

Separating Analysis from Design

(decide what's needed before deciding how to implement it)

Brian Cooney, Principal Instructor
IRM Training Pty Ltd ABN 56 007 219 589
Suite 209, 620 St Kilda Rd, Melbourne, Vic. 3004, Australia
03 9533 2300
training@irm.com.au

Successful projects solve business problems

Looking at what business objectives you are trying to satisfy before leaping into the technology enables you to use the technology wisely, manage scope and cut costs, producing systems which work for your clients.

It's easy to concentrate on the technical features of any project and lose sight of the reason for its existence. Every project exists to solve a problem. Either what you have doesn't work well enough and needs improving, or you need to invent something totally new. After all, "If it ain't broke, don't fix it."

Too often the pressures of deadlines and budgets lead us to bypass the important process of analysing business needs, and we leap straight into the technology. But how often do we find that the system doesn't do exactly what's required. Users are obliged to use workarounds, reducing some of the benefits we were supposed to provide which also affects our credibility. How often do we need to invest extra time and expense in providing Version 2 (and 3 and 4 and ...), when some careful work might have exposed the real needs earlier?

The classic waterfall lifecycle

There are many variations on the System Development Life Cycle (SDLC) and the following may not be exactly the terminology which your methodology uses. The content of the phases is certainly simplified in this table, but projects follow the following phases:

	PROCESS	DATA	DELIVERABLE
INITIATE	BUSINESS REQUIREMENT		<i>Terms of Reference</i>
ANALYSE	Business Functions	Entities	<i>Business Specification</i>
DESIGN	Program Specifications	Database Design	<i>Technical Specification</i>
BUILD	Programs	Database	<i>Tested System</i>
IMPLEMENT	OPERATIONAL SYSTEM		<i>Post-Implementation Review</i>

The output from each phase is the input to the next. Garbage in, garbage out. Or to put it a little more elegantly, you can't make a silk purse from a sow's ear. You can't implement a good

operational system without a good tested system, you can't build a good system without good technical specifications and you can't design a good system without a good business specification. Ensure that talented people are used early in your project, to ensure that you can achieve a good output at the end.

Don't fool yourself into thinking you are doing anybody any favours by bypassing business analysis and moving straight to design, or by combining the two. If the deadline is tight, negotiate a phased implementation. If the budget is tight, remove some functionality.

Many surveys, including the work of Boehm, have shown consistently that the increase in cost of repairing errors increases exponentially the later that repair takes place in the SDLC. An error which costs a dollar to repair during the Initiate phase will cost \$10 to fix if it is left until Analysis, \$100 at Design, and \$1000 if nothing is done until Implementation. Yet how often do we say "We'll be able to fix that in the code"?

Software tools can't think for you

Would Shakespeare have been a better writer if he had a word processor?

Any tool will believe what you tell it. There are some terrific tools around which will take the mindless hack work out of your job so that you can concentrate on doing what you do best. But don't think that if you have the best tool possible your systems will be the best possible. If you don't believe me, grab your favourite word processor and write a sonnet which people will be delighted to quote 400 years from now.

You still need to do your own thinking. That's what business analysis is all about.

Separate business analysis from technical design

Business analysis is about thinking *what* your solution should do, while design is about *how* to make it happen using the technology available. Bearing this in mind will make it easier to avoid blending these two phases. *What* and *How*. This will help you to build more robust systems.

What your business does doesn't change as much as *how* it gets done. Technology changes more quickly than business needs. Companies restructure. But what they do hasn't changed, just which individual does it. Business analysis gives a logical view of process and data needs and is not bound by physical implementation.

IT specialists and business users regularly complain about their misunderstandings. Of course they don't understand each other. The technical people are concerned with making best use of their technology, the users are concerned with achieving their business goals. Business analysis should neatly bridge the gap.

The Terms of Reference produced during the Initiate phase should include, unambiguously, the Problem to be solved, the Objective to be achieved in order to solve that problem, Constraints and Scope.

The purpose of Business analysis is to:

- find the cause of the problem – after all, people are really describing symptoms, not problems

- look at alternative solutions which will achieve the objective
- pick the most acceptable solution
- document that solution in detail, so that no ambiguities are passed to the design team

Techniques of business analysis

In order to dig beneath problems and find solutions, many techniques are used which are beyond the scope of this paper. They include carefully structured interviews, document gathering, brainstorming, risk analysis, cost-benefit analysis and many more.

Tools of business analysis

Only four tools are needed for Analysis: two for Processes and two for Data.

