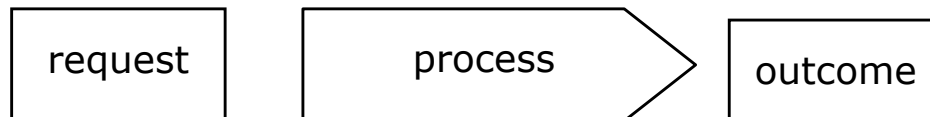


Process modelling

'Process' as 'business process'

The kind of 'process' I want to consider is one which begins with a requirement to carry out a piece of work – typically, but not necessarily, a request from or on behalf of a 'customer' – and ends with an appropriate outcome.



By 'typically, but not necessarily' I mean that a process instigated by a request from a customer is a paradigm case of the kind of process I am talking about. It is this which provides the model.

The 'appropriate outcome' would normally be the successful completion of the piece of work, but could be for example a formal annulment of the original requirement – the cancellation of the original customer's request, for example.

Here are some examples of the kind of thing I have in mind:

- Processing a customer's order to buy something.

- Handling a customer complaint.

- Processing an insurance claim.

- Processing an application to join an organization, open a bank account, invest money, borrow money...

It is easy to see in these examples why the 'process' does not necessarily have only one possible outcome. The goods the customer ordered may or may not exist. The insurance claim may be honoured or rejected. The loan company may accept or reject the loan application. And so on.

Stable state and unstable state

Imagine an organization, or 'department', or 'system', with no other role in life than to carry out a particular type of request or instruction. Imagine it currently has no request to carry out. It has nothing to do. It is doing nothing. We shall call this a 'stable state'.

make work make sense

Now imagine that organization or department or system being hit with a request of a type it is there to carry out. It will need to swing into action and start doing what it knows it has to do to carry out the particular type of request. Person A will do one thing; this will lead to person B doing something else; this will need to be approved by person C; which will then mean that person A can contact the customer and tell him that... etc. And so on until the request is either met or cancelled or rejected in some 'well-formed' way – a way conforming to the rules of the particular process.

When that has happened, stability is restored. The organization is in the same state of stability as it was in before it was hit by the request. The period in between, when person A was doing something, which led person B to do something else and so on, was a period of 'instability'.

This, then, is another way of looking at this kind of business process. It is all the events and activities which take an organization (department, system, etc) from one state of stability to another – from the initial destabilising event through to the restoration of stability.

It is important to note that we are talking here about one 'request', one instance. The process is all the activities and events which need to happen to restore stability in respect of that one initial destabilising event. The complete description of the process may well have to include a number of things which will only happen for some instances and not others. In the insurance claim example, the process is the handling of just one insurance claim. It might have to be described with a lot of ifs: if the claim amount is more than £1000, then inform a claims assessor; if the customer has a no-claims bonus, then calculate the bonus reduction; etc. But the process itself is not a 'batch' thing. It is not for example all the claims of a particular type which happened to come in on one day.

Saying the process is not a batch thing does not mean banishing batch processing completely. Sometimes the 'instance' involves treating a set or batch of things together. An example is fund pricing in the unit trust context. Pricing happens at the level of the fund, and normally at a regular frequency, for example daily. It needs to take into account the current values of all the fund's assets at the time. But it also needs to take into account all the day's deals (inflows and outflows of cash). Pricing cannot happen without 'batch

processing' all the day's deals for the particular fund. But the pricing process itself is still happening at instance level.

Another apparent exception to the 'instance' rule is where an organisation's processes have to fit in with external batch mechanisms, or where reasons of efficiency and financial control take precedence. An example is UK direct debit processing, which is a service cycle spread over a fixed number of days. Subscribers submit files of direct debits and receive a cash credit for the file total.

But the fundamental customer-centric business process is still at instance level – in this case a payment due, for example a monthly premium on a life assurance policy. The 'customer request' is 'collect my premium', and the process is over when this is achieved. It happens to be achieved by making use of batch mechanisms.

Subprocess

Think of a department whose job is to process orders. Orders are coming in all the time from many different customers. It is in a constant state of 'instability' with respect to at least some orders – which will be at many different points. Some will have just come in. Some will be awaiting credit check. Others will be being matched against stock, or waiting for this to happen. Others will be awaiting despatch, and so on.

