers, staff, product range, physical or IT capacity, and cash are all Stocks, as are some operational items, like inventory and outstanding orders. Intangible items like staff skills, information and reputation are also Stocks. Since performance depends on these Stocks, how performance changes over time ***must*** reflect how those Stocks, too, change over time (more customers next month gives higher sales next month; more staff gives higher service capacity, and so on)

3. ***Stocks accumulate and deplete*** from period to period – so they fill up and drain away, or are won and lost. This behaviour is mathematically quite different than the 'Y depends on X' that links items in principle 2. For example …

   customers at the start of next month **=** *customers at start of this month*
   + *customers won during the month*
   - *customers lost during the month*.

   Customers won or lost per month are the 'Flow-rates' that cause the Stock's quantity to change. This universal, absolute mechanism applies to all accumulating Stocks , and is in fact very familiar, because it is how the balance on your personal bank account or credit card changes over time! See sdl.re/customersales for a very small model linking principles 1-3. All we need to do is apply the same principle to everything else in the business system that accumulates, and then join them up – which is what principle 4 achieves.

4. The rates at which any Stock accumulates or depletes in any period (its flow-rates) ***depend on the existing quantities of Stocks*** that exist at that time. In many cases, for example, customers won/month depends on the current number of sales people we have – so the stocks of customers and sales people are interdependent. The customer win-rate may also depend on the current number of customers (through word-of-mouth). Whenever a Stock drives its own rate of change, this causes ***feedback*** that may accelerate growth or decline, or slow down that growth or decline. Such feedback may operate only around the Stock itself, as in word-of-mouth, or involve other Stocks. For example, a product winning customers may then be stocked by more stores, who then attract more customers. ***Decisions*** and ***external factors*** may also drive Flow-rates – hiring drives additions to the Stock of Staff, for example, and competitors' prices will affect the Customers loss-rate.

Combining these principles leads to a working, quantified model capable of simulating any business, any function, or any challenge or initiative where we want to understand and manage how things change over time.

> *Anyone already familiar with SD modeling may be used to focusing on the feedback loops in the system. However, the rates at which Stocks accumulate are the underlying causes of model behaviour, and feedback is a* ***consequence*** *of the interdependence between those accumulating Stocks. Any feedback loop can exist only if it includes at least one accumulating Stock. Without this, you have simultaneously mutual causality (the dreaded 'circular calculations' in spreadsheets). This is not just a mathematical*

nuisance, but reflects a real impossibility in the real world. A Stock must exist in any feedback loop in order to pass material or information forward to the next period of time, and thus break the simultaneous causality (**this** month's customers recommend us to others, who are added **next** month).

The unique mathematical properties of accumulating Stocks are thus fundamental to how any business system behaves. The agile process identifies these items very early in steps 2-3, and feedback loops become apparent only when step 4 discovers them.

Numbers matter!

We need models to tell us ***how fast*** things are changing, and ***by how much*** we might improve performance ***over what time***. To achieve these improvements, we must make decisions, or do things, that are also quantified – by how much to change the price, how much marketing to spend, how many staff to hire, and so on. Even if it were possible to develop qualitative causal models that are accurate, they cannot answer these questions – and with no numbers to confirm their causal relationships, qualitative diagrams may be inaccurate in any case.

So if our models are to actually ***work*** and generate quantified behaviour, they must include formulae for the relationships between items, such as the examples in principles 2 and 3 above. Not every causal relationship is easy to define in mathematical terms, so we sometimes need to estimate how an item changes as the factor on which it depends varies. For example, if service workload is less than 100%, then service-errors per 100 customers = 0 (or some very small number), but that error-rate grows in some non-linear way as workload increases above 100%. SD models capture such estimated relationships with "look-up" functions, similar to spreadsheet look-ups, but much easier to use.

The agile process – follow the principles, and quantify

A simple case demonstrates a quick and practical process for building reliable models, following the principles above. We explain later why this process is termed "agile".

The company in this case provides IT-support to mid-scale businesses – small retail groups, accounting firms, construction and transport companies, and so on. It employs skilled technical staff to support clients' needs for computer hardware, software and communications. The company recently tried to grow by taking on more clients. This was successful at first, but some two years later it was suffering a high rate of client losses, due to service quality problems.

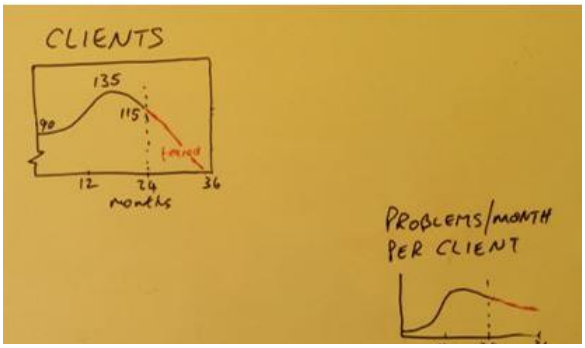
Here are the steps in the process, with a description of what was done with the CEO in this case.

Step 1. How performance is changing over time. Draw a time-chart(s) of the issue(s) of concern, including relevant history and desired future. The time-chart **must** have a scale on it and the chart's line must reflect how the performance indicator has actually changed in the past (*unless the situation is entirely new*!) and is expected to change in future.

> **The CEO:** "*2 years ago we had about 90 clients, and were doing well, so took on a sales executive to sign up more new clients. He helped us grow to about 135 clients, but then we had signs of rising client problems, peaking at nearly 3 per month for every client. Staff have to fix those problems fast, rather than get on with their normal job of client support. We have since lost some long-standing clients and are now down to 115. We fear it will take time to reduce the rate of problems, and in the meantime clients will continue to be lost*. "

Figure 1 shows the time-charts sketched with the CEO, showing what happened over the latest 24 months, and what he feared may happen out to month 36. Note that the Clients chart is inside a box, to indicate that it is in fact a Stock.

