IDIOM

a new approach to systems development:

Requirements and the beast of complexity

by Mark Norton, April 2010

Requirements
and the beast of complexity

Contents

Executive summary

The IT industry continues to earn its reputation for poor development performance and ROI in the minds of many business executives. Roger Sessions, in his thought provoking paper "The IT Complexity Crisis: Danger and Opportunity"[1], describes an IT industry in deep distress – and he claims that it is getting worse.

In **Part 1** author Mark Norton outlines reasons why the industry standard approach to requirements analysis and specification is a significant contributor to poor IT development performance. In particular, the current approach does not result in requirements that can be shown to be complete, correct, or consistent. Furthermore, extensive use of narrative to specify requirements creates opportunities for downstream misinterpretation and obfuscation.

Part 2 then proposes some characteristics that are seen as desirable in any upgraded or replacement requirements approach that seeks to address these fundamental weaknesses.

Part 3 develops these characteristics into a new requirements approach. It focuses on decision making as a core business competence that can be readily captured, tested for completeness, correctness, and consistency, and then used to drive systems development projects.

The umbrella term for this approach is decisioning.

The concept of 'decisioning' recognizes the fundamental importance of decision-making to any organization.
An organization's decision-making is the proprietary know-how that drives value and differentiates one organization from another in the marketplace. Because of this fundamental importance, decision-making policy is usually apparent at the strategy level, from where it can be distilled into precise 'decision models'.
This article provides insights into decisioning as a new approach for the capture of these decision models. The models are an independent, auditable, and electronically testable specification of business policy. The models can also be rendered into business accessible structured narrative, and simultaneously into computer accessible source code, to provide a live, persistent, and direct connection between business policy and operational computer systems.
The nature and role of decision models as the critical link between business policy and computer systems allows them to provide essential context for systems development projects.
When used with an appropriate methodology this context acts as a guide-wire for project development, reducing project risk, cost and time by significant margins.
And because they are directly accessible by the organization's domain experts, decision models can be used by the business to control automated system responses on a day-by-day basis – analogous to a remote control for the organization's computer systems.

1 & 2. http://www.objectwatch.com/whitepapers/ITComplexityWhitePaper.pdf
3. http://www.theiiba.org/AM/Template.cfm?Section=Body_of_Knowledge
4. http://en.wikipedia.org/wiki/Requirement
5. Grammar: A group of words containing a subject and a predicate and forming part of a compound or complex sentence.

Requirements
and the beast of complexity

Part 1.
The problem

Roger Sessions, in his thought provoking paper "The IT Complexity Crisis: Danger and Opportunity"[2], describes an IT industry in deep distress. Roger lays the blame for poor, even catastrophic, IT productivity at the door of complexity. He also acknowledges that 'some have suggested the culprit is poor communications between business and IT'. This article endorses both of these viewpoints and promotes a new approach to requirements management that reduces project complexity and improves communication between business and IT. This new approach can be used on its own, or as a supplement or precursor to existing approaches. Critical features of the approach are: detachment of business requirements from individual projects; and the production of testable requirements that can be shown to be complete, consistent, and correct prior to use within the SDLC.

What is a requirement?

BABOK® Guide[3], Version 2.0, states:

> **A requirement is:**
> **1.** A condition or capability needed by a stakeholder to solve a problem or achieve an objective.
> **2.** A condition or capability that must be met or possessed by a solution or solution component to satisfy a contract, standard, specification, or other formally imposed documents.
> **3.** A documented representation of a condition or capability as in (1) or (2).

From Wikipedia[4]:

> *"In engineering, a requirement is a singular documented need of what a particular product or service should be or do. It is most commonly used in a formal sense in systems engineering or software engineering. It is a statement that identifies a necessary attribute, capability, characteristic, or quality of a system in order for it to have value and utility to a user."*

These definitions are representative of many attempts to define the term 'requirement'. We can make some initial observations about a 'requirement' as defined by these definitions:

- A requirement is generally assumed to be a document, with use of a natural language narrative implied;
- The requirement narrative could describe any one of a wide variety of loosely defined concepts including a condition, capability, product, service, attribute, characteristic or quality.
- Some concepts in the above, if used on their own, risk being meaningless (a quality? a characteristic?), which implies that there is a larger collective unit, but this assumed unit is not directly addressed by the definitions.