The stable versus unstable view is useful because you can say the business area 'wants' to be stable. It isn't that it doesn't 'want' to do the things it is in business to do – but it doesn't 'want' to *be doing them*. The less time it spends in doing the thing, the more efficient it is at meeting its customer's needs, and the more customers' needs it can meet. Just as important, the less 'work in progress' it has – the less noise, the less stuff, the less to worry about.

I'm now going to state two assumptions – which may seem obvious – but because of their nature it is important that they are made explicit. The first is that it is a good thing that the department or business is well managed: that order is better than chaos; that it's a good thing to know where things are rather than not to know. The second is that it's a good thing to be customer-centric; to look at things from the customer's point of view; to care about the service we are giving our customers. We are therefore assuming a 'customer-centric'

make work make sense

rather than an 'admin-centric' view: we are not trying to create empires out of filing cabinets and paper clips.

(In a competitive, commercial environment these two assumptions are of course not only obvious, but obviously linked.)

So with these two assumptions in mind, we can now identify some fairly apparent desirable features of this order-processing department. The department will follow rules. It will know how to process different kinds of orders. It will follow set procedures. It will know there are points in the overall process where bottlenecks tend to occur, and it will try to manage its work so as to minimise bottlenecks. It will try to complete each process as fast as possible, so that customer orders are met as quickly as possible. This benefits the customer, who gets what he wants. It also benefits the business as it improves cash flow, and reduces the cost of 'work-in-progress'. The less work-in-progress the less queries, chase-ups, mistakes, and rework.

So we want to get through the process as quickly as possible, and by as standard a route as possible.

Relating all this to the business process of handling an order, it is fairly evident that the overall process (from 'destabilising event' to 'restoration of stability') can normally be broken into a finite series of steps, about which the following are generally true:

The steps would be arranged in a fixed pattern. The most common pattern would be sequential, but there may be occasions where steps can or should happen in parallel.

The boundaries between one step and another would often correspond to hand-offs, or breakpoints where interaction is needed with an external agency. The break points should all be dictated by the business, not by system constraints. They are to do with things like getting necessary input and authorisation.

The boundaries between steps very often correspond to bottlenecks: in a process consisting of steps A, B, C etc the bottlenecks could be described as 'x instances awaiting B'; 'y instances awaiting C' etc.

What constitutes step A, step B etc can normally be described in purely business terms, and the descriptions generally have meaning in relation to what the business process is all about. So one step could be 'authorise order',

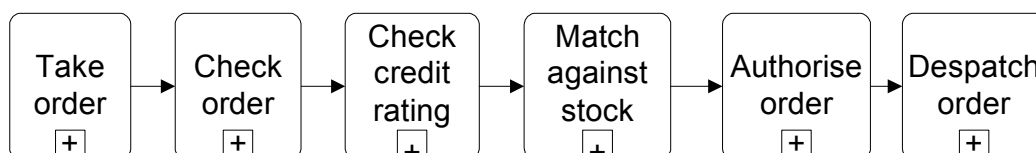
another could be 'match against stock'. These steps, described like this, would need to be carried out regardless of what computer system was used, or whether a computer system was used at all.

The steps are therefore not arbitrary collections of actions, and nor are they events or processes which only have meaning when described in term of a particular computer system. 'Run job C123' is not an example of this sort of step. 'Check customer's credit rating' is.

I am going to use the term 'subprocess' to refer to this kind of step. (I shall explain later why I am using this particular label.)

The concept of 'process' is no less arbitrary than the concept of 'subprocess'. A process is not an arbitrary sequence of actions. It is everything that has to be done (ie all the subprocesses) in order to restore the original stability which the event or request disturbed. If stability has not been restored (if the request has not been met or annulled) then the process has not yet finished.

The diagram below shows an example of an order process divided into subprocesses. As mentioned in the Introduction, the notation is 'BPMN'⁴. But the detailed features of the symbols need not concern us here.



These subprocesses, and their sequence, will not necessarily match every order handling process. It is enough for present purposes that the diagram shows a possible order handling process. The subprocesses required, and their sequence, will depend on the rules of the business – the 'business rules'.