Figure 1: Step-1. Time-charts of the IT-company's situation



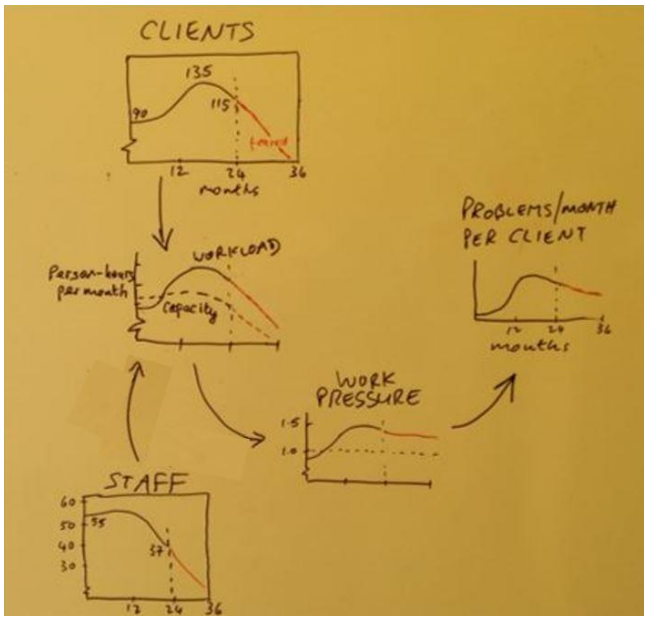
Step 2: How performance depends on Stocks. Starting from each performance outcome, ask "*What items does this depend on*?" to trace the causes of performance outcomes in each period. We are looking for the direct and immediate causes of every item, not distant, indirect causes (like "*hiring the sales guy caused the* problems"). For this direct causality to be correct, it must be possible to ***calculate or estimate*** the value of the performance indicator arithmetically from those immediate causes. To prove that this causality is valid, add time-charts to each item.

Repeat the question "*What items does this depend on*?" for the new items you just added – we must again be able to calculate or estimate each item from those on which they depend. Continue until the chain of causal relationships reach one or more of the following items – Stocks, decisions or external factors.

> **The CEO:** "*It looks like those client-problems were caused by the pressure of work on our staff. We log the work they do – installing equipment and software, upgrading systems, training users and so on – and that workload increased much more than our capacity to do the work. People were working into the night and at weekends. We estimate the service demand workload may have been at its peak 60% more than people could do in a regular work week*."

Figure 2 shows the chain of relationships sketched by the CEO. The visual links show that, if we know the work pressure, we can estimate the problem-rate clients will experience. What determines work pressure? – it is simply the ratio between the hours of work that must be done and the capacity of staff to do that work. The Stocks of Clients and Staff are clearly driving performance in every period. (*These causal chains are very short in this case, but are rarely more than 3-4 items long in other cases*). The relationships in this case are simple and clear – clients' problems appear to be entirely explained by the pressure of work, not by major software upgrades, for example, or by other factors. But where causes are less obvious, uncertainties are resolved by seeking numerical evidence for the causal relationships.

Figure 2: Step-2. Quantifying the causes for the IT-company's problems over time



Although the diagrammatic relationships may look right, we can't be certain that they ***are*** right,

especially in more complex cases. So, *Immediately replicate the diagram in a working model*. Figure 3 shows the model for this case (see sdl.re/AgileITstep2).

Note that the whiteboard diagram in figure 2 displays service demand and capacity in a single chart, but the model has a separate item for each. Red chart lines show actual history and management's feared future – continuing client problems, and further loss of clients and staff. Blue chart lines show what the model is calculating. Allowing the model to continue past month 24 will show its forecast for what might happen in the later period.

To make the working model, we must formulate the relationships in step 2, just as we would in a spreadsheet model …

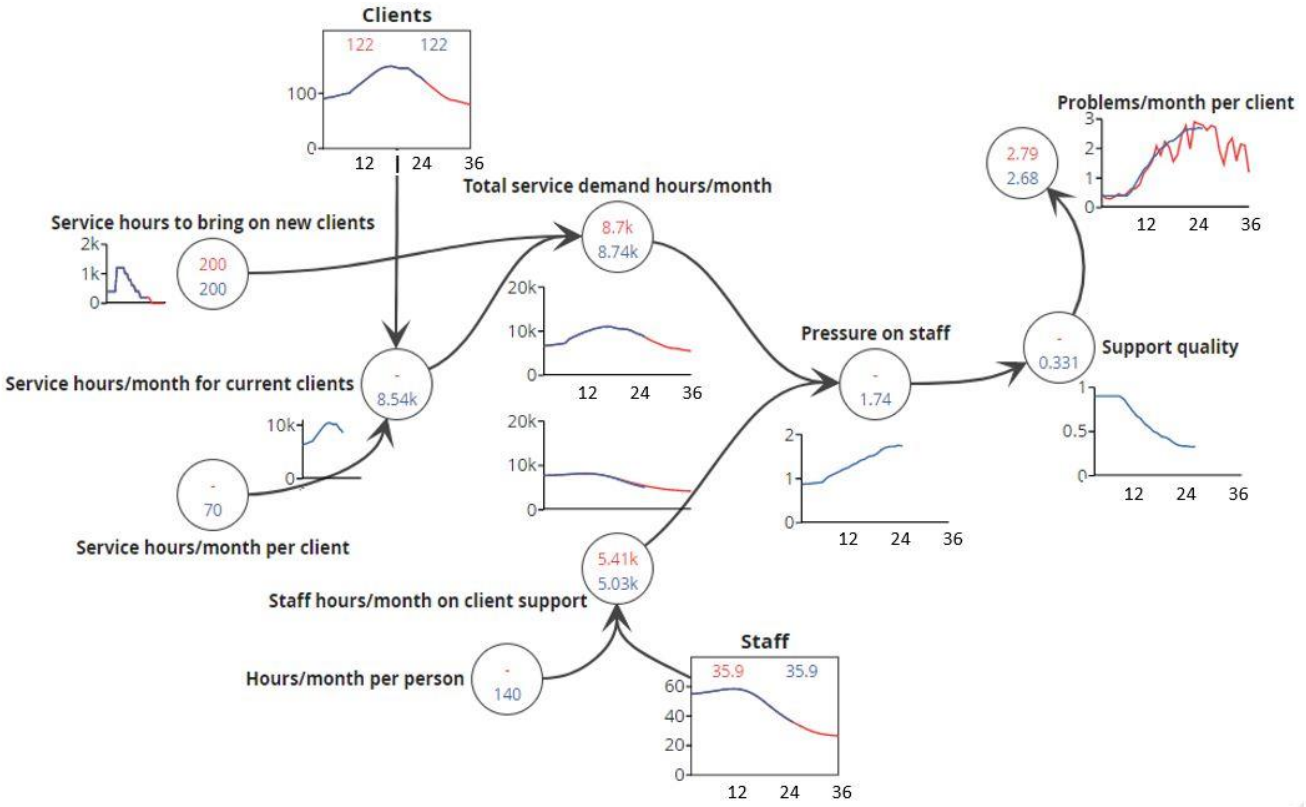
Pressure on staff = Total service demand / Staff hours per month on client support

Total service demand = Service hours to bring on new clients
+ Service hours/month from current clients

*Staff hours per month on client support = Staff * hours/month per person*

Note that this is what formulae actually look like in SD models, with natural-language names – not abstract spreadsheet codes, such as "A$21 * B5"!