We can assert that there is an implied reliance on natural language rules to aggregate the low level requirements concepts listed above into bigger linguistically valid units, also called requirements regardless of their size and scope. How big can a 'requirement' be? A requirement as described above could refer to anything from a clause[5] to a standard the size of War and Peace (e.g. 'the system must comply with the GAAP'). The use of the word 'singular' in the Wikipedia definition (and implied in BABOK) is meaningless when the unit itself is a collective term of unbounded size and scope.

The problem with narrative

To demonstrate how nebulous these definitions are, we suggest that you try taking a requirements document and actually listing each individual 'requirement' as per the definitions above (i.e. each quality, each

characteristic, etc). This is a surprisingly difficult task. Identifying each individual requirement and ensuring that it is unique within a narrative is a challenge made even more difficult by the 'idiom effect' – the idiom effect is the localization and personalization of terms used in the narrative by the author. You are also likely to find that in order to classify individual requirements as proposed, you will end up creating your own idioms – idioms are phrases that have only local, even personal, meaning.

A dearth of standards for writing requirements in narrative form means that requirements specifications (especially 'business requirements') are frequently reduced to a series of personal idioms. Most practitioners develop their own requirements writing idioms, which usually change over time, sometimes within a single document. Even when carefully written, English is highly nuanced; ambiguity and imprecision are typical and endemic. When compounded with a variety of personal meanings and interpretations, the majority of requirements specifications are imprecise, no matter how careful the author. Therefore a consequence of capturing requirements in narrative is to create a new class of analysis problem for the downstream developer – analyzing the requirements analysis. And because the requirements document involves some degree of personal abstraction, there will be misinterpretations (the chance of another brain exactly interpreting a narrative requirements document as intended is implausible). This also often gives rise to tensions between the analysis and downstream disciplines, further aggravating the situation.

The network effect

A further cause of requirements induced complexity, and the reason that the larger projects suffer more, is the network effect .

The network effect is an exponential growth in the number of connections in a network as the number of nodes increases.

For requirements, this means that as the number of individual requirements increases the overall complexity increases at a compounded rate because of a dependency network between the requirements. The term requirement is usually used in its plural form – value can only be realized when each and every requirement in the solution set is eventually built exactly as it needs to be, and that it interacts with or supports its peer, ancestor, descendent and derivative requirements exactly as required. This network effect between individual requirements significantly compounds the requirements communication and complexity overhead.

T Capers-Jones[6] has documented this effect over tens of thousands of projects, and provides overwhelming evidence that size is the basis for an exponential growth in cost, time, and risk.

Compounding the complexity through the SDLC

Requirements as defined above do not represent a useful endpoint in their own right – the requirements must be transformed into a solution if their value is to be harvested. So the somewhat ambiguous requirements starting point described above is transformed through the series of complex transformations that is today's Software Development Life Cycle (SDLC). In the typical SDLC there are three major manual transformations required to turn a raw or 'stated' requirement into a solution – and as noted, the requirement cannot contribute to utility or value until it is delivered within a solution. The three transformations are:

- The domain 'need' or want is translated into a documented 'requirements' specification;
- Documented requirements are translated into technical specifications;
- Technical specifications are translated into computer languages.

A fourth transformation, from computer language into executable, is usually fully automated and is in a different class to the above three.

These manual transformations share some characteristics:

- Each source specification may be transformed into multiple target specifications and vice versa (i.e. the transformation is often many-to-many);
- No transformation retains complete traceability between source and target specifications;
- The target specification is typically not able to be fully interpreted by the source specification author – direct manual reconciliation is not normally viable;
- An atomic requirement or specification at any of these levels is meaningless – a requirement or technical specification at any level must be part of a community of related specifications, so that the entire set of requirements must be accurately transformed together as a single unit.
- None of the sets of specifications at any of the stages can be empirically tested and proven to be consistent, correct, and complete (the 3 C's).