Business rule

This is another concept I want to make comprehensible in purely business terms. Under the heading of 'business rule' I want to include both rules governing the sequence of

⁴ Business Process Modeling Notation (BPMN) specification, Version 1.0, Business Process Management Initiative (BPMI), 3 May 2004.

make work make sense

subprocesses, and also rules as to what happens in a particular subprocess.

So for example there could be business rules stating that after taking the order it needs to be checked; then the customer's credit rating should be checked; then the order should be checked against available stock and so on. Rules like this would be implemented by sequencing the various subprocesses in the overall process. (The rules could for example say that the check against available stock should come before the credit rating check: this would not be an illogical sequence.)

Then there could be business rules within each subprocess. For example:

In the subprocess 'check order':

All orders must be for a known customer.

Items must be identifiable as goods the business trades in.

Quantities must be specified.

...etc.

In the subprocess 'authorise order':

All orders of £100 or less do not need authorisation.

All orders greater than £100 and not greater than £1000 can be authorised by the orders supervisor.

All orders greater than £1000 must be authorised by the orders department manager.

...etc.

Later on I will want to distinguish between two concepts of business rule – the 'raw' business rule, and the 'implemented' business rule. It will be seen that what I am talking about here are 'raw' business rules.

Before leaving this initial introduction to business rules however I need to dispel a possible misunderstanding. I am not saying the process, subprocesses etc come first, and then you decide what the rules are. The definition of the process itself (where it starts and stops) and the analysis into subprocesses (A then B then C) are themselves rules. But by their very nature rules fit inside other rules. There is the rule that you have to do B. Then there are all the rules about how to do B.

Task

The concept of business rule allows us a straightforward way to introduce the concept of 'task' as a further component of 'subprocess'.

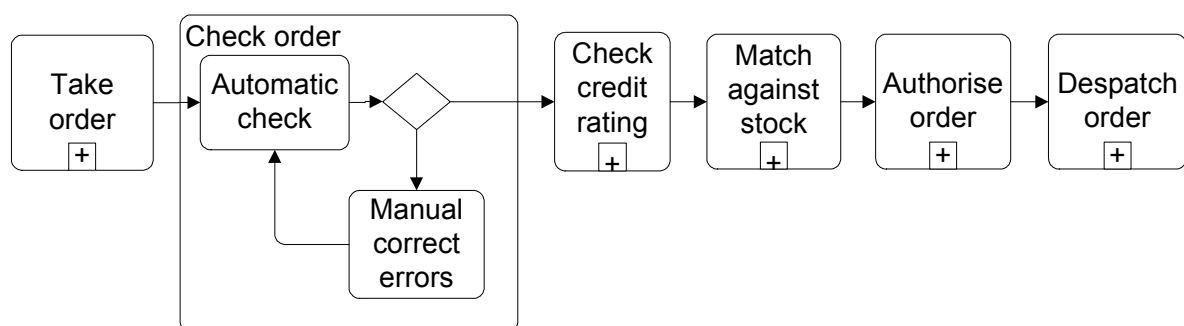
Consider the subprocess 'check order', and the business rules which have been suggested for it. Imagine these rules being run against the order 'automatically' – either by a computer program or by an employee whose job was to validate the order against the rules. It is fairly obvious that all orders would end up in one of two overall categories: those which obeyed the rules and those which did not.

The ones which obeyed the rules could go on to the next subprocess: 'check credit rating'. But what about the ones that didn't pass? Do they just get rejected and discarded? This would clearly be illogical. There may be all sorts of reasons why an order failed the checks – it may have been taken wrongly; it may have gone to the wrong company; it may have one letter missing; etc. In general, errors and anomalies would need to be sorted out 'manually' – by the use of discretion to see if there was a chance of correcting them.

By contrast, applying the rules themselves could be seen as an 'automatic' function.

We can call the two functions 'tasks' – one of which (running the rules and getting the results) is an automatic task, and the other one (correcting the errors) is a manual task. Once the manual task has completed, the order will need to be checked again against the same business rules.

The diagram below shows the process with these tasks in place.