For processes, the Dataflow Diagram (DFD) shows the business processes in a system and their connections to each other, while the business specifications show the business rules for each process. For data, the Entity-Relationship Diagram (ER Diagram) shows the things which need to be stored from a business perspective, and the Data Dictionary details the specific items (fields) needed for each entity and their connections to dataflows on the DFD.

The purpose of this paper is not to show how to construct these models, but to demonstrate the importance of their use in a business model rather than a technical model.

Any idiot can draw an ER diagram

This is true, but the quality of the resultant model often leaves much to be desired. Reflect business needs in the model and leave nothing to chance. Be careful with such things as mandatory and optional relationships and data items, as your final system must cater for 100% of business requirements, not the most likely events. Never try to combine documenting business data requirements with a database design.

If the business model is accurate, the design will be easy to produce, and can be easily modified in light of business requirements and new technology.

Why a dataflow diagram is better than a function hierarchy

A function hierarchy shows the major functions in a system and drills down to their detail, but shows nothing of data requirements for those processes. It's very easy to inadvertently omit lower level processes. A function hierarchy is not immediately obvious to a user who needs to double-check the accuracy of your model.

A dataflow diagram similarly shows the major functions and drills down, but as the dataflows act as the "glue" which holds the processes together, it is simple to construct such a diagram in collaboration with a business representative. "What information do you need to perform this

process? Where does it come from? What information is produced? Where does it go to?” In this way the individual processes can be gradually added to the model and nothing will be missed. Complex processes can be drilled down and related processes can be grouped together.

A large dataflow diagram is difficult to read and difficult to maintain, so it loses its effectiveness as a communication tool with both business users and the technical team. Instead, have about seven processes per diagram and drill down to more detailed diagrams, with a numbering system to cross-reference. For example, the details of process 3 will be numbered 3.1, 3.2, 3.3 etc and will appear on diagram number 3. Then the high level diagrams can be used for discussion with management and the lower level diagrams can be used for detailed double-checking with users who are expert in that area.

The function hierarchy now exists automatically. There is no need to draw it separately. Many analysts prefer to use a process model which includes “swim lanes” for identifying *who* performs each process. This is simply a dataflow diagram in a different format. It’s fine to begin analysis using such a diagram, but by the end of analysis the swim lanes should have disappeared, as your model should be resilient enough to survive restructures in the business. You are documenting what needs to be done, not who does it or how.

The dataflows should be specified with their detailed contents in the data dictionary, as should the datastores. Gradually the dataflow diagram will help to build up the data model so that your data definitions will support the processes.

Include all processes in the dataflow diagram, even the ones which seem trivial. I heard of a project recently which fell into the trap of “Reporting is easy, we’ll leave that until the end.” At build time, the data required for reports proved to be inaccessible.

Multiple versions of the model

Document the existing system first, including all its faults, with no improvements added. The cause of the problems is in there somewhere and you should not jump to conclusions about the best way to solve it.

Remove the *how* from the current physical model, giving a current logical model. Then work out the best way to solve the problems and include new requirements. This gives the new logical model, ready for design.

In some cases problems can be solved by simply rearranging work practices, saving unnecessary expense in design. Does this do the technical team out of a job? Not at all. They can be released to use their expertise where it will make most difference on other projects.

Writing the spec

Do not write a single large specification. It will be wrong, as you will miss some detail and will be unable to easily modify it. And nobody will be able to memorise the whole thing. Instead write a single specification for each detailed process – a “mini-spec”. Each can be double-checked with the appropriate business expert. The collection of all these mini-specs is your specification for the whole system. Each is individually easy to maintain without affecting the rest of the specification.

Specs should not include any technical information – simply the business rules for that process. The design team can work out the best way of making it happen.

What to automate?

Looking at the dataflow diagrams with the business representatives and the technical team will enable you to decide on the best implementation. This is based on such factors as what is acceptable to the business, what is technically possible, what fits with constraints of budget and time.

You may decide on a complete automation, partial automation, phased implementation or some other great idea. Then hand over to the design team.

Combining analysis and design

Don't ever do it. You are saving nothing.

Brian Cooney is principal instructor at IRM Training and has worked in IT for over 20 years both for end-users and software vendors. He regularly presents workshops in Business Analysis, Requirements Investigation, Project Management, Data Modelling and has trained hundreds of business analysts from many of Australia's top organisations. Brian is an Accredited Professional Speaker and qualified instructor – his workshops are educational, entertaining and consistently popular. For workshop details please visit www.irm.com.au