Figure 3: Screen-shot of the work-in-progress model for step-2 of the IT-company's issue



If you continue sketching out the white-board diagram, be sure to include quantified time-charts for every new item and keep developing the working model alongside the diagram. ***Do not*** be tempted to lay out causal relationships without data, either on the white-board or in the model, or all these benefits are lost – ***there is no qualitative mapping in the AgileSD process***. It is very well known that misperceptions about how things actually work are common, so causal diagrams based on participants' 'mental models' will likely contain errors. And if we will eventually need a working model in any case, why not quantify and model from the start?

Often, the people you are helping may quickly trust that the model is indeed capturing well how their world works, so you can cut short the white-board diagramming and continue with the software model alone. This will only be possible, though, [a] if the model shows time-charts on every important item, and [b] if you can show both real-world data and calculated results together for those items.

> *System dynamic modelers may be used to showing model results separately from the model structure. So, for example, figure 3 would be shown with no data or charts, and selected results-charts would appear on another page. By combining the charts and structure you make it much easier for the users to understand what-causes-what and substantially increase confidence that each relationship – and the model as a whole – is actually correct. See how to do this in the software user guide at sdl.re/syqfull, clip-7. Other software tools will require a different approach, but it is essential to invest the effort to combine data and structure in this way.*

Throughout this process, be sure to use properly specified names for every item, avoiding abstract or ill-defined labels. Try to use names commonly understood by those involved in the case, although you may have to agree with them corrections to any ambiguity – people may talk about 'sales' when they in fact mean 'revenue', for example.

If these disciplines are followed, participants continue trusting the model at each stage because they recognise every item, and all the values for every item. If they do not have all the data, some items may have to be estimated, but the working model tells us if those estimates are reasonable. If, for example, the CEO had estimated the *service-hours/month per client* wrongly in figure 3, then the blue chart for *Total service demand* would not match the known values in red. The model is therefore validated from the start, and its validity is maintained throughout the process.

Step 3. How Flows change the quantity of Stocks. Add in-flows and out-flows to each Stock, including time-charts for how those Flow-rates have changed over time, and may change in future. (*Note that if the issue of concern identified in step 1 is itself a Stock, such as "grow to 300 clients by the end of next year", then the agile process starts here and misses out step 2*).

Step 3 explains how the quantity of each Stock from step 2 changes over time, which can ***only*** be accounted for by the rates at which the Stock has been added-to and lost in every period. This is the basic 'physics' of how the real world works, and there is no valid means to ignore the mechanism. You cannot, for example, explain the number of customers in terms of marketing spend or price – those factors (and others) drive the rates at which customers are won and lost, not the Stock itself.

> *This has important implications for business analysis. If the quantity of a Stock, such as customer-numbers, is 100% determined by the whole history of their win- and loss-rates, then it is meaningless to use statistical methods to explain their value. And if that is the case, then it is also meaningless to use such methods to try to explain anything that* ***depends on*** *that Stock, like sales and profits. It* ***is*** *safe to use statistical methods to explain everything up to a flow-rate – working out which factors cause customers to be won or lost, for example – but they fail at the Flow-to-Stock boundary.*

This critical role of Flow-to-Stock relationships also means that you ***must*** identify in-flow and out-flow rates separately – 'net' rates of change are not good enough. A different mix of factors will drive the win-rate from those that drive the loss-rate, often because different people are involved. It is *potential customers* who are available to be won, for example, but *current active customers* who

might be lost. You should also have separate in- and out-flow rates for the hiring of staff -v- staff losses, the launch of new products -v- the discontinuing of current products, the addition of new capacity -v- the retirement of old capacity, and the in-flow of cash -v- cash out-flows.

Here is what the CEO said about the Flow-rates in this case, and figure 4 shows the resulting additions to the diagram – clients won and lost/month and staff hired and lost/month.

> **The CEO:** "*The sales guy was successful at first, he raised the new client rate from 2 per month to about 7, but he struggled to keep that up, and now we are winning few clients at all. Previously, we rarely lost clients, but starting about a year ago, client losses increased to a peak of 8 a few months back. We are losing fewer now. On the staff side, we kept hiring about 2 people per month as before. We previously lost people very rarely, but in recent months many more have left.*"

The model is again developed in parallel with the diagram to confirm that the behaviour patterns of the Stocks are consistent with how the Flow-rates are thought to have changed. It is not easy to sketch by eye how, say, the client win-rate and loss-rate changed over time to be consistent with what happened to the Stock of clients, but the model will ensure that these relationships are exactly correct.

The stage-3 model is at sdl.re/AgileITstep3, and the screen-shots in figure 5 show how the win- and loss-rates for Clients and Staff changed over time, and might change in future.