Adding these two tasks to the subprocess 'check order' has some important consequences.

make work make sense

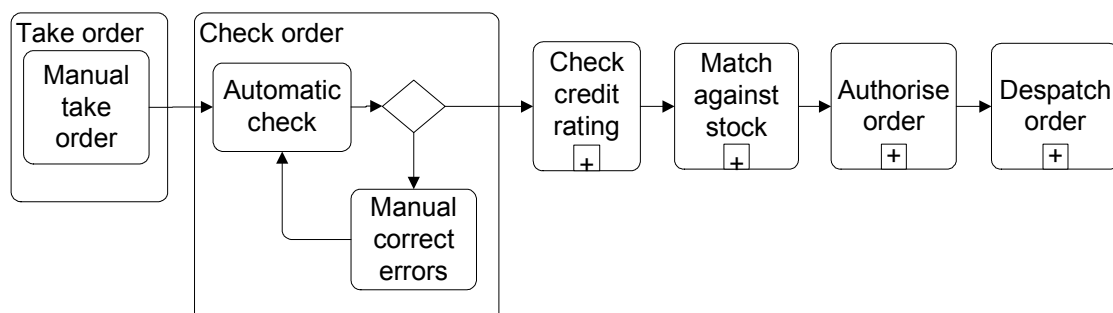
One is that all the work of the 'subprocess' is actually contained within the tasks – the 'subprocess' itself is only really a 'container' for the tasks.

Another is that some orders (the perfectly correct ones) will only need to go through the automatic task. Others (the imperfect ones) will be routed – by the automatic task – down to the manual task. The manual task will then route the order back to the automatic task for rechecking.

Now take an extreme case of bad data – an order which fails just about every rule in task: Automatic check. It obviously cannot get further without every error being corrected. But it did get captured. So the business does know about it. It isn't just sitting around on someone's desk. The order has been rejected as *data*, but it has not been rejected as *work*. This simple feature is hugely important for control and measurement and completeness. The process lets 'garbage in' – but it won't let 'garbage through'.

This concept of task is not arbitrary either. Whether the subprocess is divided into one, two or more tasks is decided by the possible instances going through the process, and the work the process is carrying out.

Take the first subprocess for example: 'take order'. It would be fairly easy to imagine a business where all orders are taken in the same way, and all 'manually' – eg over the phone or by fax or letter. So in this example process the 'take order' subprocess consists of one manual task: 'take order'. See the diagram below:



In this example business there is only one way of taking orders, which is manually, so there is only one (manual) task.

This is a convenient point to introduce another key concept, that of 'workflow status'.

Workflow status

In the example being built up, there is only one way of moving an order to the state of 'awaiting check order', and that is by means of the manual task 'take order'. 'Awaiting check order' just means that the order is at the point in the process where the 'take order' subprocess has happened, but the 'check order' subprocess has not yet happened.

We shall call these states 'workflow statuses', as they are statuses which objects (or instances or requests) have as they pass through the flow of work.

Now take the next subprocess, 'check order'. This is the subprocess that moves the object (in this case an order) from workflow status 'awaiting check order' to workflow status 'awaiting check credit rating'. But as we saw above, some orders (perfectly correct ones) will only need to pass through the automatic task, whereas others (imperfect ones) will go first through the automatic task, then the manual task, then the automatic task again for rechecking. (This is of course assuming the order is perfectly correct the second time: it might go round the loop again.)

So, as mentioned above, this shows that the concept of 'task' is not arbitrary. It is not that the two tasks making up the 'check order' subprocess just represent approximately half of the work needed to move an order to the next workflow status. It is to do with the logical possibilities arising from the objects themselves, and what needs to be done to move them on to the next workflow status.

Although the examples of workflow status given so far are at 'subprocess' level, there is no reason why we should discount workflow statuses at 'task' level as well. We could describe an order which has been through the automatic check task, has had some errors detected, and is now waiting for someone to correct them in 'manual correct errors', as being at workflow status 'awaiting manual correct errors'.

Which brings us neatly to another piece of jargon.

Subject entity

We have actually introduced this concept already. It is the 'object' or 'request' which moves through the process. In the

make work make sense

current example, it is the order. I now want to call it by a specific name: 'subject entity', and define it a bit more formally.

The 'subject entity' is the object or request going through the process. It is this which has the 'workflow status', and for each instance of a process (eg an order process or an insurance claim process), there can only be one subject entity instance – eg order number 123, or claim form 456.