If requirements are the answer, what is the question?

Given that the requirements specification is the essential link between business value and the eventual solution as represented by an operational system, it should not be surprising that the loose original definition of a requirement specification compounded by multiple un-reconcilable many-to-many transformations gives rise to cost, risk, and delay.

This article Part 1 has argued that 'requirements' are the Achilles heel of the SDLC, a weak link that in its current form provides a poor foundation for the more mechanical but still expensive downstream development phases. This weakness stems from:

- An imprecise definition of what constitutes a 'requirement', and one that is open to a variety of personal interpretations;
- Too much reliance on unstructured narrative documentation;
- Assuming business requirements specification to be an SDLC project task – in fact, business requirements are entirely independent of any given project.
- Detachment of requirements specification from requirements solution – creating a need for complex handovers that include transformations/translations, and corresponding opportunities for obfuscation and introduced errors.

Returning to our original BABOK and Wikipedia definitions, we can also assert that the requirement should:

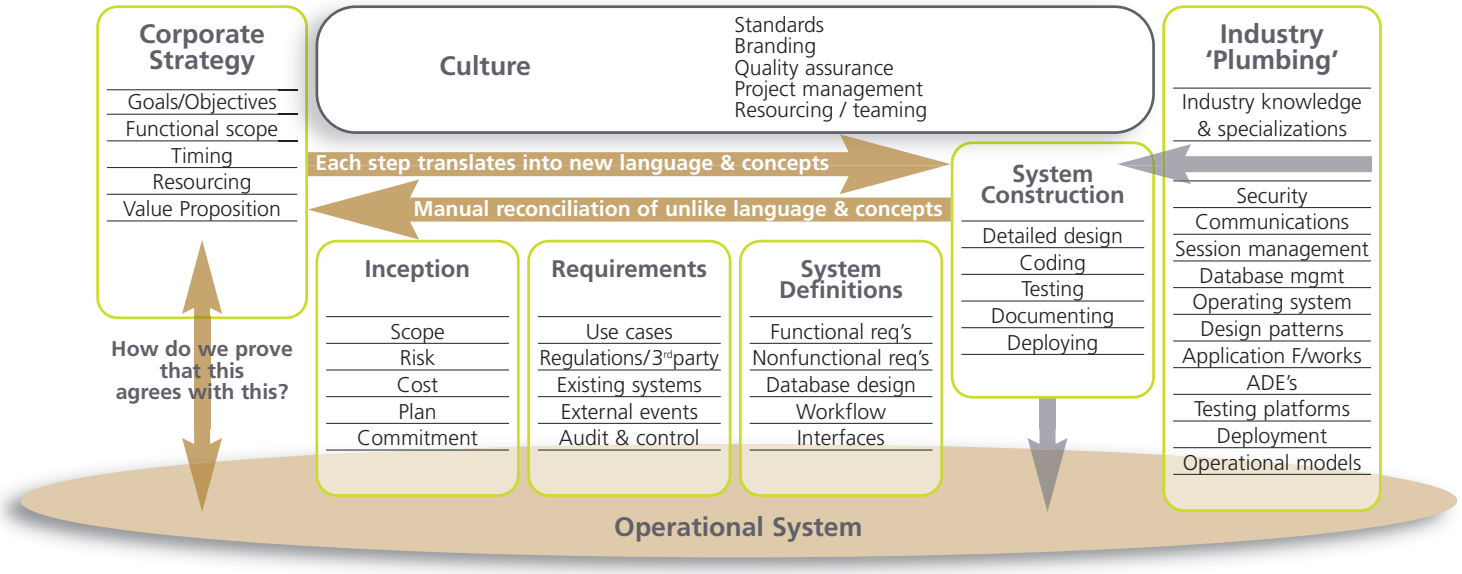
- be anchored in a business purpose; and
- contribute to utility or value.