Figure 4: Adding the step-3 flows of clients won and lost, and staff hired and lost

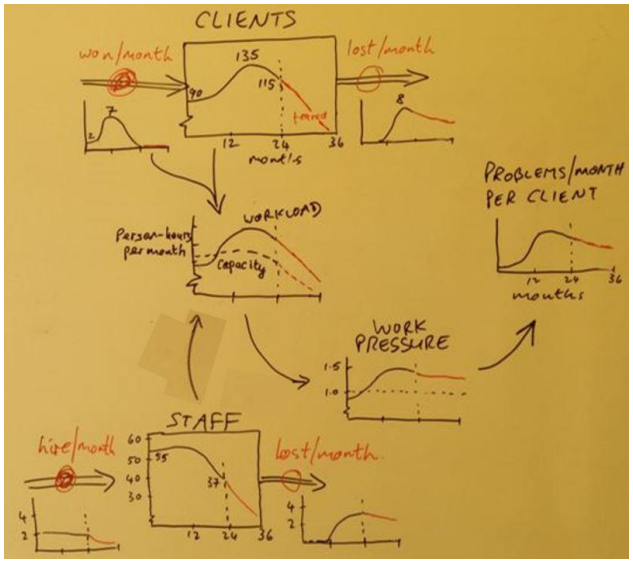
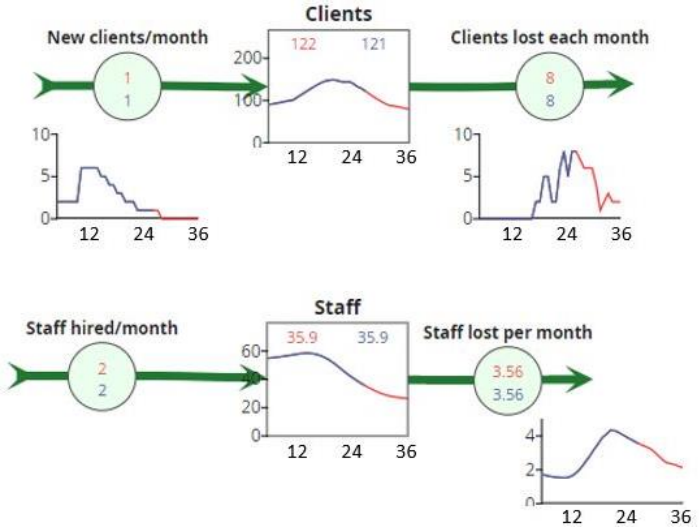


Figure 5: The win and loss rates for clients and staff added to the working model at step-3



The heavy arrows in figure 5 indicate that the arithmetical relationships in this step are fundamentally different from the formulae in step 2 that work for every period:

Clients (at end of month) = Clients (at start of month)
+ New clients/month – Clients lost/month

These Stock-Flow relationships are what cause things to change from period to period and are therefore the mechanisms that drive changing performance. This means they are ***critical*** to understanding and managing that performance – the ultimate key-performance-indicators (KPIs).

Step 4: How existing Stocks (also Decisions and External factors) **drive the Flow-rates.** The last 'unknowns' we have by this point in the process are those Flow-rates added in step 3 ... so simply repeat the "*What causes this*?" questioning we did in step 2 to find out what is driving those Flow-rates.

Here is what caused those win- and loss-rates for the Clients and Staff in this case.

> **The CEO:** "*Our sales guy won the new clients – we didn't pick any up from competitors, or so far as we know from recommendations by existing clients. The later client losses were driven by the problems we created for them. It is a nuisance and takes some time to find another provider, but if you have had a lot of problems over many months, you will lose patience and eventually switch to another provider. On the staffing issue, we were continuing our normal hiring rate, but the faster turnover was inevitable – people cope with over-work at first, but after a while they have had enough of the pressure and leave.*"

Generally, step 4 identifies three types of item driving the Flow-rates:

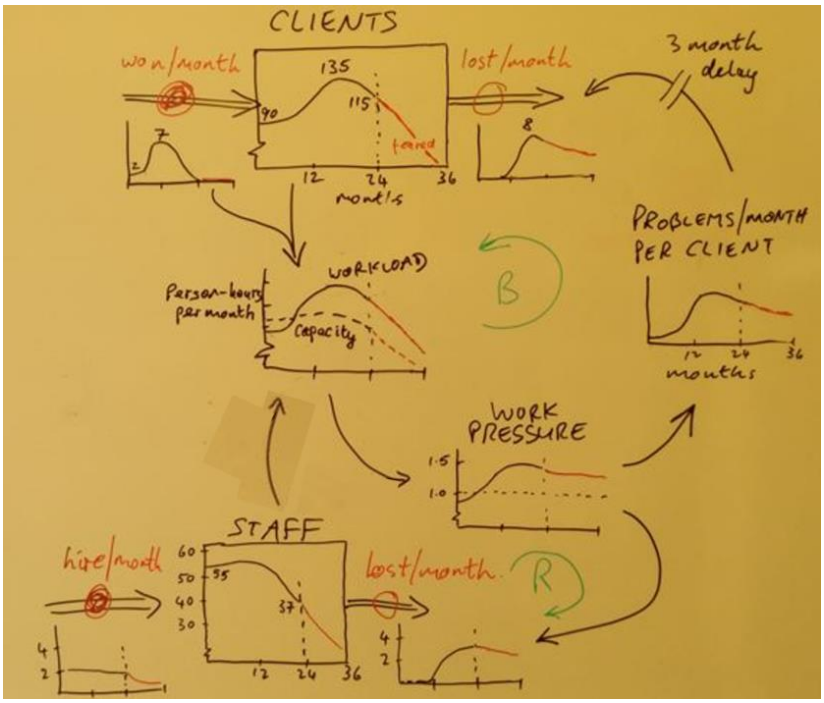
Decisions – in this case, the sales-person's efforts, and the hiring rate.

Existing Stocks. Client-problems drive the Clients loss-rate, and those are in turn caused by the imbalance between the Stocks of Clients and Staff. The staff loss-rate, too, is driven by those same two Stocks, through the work pressure.

External factors – none in this case, but market conditions or competitors' activities are often important in other cases.

Figure 6 shows the result of completing step 4 for our case. The only additions from figure 4 are the links at top-right and bottom-right, explaining the loss-rates of clients and staff.