The subject entity is typically a request for something to be done – either implicitly or explicitly. The 'something to be done' is typically the thing it is the job of the process to carry out. (So for example a 'claim' would be the subject entity of an insurance claim process.) It may be identified with or linked to a physical object – for example a life assurance proposal form or a claim form – but it does not have to be.

It may also be identified with or linked to a data record or set of data records (a 'data set'), eg a policy application, or the details of the funds and/or amounts which need to be switched in a switch request. It is probably difficult to imagine how it could not be so linked to a set of data. But the subject entity is not primarily a collection of data. It is primarily an instance of work, a request to carry something out.

Further task analysis

Now let's break the next subprocess ('check credit rating') down into possible tasks. Again this will depend on the business rules. For the sake of argument we shall say the rules are these:

- 1 If the customer has sent cash with the order, then pass credit check.
- 2 If the customer has not sent cash with the order, and the amount of the order is less than or equal to the total current unused credit, and the customer is not in arrears, then pass credit check.
- 3 If the customer has not sent cash with the order, and the customer is not in arrears, but the amount of the order is greater than the total current unused credit, and the customer has enclosed a bank reference justifying the credit increase, then the credit increase needs to be manually approved.
- 4 If the customer has not sent cash with the order, and the amount of the order is less than or equal to the total current unused credit, but the customer is currently in

arrears, then write to the customer to request payment before accepting the order.

- 5 If the customer has not sent cash with the order, and the customer is not in arrears, but the amount of the order is greater than the total current unused credit, and the customer has not enclosed a bank reference justifying the credit increase, then write to the customer to request a bank reference justifying the credit increase payment before accepting the order.

...and so on.

Some of these rules allow the subprocess to be passed automatically – for example if the order (subject entity) meets the criteria of either rule 1 or rule 2.

Because of rule 3, the subprocess would need to make allowance for manual approval before a subject entity meeting that set of conditions can pass to the next subprocess.

Rules 4 and 5 require the subprocess to accommodate writing to the customer for additional documentation or action.

Because of this, the subprocess would also need to allow for both recording and acting on the customer's reply, or, if the customer does not reply, some sort of follow-up. (Events and conditions like these are familiar office administration procedures.)

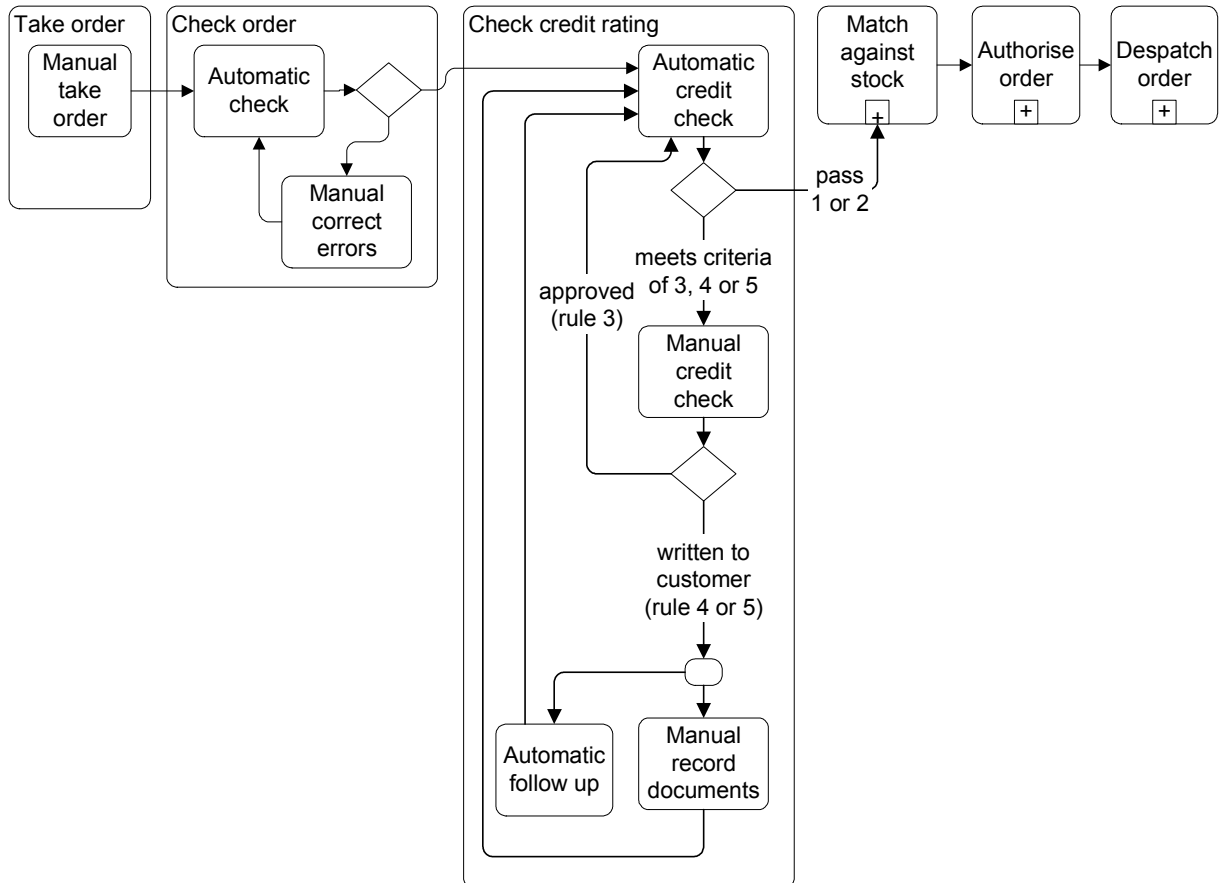
The subprocess could be modelled as in the diagram on the next page.