In Part 2 we will discuss how these two requirements characteristics provide a basis for resolving the requirements dilemma – provided that we can address the weaknesses above.

---

6. http://en.wikipedia.org/wiki/Capers_Jones

Figure 1. **The Traditional SDLC**



Part 2. The solution

Elements of the solution

In order to address the weaknesses identified in Part 1, our solution to the requirements dilemma will need to make a few changes to the traditional requirements approach:

Desired Change	Effect
Reduce or remove the dependency on narrative.	Reduce ambiguity. Remove need for reinterpretation.
Explicitly define all proper terms and the relationships between the terms – a complex, structured, and active glossary.	Reduce the 'idiom effect'. Provide a common language between business and IT.
Identify, define, and structure the 'business policy' motivated actions that modify the entities described by the terms.	Capture and bind into the solution strategic business directives that are active within the requirements scope – thereby aligning value change mechanisms with strategy.
Integrate the analysis and solution phases.	No handovers. Reduce cost, risk, time and other negative impacts of transformation.
Test the requirements.	Early (pre development) confirmation of consistency, completeness, correctness to reduce risk.

At first glance this might look like a prescription for an SDLC – it is not. The above activities will not:

- Produce an orchestrated set of processes.
- Interface to existing processes, people or devices.
- Acquire and/or persist any artifact.

These excluded outcomes are the key – they are the elements of traditional requirements that bind the requirements to both technology and platform. Because these are explicitly NOT present, we can deal with the requirements in a technology agnostic manner. In other words, these requirements now represent purely business rather than systems intent. Of course, it also means that we cannot produce an active system directly from the requirements that are defined in this manner.

In which case you might ask – is it worth doing? The answer is an emphatic yes! We will expand on the benefits shortly.

All requirements are equal, but some are more equal than others

Obviously businesses planning documents are not going to help us determine the preferred activities and sequence of some low level process, or the existence of fields on a screen, or the layout of a document, or any other of a myriad of low level details that often fall under the requirements umbrella.

But we can infer, if they are not already explicitly defined, the core value transformations that the business anticipates as part of its strategic intent. An insurance company will define what risks it wants to cover, what market segments it will solicit, what terms and conditions it will offer and at what price.

This will be done at a business planning level, and it is not dependent on a systems development project (although it might give rise to one). Similarly, a hospital will determine what treatment specialties it will offer, with what levels of pre and post care, to what customer/patient types, and at what cost.

In fact, all organizations are predicated on a declared premise that they act upon, and most importantly, add value to their subject entities.

Given the fundamental nature of this assertion, we can say that capturing the details of how an organization proposes to add value to these core entities is the essence of its value proposition, and provided that this information is captured correctly, is correct by definition. In fact, if we can capture this value proposition in a testable form then we might even assist the business to clarify and confirm its strategic intent.

Furthermore, if we capture the value proposition using a pre-defined 'idiom' – in this case, a language that is locally defined by and for the business value proposition itself – and we are able to empirically test and prove the proposition, then we have produced an artifact that can underpin an SDLC without the cost of transformation.

So we have two levels of requirements in play:

- those that are business strategy driven, and which must by definition be addressed by any solution that is used by the business; and
- those that are solution specific and subservient to the strategic requirements, adding detail in the context of a specific solution but never altering the strategic intent.

In fact, we argue that these two types of requirements represent quite different concepts. The former arise from business planning independently of any IT driven SDLC, and the latter can be bundled into technical design phases within the SDLC as design detail.

We will now refer to these as **business requirements** and **design requirements** respectively.

Together they more or less replace the traditional project requirements phase, whose primary remaining task is to define the scope of any specific SDLC project. The design requirements we will now ignore – they cannot by definition validly change or conflict with the business requirements, and can be easily obtained within the downstream technical design phase through RAD or traditional development approaches.