Figure 6: Completing the interdependencies in step-4 for the IT-support case

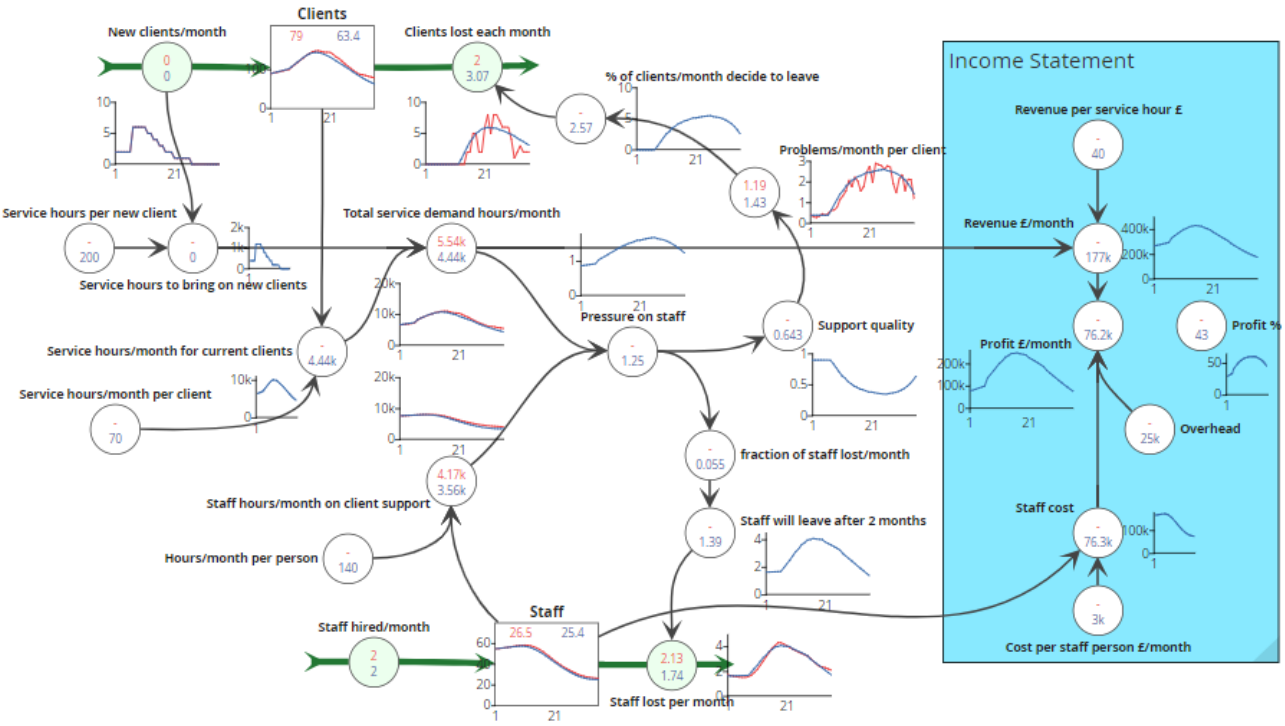


Note that the client *win*-rate fell during the second year, due to the reputational damage caused by poor service quality. So the client win-rate was ***also*** driven by the imbalance between Stocks of

Clients and Staff, as well as by the Decision to deploy the sales executive. This connection is not yet shown in figure 6.

You can see the stage-4 model at sdl.re/AgileITstep4. See also *a full working model at* sdl.re/BMITsupport4a, including a simple Income Statement for the business (figure 7). This model allows you to change some decisions and other values, and explore how service quality, profits and numbers of staff and clients change over the full 3 years. You may run the model to month-24, then change some decisions to try and fix the problem, or alternatively change decisions earlier to avoid the problem in the first place!

Figure 7: The IT-support model



Dynamic business models, KPIs, balanced score-cards and spreadsheet models

Note that figure 7 is the ***entire*** model and has fewer than 30 items – there is nothing else going on. A good SD model is therefore 'parsimonious', featuring the minimum set of items needed to explain and manage the outcomes of interest. You may choose to add further items, such as the profit % of revenue (shown here next to the Income Statement), or staff-cost ratios, but they do not explain or control system performance.

This means that such a model can provide the most compact set of KPIs or balanced score-card possible. Indeed, figure 7 ***is*** such a score-card and KPI-set. If the model shows not only what every key indicator is doing, but also why it is changing and how it will likely change in future, what value is served by having separate tables and reports? For businesses with more moving parts, sections of the model can be separated out. The VP Marketing, VP Product Development, VP Human Resources and so on can therefore see what is happening in their own domain, whilst still being clear about their dependencies on each other.

Managers and analysts alike may by this point be feeling anxious that figure 7 is a quite alien view of a business model and be craving the comfort of their familiar spreadsheets. But there is a quite simple way to understand how such a model relates to an equivalent spreadsheet model.

- Think of each object as being a single spreadsheet column, with its name in the top cell and each month's values running down the cells in that column.
- Think of the linking arrows as equivalent to the cell references in that spreadsheet. (*These links makes it positively difficult to create the cell-reference errors that plague many spreadsheets*!) The heavy arrows for the gain and loss of clients and staff are used simply to highlight the special cases of accumulating Stocks.
- Each object contains the formula telling that item how to calculate its values from the items on which it depends.
- There are in fact two data-columns for each item – one for real-world values and another for the calculated values – shown in our models in red and blue respectively.

If the relationship between a DBM and an equivalent spreadsheet is so simple, why do we need a system dynamics model at all? The clarity with which a DBM displays causality and changing performance should already be obvious, but the spreadsheet option has more serious limitations.

Unfortunately, human brain cannot intuitively figure out the combined impact of ***stock-accumulation***, ***interdependence*** and ***feedback***. We can only manage such systems at all because many feedback mechanisms act to stabilise things without our intervention – if you win more customers than you can serve, then bad service loses the excess and normal service is resumed (though not in a good way!) Introduce just one more effect and worse things happen, as occurred when the IT service case started losing staff as well as clients.

More complex business models feature many more accumulations, interactions and feedbacks, making it impossible to develop a rows-and-columns model for anything but the very simplest of cases. Two further mechanisms add to this impossibility.

First, business systems feature many ***thresholds***, where the relationship between cause and effect is highly non-linear. The model in figure 7 manages to capture one such relationship in a simple formula (*how support quality varies with pressure on staff*), but many such relationships can only be described in visual terms. These are easily captured by the look-up functions in SD software, but tough to replicate with spreadsheet tools. Then there are ***intangible*** factors, such as staff skills and reputation. These follow the same rules as the tangible items in dynamic models, but they complicate the interdependencies and thus the system's behaviour.

So, while spreadsheet models of complex dynamic systems may be theoretically possible, it is quite impractical to build such models. And in any case, why try to do that, when the task is easily and quickly done with a tool that is fit-for-purpose? You may of course still need your spreadsheets, for other analysis to which they are better suited, but it is easy to make the necessary links with DBM data.