There could be an automatic task to apply rules 1, 2, 3, 4, 5 etc. If the order passes either 1 or 2, then it goes to the next subprocess, with workflow status 'awaiting match against stock'.

If the order meets the criteria described in rule 3, then the order would need to be routed to a manual task to allow the increased credit to be approved.

To keep it simple, it will be assumed that the same manual task could allow for both approval (rule 3), and also writing to the customer (rules 4 and 5). So if the order met the criteria for either rule 4 or rule 5, the automatic task would also route it to the same manual task.

make work make sense



In the case of just approval (rule 3), then the order would need to route back to the automatic task, which would allow the order to pass to the next subprocess. (In theory this action could be taken in the manual task. Arguments for the approach suggested here will be presented later, when we get on to system design.)

If the customer needs to be written to (rules 4 and 5), then we need two additional tasks: a manual task for recording and acting on the customer's reply; and an automatic follow-up task to send the order back round the loop again if the customer does not reply after n days.

To keep things simple, I have not kept to strict BPMN notation at this point. And the details of the tasks need not concern us over-much. The precise number, nature and configuration of tasks and the routing between them will be determined by the business rules governing the process itself. Enough should have been explained to establish the principles of the process-modelling approach presented here.

Process model

So far we have only looked at one example business process. The business or administration operation the process belongs to is unlikely to have just one process. Instead there are likely to be a number of processes, with interactions, dependencies and all sorts of other relationships between them.

For example a trading organization which is likely to have an order-handling process is also likely to have:

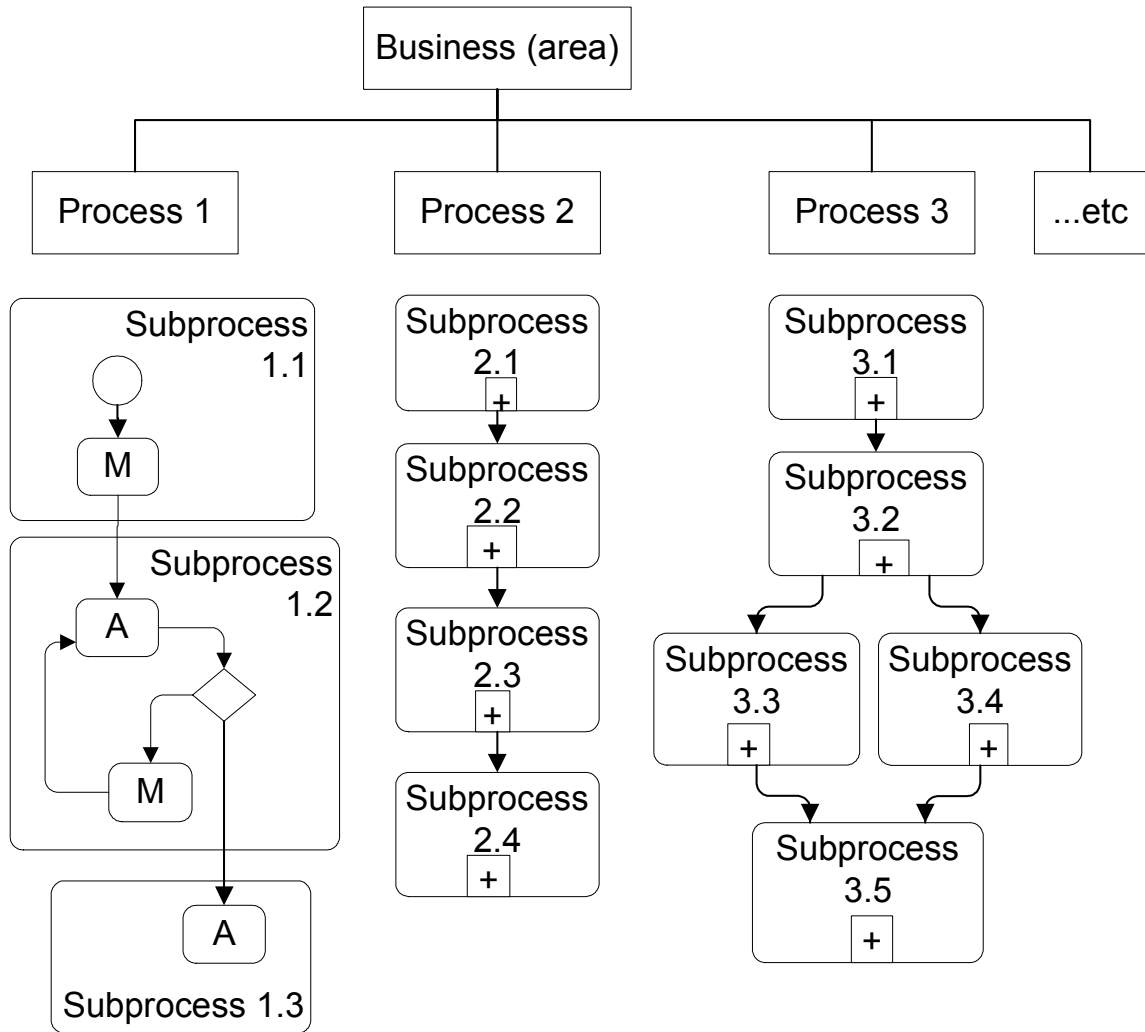
- a process to record details of new customers;
- a process handling changes in customer data (bank details, address changes etc);
- billing and settlement processes;
- bad debt recovery processes;
- a process for ordering stock and raw materials;
- a payment process;
- ...and so on.

Many people working for the organization will be involved in more than one process, and many of the key entities (customer, stock item, sales representative etc) will be relevant to more than one process.

Our approach however assumes that a business can be logically analysed into a finite set of processes, and that, following the rules of the business, each process can be analysed into a finite set of subprocesses, and then each subprocess can be analysed into a finite set of tasks.

This is what is meant by a 'process model' of the business or business area. See diagram below for a schematic example:

make work make sense



I would now like to consider what having such a process model means for a business. We have defined a finite set of business processes, which are typically the carrying out of requests. Since this finite set of processes just is what the organization does, then the carrying out of these requests is what the organization is there to do. We have broken down each of the processes into their logical constituent subprocesses, and then identified what all the business rules are for each subprocess. We have used the business rules to analyse each subprocess into the correct number and type of tasks and determine the routing between one task and the next.

The next thing is consider all the interactions between one process and another: how one process may initiate another; how one process may terminate another; how an action taken or an event occurring in one process could – perhaps via business rules – determine the outcome of another process.

Imagine we have done all this, and written it up in the form of diagrams and text. We would now have a complete description of how that business or business area functions. The modelling approach presented here would obviously be of value to a business analyst. But there is more to it than that. We shall see important implications for computer system design, change implementation, change management and business management itself.

But we must not run before we can walk. We need to lay some more foundations, in the form of an approach to system design based on the process modelling approach already described. But even before that I want to take a short but important detour, which is about the relationship between the process modelling approach presented here and classic business process re-engineering (BPR).