Benefits of business requirements living outside of the SDLC

By focusing on the business requirements exclusively and independently of any SDLC as proposed, we can realize many benefits:

- Provide a domain specific idiom that is business aligned and at the same time understandable by IT;
- Provide provably relevant business requirements to scope and de-risk all downstream effort, particularly systems development projects;
- Detaching business requirements from system development projects allows for an independent existence and strategic continuity – the business requirement does not change unless the business does (by definition), but the systems supporting that requirement may change with every new technology generation, and/or with multiple simultaneous deployment options and strategies;
- Reducing the size and increasing the focus of the study area offers significant (and potentially exponential, given the network effect) reduction in cost, time, and risk for the equivalent analysis done as part of a larger systems project;
- Ensure the alignment of all downstream activities that derive from the business requirements.

In addition, the business itself can use the business requirements phase to clarify, confirm, and prove the assumptions underlying the business intent.

Capturing the business requirements

The proposed approach implies a significant makeover of the traditional requirements process.

There is an existing link between the traditional view of requirements as outlined in Part 1 and the elements of the solution identified above – the concepts of **business purpose** and **value**.

It is intuitively appealing to use these as a basis for a requirements approach makeover, and these shared fundamentals would also no doubt be reassuring to the business leadership – the alignment of systems with business strategy regularly features as a priority in executive polling. We will now describe a requirements process that derives directly from analysis of these business fundamentals.

We have identified two core elements of a new requirements approach:

- A description of the relevant subject entities using prescribed terms and a model of the relationships between the terms; and
- A description of the activities that modify those terms in order to give effect to the business value proposition, and a model of the relationships between those activities.

We will refer to these two sets of descriptions/models as the **Fact Model** and the **Decision Model** respectively. Let's look a bit closer at these terms and what they mean, as their usage within this approach may vary from traditional concepts.

The Fact Model (a term attributed to Tony Morgan[8]) is a form of data model. Note that while the Fact Model will eventually be a localized view of the domain model, it should NOT be simply extracted unquestioned from an existing domain model – it is important that the content and structure of the Fact Model be driven by the adjacent Decision Model requirements in the first instance. Of course the Fact Model must be derivable from the eventual implemented domain model otherwise the business cannot support its decisions by

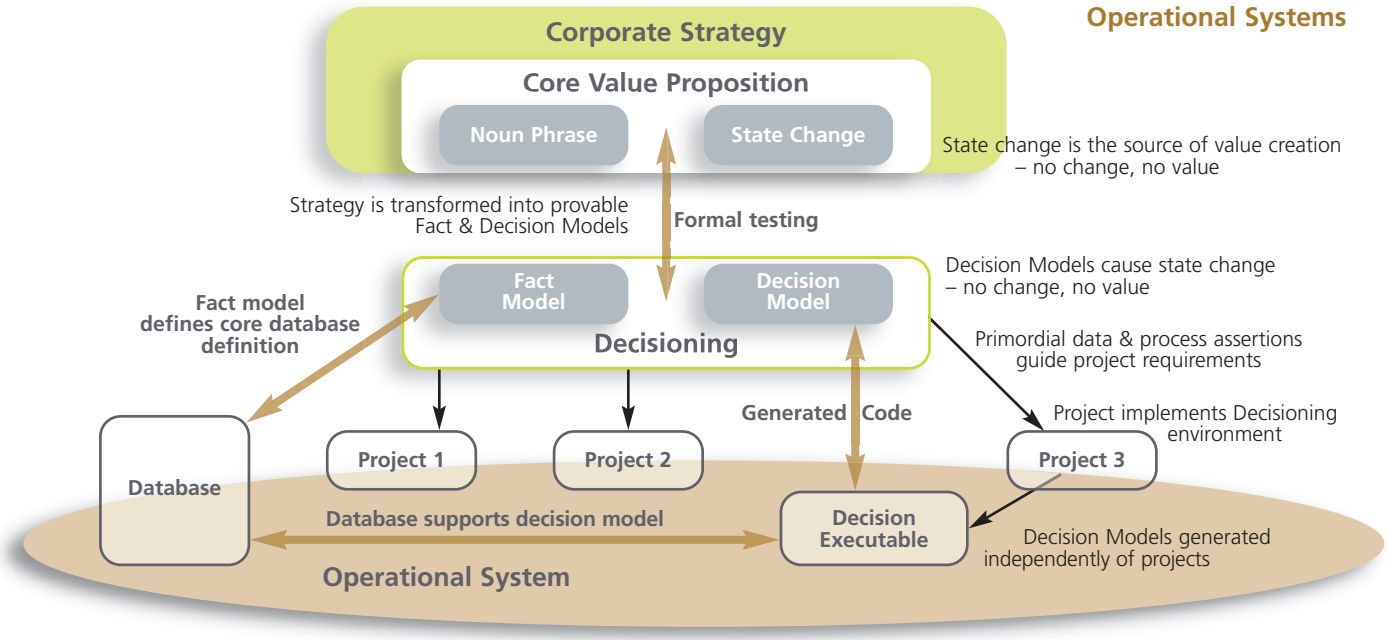
7 & 8. Morgan, Tony. "Business Rules and Information Systems: Aligning IT with Business Goals" (2002), ISBN 0-201-74391-4