Choosing time-scales and time-units

If you are new to dynamic modeling you may have noticed that the IT-support model covered a quite long time-scale (3 years), but in quite small time-units (months). This is a common feature of DBMs and arises for both practical and theoretical reasons.

The ***time-horizon*** of the model must of course cover the time-scale of the issue or plan if it is to be useful – from the time we became concerned to the time by which we expect the challenge to be overcome, or the plan to be achieved. But in our case example, that could just have been the last few months and a few months into the future. We go further back in history for two reasons. First, the accumulations in the system mean that a problem appearing to be quite recent may in fact have

its origins much further back in time. That extended history may even include periods when everything appeared to be OK, although, as in this case, it already contained the seeds of its later problems. Secondly, including plenty of history provides the evidence of how the system actually works. In this case, the last few months would tell you only how bad the problem is when in crisis, but nothing about where the system needs to be moved to in order to work well again.

For larger business plans, still longer time-scales may be necessary. If you are developing a substantial new product, or planning a major expansion of capacity, for example, the time-horizon may need to be many years. We are ***not***, however, claiming that the resulting model is forecasting exactly what will happen – there are far too many uncertainties to make that claim. Rather, it is projecting a scenario that describes the scale and direction in which things will need to change for the outcome or plan to be successful. As events unfold, we will keep updating the model to adjust the longer-term projection in any case.

What about the ***time-units*** of the model? Business plans, whether for established businesses or for new ventures, are typically laid out in years, or possibly in quarters – and rarely include more than the latest year's results for comparison. Once again, more history is important, for the same reasons we just noted. But modeling plans and challenges in annual or quarterly terms is wholly inadequate, both because the real world is changing faster than this, and because we can respond faster too. If a recent sales campaign is more successful than expected, for example, you will likely see evidence of that in months (or even weeks or days!). And if that means you will be short of product supply or support staff in a few months' time, then you can take preparatory action also in a matter of months, weeks or days.

There is also a technical reason for modeling in short time-units. If you choose time-units that are too long, then values may change by a large fraction from one period to the next. As a result, the model will fail to reflect properly the interdependencies. If, for example, customer numbers are growing by 2% per month, but your model is in years, you will only reflect this as a 24%/year growth (*already an error, because this monthly growth rate actually equates to 26.8%/year*!). But if you have supply capacity or service support that can handle only an increase of 10%, then you will entirely miss the implications of running out of those items by month 5.

So, *model in short time-units, over long periods, including plenty of history*.

> *Experienced SD modelers may be used to dealing with the challenge of change-within-time-periods by forcing models to recalculate several time-steps within each time-unit, using the 'dT' feature. Although technically correct, the concept is inexplicable to non-specialists, and introduces fictitious units of time. A quarterly model is much preferable to a yearly model with 0.25 time-steps, and a weekly model is preferable to a monthly model with 0.25 time-steps. Not only are the units more understandable, but real-world activities mostly happen over periods of days, weeks or months, not over artificial fractions of longer periods.*

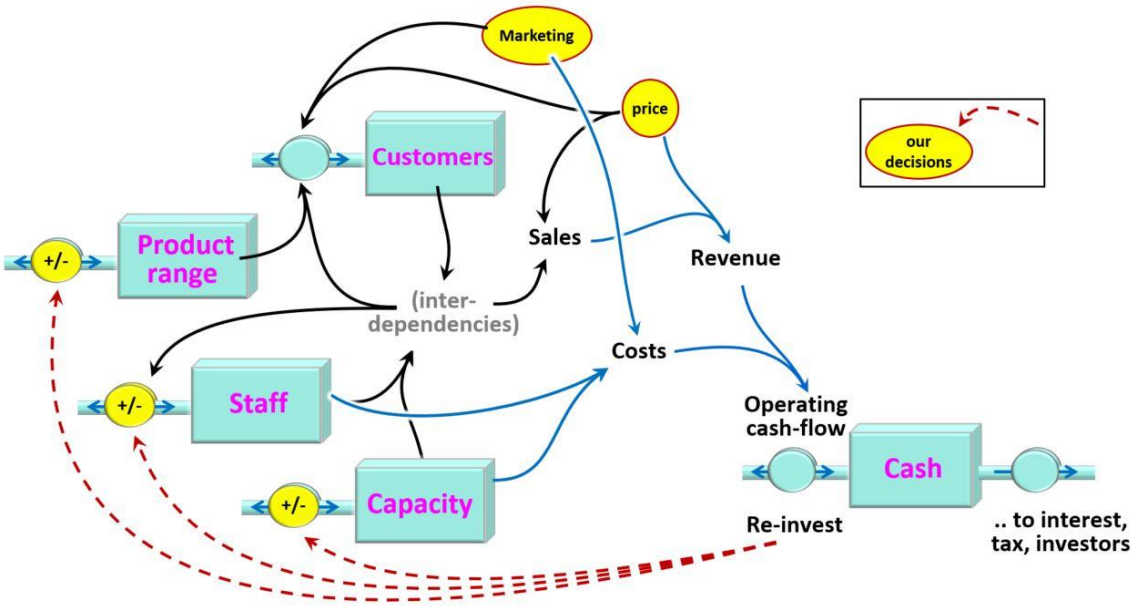
The core 'Strategic Architecture' of an enterprise ... and common extensions

The process described to this point is enough to produce a core model for many cases. These may be limited in scope to a single function or issue (*as in the IT-support case*) or be more extensive and capture the system of inter-related resources in a whole enterprise. Many enterprise models share a

common 'strategic architecture' – a minimum set of standard items. (*See figure 8, which for simplicity combines in- and out-flows into two-way net flows*). In this structure:

- customers drive sales and revenue, and are won by attractive products, prices and marketing
- staff and capacity make it possible to sustain customers and sales, but also drive costs.
- we reinvest some of the operating cash-flow to sustain or add to product range, staffing and capacity.

Figure 8: The generic core strategic architecture of a business.



Certain business-types may need further resource-Stocks, such as intermediary customers (retailers for consumer products, or dealers for auto-makers), and orders or contracts in the case of very-large-value products (ships, aircraft) or projects. Note that we chose the IT-support case to illustrate AgileSD because there was no 'product range' issue, and because staff ***are*** the capacity for a service-based firm.

This core model template – including special-case resource-Stocks – may often need to be extended to capture important additional features and mechanisms. (*Warning! Some of the models linked-to below may appear quite complex, but all are nevertheless consistent with the principles set out previously*.)