definition. But if it is developed as proposed in response to decision analysis then it can be used to either help derive a new domain model, or validate an existing domain model; whereas if it is extracted from the domain model on an assumption that the domain model is already correct (in which case, how was this assumption validated?), then this ability to drive domain model derivation or validation is compromised and we lose a major benefit of the proposed approach.
The Fact Model is in fact a transactional view of the required data, and any modeling technique used should acknowledge the need for a transactional context i.e. a root. For this reason the W3C Schema standard (.xsd) is a particularly suitable documentation standard and this article will assume that the Fact Model is represented by one or more XML schemas.
The Decision Model is likely to be a less familiar concept. This article proposes the term 'decision' to describe the activity component of the business requirement because a business activity that adds value to the subject always involves some discretionary consideration by the business – this implies that a decision must be made. If a value change is not discretionary in any way then we are simply capturing the new value, and by definition, the value change is not derived from the business value proposition. For instance, capturing the change in price of a commodity is not discretionary and does not create new business value; whereas revaluing a portfolio or selling a quantity of the commodity because of the change in price is discretionary and creates value for the business. Of course non-discretionary or 'decision-less' activities may still be valid requirements, but they represent cost rather than value creating activities of the organization and can be relegated to SDLC level analysis whenever a project addresses the relevant domain. In an ideal world, they might even be (software) commodities themselves.
While a range of standard methods exist for declaring decision logic, we must be careful that the method used meets the following 'decision centric' guidelines if it is to be useful to the proposed approach:

- The Decision Model must fully address the act of creating value as defined by the business, where value is defined as a forward looking change in the state of an entity that assists the business to achieve its objectives – there is no point in making half a decision!
- The Decision Model must interface to the Fact Model to deposit the results of the decisions made, and to acquire data as required by the decision logic.
- The Decision Model must only use the terms defined by the Fact Model in its descriptions.
- The Decision Model must be testable. If the validity of the Decision Model (and by inference the associated Fact Model) cannot be empirically demonstrated, then by definition it cannot be asserted to be a valid model of the business intent.
- A precisely defined and provably valid set of requirements as defined by a set of Fact and Decision Models must by definition be able to be transformed into computer code without human intervention – whether this actually occurs is immaterial, but it must be conceptually possible, otherwise the requirements by definition cannot be implemented and are invalid as presented.

The relationships between the various components of the approach are shown in Figure 2.
Using Fact and Decision Models as proposed above to capture business requirements will allow us to harvest all of the benefits asserted in this article, including dramatic improvements in business and project outcomes. Part 3 will outline the actual approach for what we will now call **Decision Centric Analysis**.