> **The quality of Stocks** Adding more customers, staff or products may be great, but we would prefer them to be good-quality – high-spending customers, skilled staff, appealing products. Such quality measures can be modeled by attaching an "attribute" to any Stock, which rises if we add good items, but falls if we add poor-quality items (or lose good ones!) See for example a customer-revenue model for the IT-support case at sdl.re/BMITsupport5a, and a model of changing staff-experience at sdl.re/mhgs.
>
> **Chains of Stocks** The simple case we explained earlier has just two, individual Stocks; Clients and Staff. But many Stock-items move through 2 or more stages; for example, active customers may be won from a limited Stock of potential customers, and staff may arrive as trainees before moving into a Stock of skilled staff. In the staff model

in figure 9 (see sdl.re/BMstaff3b), the business needs to grow its team of engineers by quarter-24, so raises its hiring of trainees from 'now' (quarter 13).

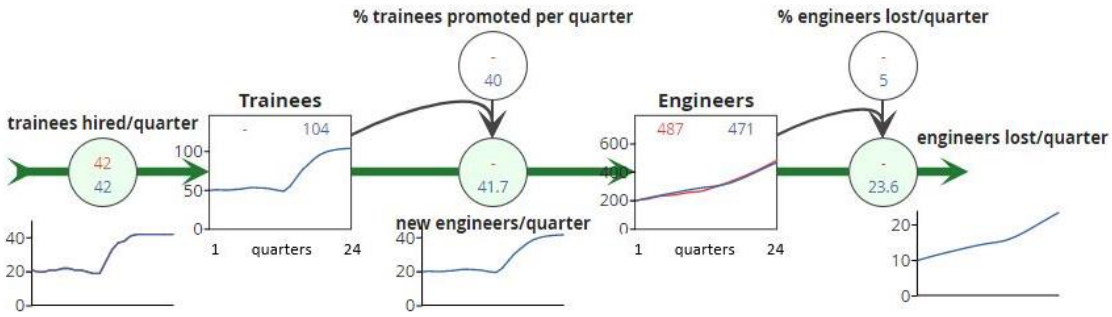
Just two levels of seniority may not be enough for our staffing strategy, so we may need to model how staff move through more levels. Customer-capture too may go through several stages – see sdl.re/choicepipeline. Stock-chains also arise for product development and for the aging of equipment and other assets.

These models are all examples of "strategic" pipelines – chains of Stocks whose long-term development have a big impact on overall business performance. But dynamic models can also handle the details of "operational" pipelines, such as supply-chains of physical products, or customer-journey models that track how customers are moved between the different processes they need to be taken through.

Competition! Modeling how competition plays out under different assumptions and strategies is ***very*** useful. Three standard mechanisms deal with all such cases; type-1 rivalry to capture potential customers, type-2 rivalry to steal customers from competitors, and type-3 rivalry for a higher share of sales to shared customers. See sdl.re/BMpharma4a for a model of intense and urgent competition in the pharmaceuticals industry, where type-2 rivalry dominates.

Intangibles We know that 'soft' factors matter, so we must include them in our models when they are significant. These may concern [a] the state-of-mind of key groups of people, such as morale or reputation [b] data, information or knowledge, or [c] quality-related factors.

Figure 9: A simple staff chain



Why "AgileSD"?

The term "agile" is borrowed from the field of information-systems (IS) – here is why.

The traditional approach to building SD models has close analogies to what is known as the "waterfall" process for IS development. A project starts by identifying the entire scope of the final system, then attempts to lay out everything about how the system must work in order to cover that scope. This process involves all stakeholders in the solution, including senior leaders. Only when everyone is happy with the whole design does work start on building the working system. Programmers then must work with everyone involved through a lengthy process of coding, testing and revision to arrive at a system that does indeed work as everyone originally intended.

The waterfall approach has benefits, such as a clear overview and control of the overall system design, and avoidance of duplication and inconsistencies between related developments. But it has serious problems. It ***takes too long***, and requires ***too much work*** from large numbers of users,

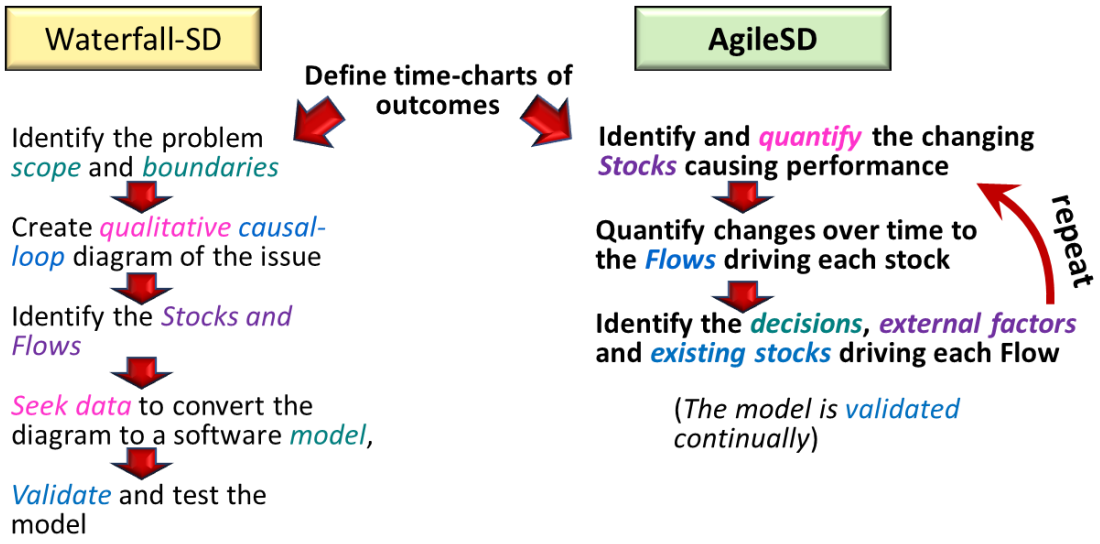
including senior leaders. Facilitating the early scoping process can only be done by ***the most experienced professionals***. Because the early design has to ***speculate*** about the final solution, the resulting system may ***not meet user needs***, and may include ***faults***.

The response of the IS community to the problems of the waterfall method was to evolve an "agile" approach to system-development. First, a simple, working solution is built quickly, in close collaboration with the people who will actually use it. This early prototype is then revised and extended in small increments, with the users' continued involvement, until the system covers the most important requirements. It may never do everything that all stakeholders would have ideally wanted.

There are close analogies between the experience of waterfall IS development and many SD modeling projects. These often take too much time and effort, and the process can only be done by experienced modelers and facilitators. The resulting models may fall short of meeting the users' needs or be too complex to understand or use. Qualitative mapping cannot be validated, so models from data-free diagrams can have fundamental errors – either by including model-elements and causal relationships that do not exist in the real world, or by missing important elements and relationships that do exist. (*A professionally-developed model of an insurance company, for example, did not include policy-holders*!)

The AgileSD alternative also has close analogies with the agile approach to IS development, notably the immediate attention to a working model that matches the real-world issue of interest, and the step-wise incremental development, during which a working model continues to be available at all times. Figure 10 contrasts the two approaches.

Figure 10: How AgileSD compares with the common 'waterfall' approach to developing new models



Or … start from models that work

There are very few truly novel business models, and very few business challenges that are entirely unique, so an even faster approach to modeling business issues and plans is to start from an existing, proven model. In the case of whole-business models, the fundamental elements and structure are mostly common for all businesses in a given sector – customers, product-range, staff, capacity and cash are linked in much the same way for all restaurants, for all airlines, for all car-makers, or for all corporate-law firms, for example.

Of course, the data and the strength of relationships in different cases within the same industry will differ considerably, and any specific case may have important structural differences from the sector template, but we can still be guided by the known structure. Any need for adjustments to that structure will emerge from the data-matching that runs through the whole process. If, for example, customer-growth is not accurately explained by the common set of factors found for businesses of that type, then some additional factor or mechanism must be involved.

The simplest cases of whole-business models are to be found in entrepreneurial settings, where they are used to plan new-ventures before launch, to raise investment, and then to manage the business from week to week. See sdl.re/NewVentures1 on why and how this works, and sdl.re/OpunTalk for a CEO discussing how he used his model for these purposes.

Exactly equivalent purposes also arise for any new issue, challenge or opportunity in on-going businesses – for example, making and integrating an acquisition, or pushing through any change in how the business works. In all such cases, you need to plan how it should work out, gain support and resources for the change, then manage the situation as it develops and evidence comes in about how the change is actually progressing.

A more complex case in marine engineering

In September 2017, the International Maritime Organisation brought into force a Convention requiring ship operators to install equipment to clean the ballast water that keeps vessels stable.

Winning international support for the Convention took many years, so equipment manufacturers had been developing, testing and selling systems for a long time. The CEO of one such company needed to model the growth of the new business he aimed to build, in order both to win the financial backing required, and then to plan the product-development, staffing, sales effort and operations-management of the business. See *sdl.re/BWTdemo*, explaining the case, the agile process the CEO followed, and the resulting model.

Many proven models also exist for specific types of issue, often focused on particular functions in a business. The 'extensions' to the core method that we explained above are examples of models that are proven to capture common challenges faced by businesses of many different types. However, the AgileSD principles are still needed in order to adapt these template models, to extend their scope, or to connect them to other structures. Any of the stock-chain models, for example, may need to be connected to model structures that work out the financial consequences of their behaviour.

Conclusions and next steps

The AgileSD approach to building dynamic, "living" business models offers substantial benefits:

- Users get a useful model quickly and with limited effort, and get further value continually as the model is extended.
- The resulting model is totally transparent, and always matches what users recognise and data they understand, so achieves high levels of trust.
- The process is self-validating – if ***any*** item in the model does not match real-world data, at any time, something is wrong.
- Not every model will prove to be useful enough to continue working with, but with the agile process this is soon apparent, so the effort can be stopped with little lost investment.
- The process is not reliant on senior professionals with decades of experience and the stature to facilitate large-group debate with senior people. Relatively junior analysts – or even the business users themselves – can generate useful models.

As with any new skill, beginners will make mistakes, but these are immediately obvious, and skill will grow with practice.

What to do next ...

Check the list of Do's and Don'ts on the next page.

Learn how to build dynamic business models with the online course, featuring 'follow-me' demonstrations and many working models – see sdl.re/courses/business-modeling-course/.

See more on the benefits of agile-based business modeling at sdl.re/PowerOfModels, and the extensive range of related videos at youtube.com/strategydynamics.

See how to use the simple software for building Living Business Models at sdl.re/sygfull.

Download this article at http://sdl.re/AgileSD, or a more extensive, referenced paper at sdl.re/AgileArticle.

Strategy Dynamics Ltd
Princes Risborough, HP27 0JS, UK

Tel. +44 1844 275518 : www.strategydynamics.com/contact

AgileSD : Doing it right!

DO	DO NOT
… work with the person or team who is responsible for the plan or issue of concern	… build causal diagrams with no data
… start from a clear, quantified time-chart of one or more objectives	… include abstract or badly-defined items
… cover plenty of history, and build any diagram or model in small time-units	… 'race ahead' by adding much structure to the model without checking that each small section works
… make sure all causal links reflect clear mathematical or 'look-up' relationships	… ask people to speculate about feedback loops for which no evidence exists ('loop-hunting') – but do look for feedback loops that emerge from the evolving structure
… highlight the importance of Stock-Flow structures and check they are working correctly	… ignore items for which hard data does not exist – ask people to make well-informed estimates (but test them!)
… build a small, working model from the start	… lose sight of the plan or issue that users want solved and the uncertainties and decisions they face
… encourage users to take ownership of the model by inviting them to work with it	… try to build every model from nothing – look for pre-existing, proven models to start from
… include real-world data (actual history and desired future) alongside simulated results	… jump into modeling complex plans and challenges immediately – learn on smaller issues (but pick useful ones!)
… make sure that simulated values match reality, for all items that change at all times	… push on with a model that is not proving to be useful – set it aside and look for new opportunities
… focus on successfully tackling the challenge or plan – including steps to remove constraints from balancing feedback and/or enabling reinforcing mechanisms	
… look for opportunities to link the model with pre-existing reporting systems or spreadsheets	
… make the model continually useful, if possible, by updating data period-by-period, revising the model if necessary	