Figure 2.
Decisioning Links Strategy to Operational Systems



Part 3.
The Decision Centric Analysis Approach

Decisioning

Defn: *The discrete and systematic discovery, definition, assembly and execution of decisions.*

We have asserted that decisions are first order citizens of the requirements domain. To provide a conceptual basis for this, let us go back to the foundations of any requirements analysis.

A business derives value from changing the state of some object or 'thing' that the business values (usually literally, on its balance sheet). This focuses us on the very core of the business – what is traded, managed, leveraged, used, built, or sold in order to achieve the objectives of the business. Until something happens to the object, the purpose cannot be achieved and no value can be derived; therefore, in order to generate value, we must have a state change on the object. Changing state implies that there is activity against an object, and this observation gives rise to the traditional process (activity) and data (object) approaches. But if we look closely at any process causing state change, we will see that the change is always the result of a decision within the process rather than the process itself. Many processes can execute against the object, but value is only derived when a state change occurs. Whenever state change occurs, the process can be shown to be a container for decisions – the state change is a result of decisions made rather than an inherent characteristic of a process. This confusion between decision and process is a systemic failure in today's methodologies and hides the importance of decisioning. A process is the glue that links decisions to the events that initiate them; and in doing so it provides a mechanism for supplying data to the decisions, and for reacting to the decisions made. If these pre and post decision activities are wrapped together with the decision logic itself, then we have a complete end-to-end process that takes the business from one valid state to another, and which generates value in doing so. But it is clear that the entire process is built around the decisioning kernel.

Decisions

Defn: *A proprietary datum derived by interpreting relevant facts according to structured business knowledge*

Decisions are the critical mechanisms by which we choose one action from the set of all possible actions. The purpose of a decision is to select the action that is most beneficial to the decision maker in a given situation.

A decision is 'made' when we resolve the available facts into a single definitive outcome; a decision cannot be multi-valued, ambiguous or tentative. A decision results from applying discipline, knowledge and experience to the facts that describe the decision context. This application of discipline, knowledge and experience within decision-making is the characteristic that most closely defines the unique character of any business. Because of this fundamental truth, decision-making behaviour is the only truly proprietary artifact in any system specification – most other artifacts can be inferred from industry practice, and do not differentiate each business specifically.
Businesses do not make decisions merely because they have available data; nor do they act without making a decision. Decisions are clearly identifiable at the heart of any automated systems response, giving purpose to the data and direction to the process. If decision centric analysis is used to rigorously identify and describe decision-making behaviour prior to systems development, then it can also be used to drive the discovery of data and its relevance to the business – it is the need for the decision that is the primary driver of the need for the data. Decisioning is therefore a primary data analysis tool, and a precursor to formal data modeling. When data analysis is driven by decision modeling, it gives rise to concise and provably relevant data models. And because decisions also predicate process responses, decisioning also implicitly drives process definition.

The starting point

The business derives value or achieves its purpose by changing the state of primary business objects (or things), whether they be insurance policies, loan accounts, patients, trains, or any other object of value (perhaps even decision models!). These primary business objects are usually self-evident, and a cursory review of business strategy documentation will highlight and clarify any ambiguities. But if the primary objects cannot be defined quickly (minutes, not days) and with certainty, the business should not be building a system – it has much bigger issues to worry about!
A primary business object is by definition both tangible and discrete; therefore, it can be uniquely identified. Also by definition, it will be a source of value for the business and so the business will usually assign its own unique identifier to it – it's first data attribute and the beginning of our fact model. In fact, a useful way to find these objects is to look for proprietary and unique identifiers assigned by the business (even in manual systems). Because it exists and generates value, external parties may also be involved and cause us to add specific additional non-discretionary attributes for interface or compliance purposes (e.g. a registration number or unique address), and we might also add other identifying attributes to ensure that the primary object can be uniquely found using 'real world' data. So there is a small set of non-discretionary data that

can be found and modeled simply because the primary object exists; this set of data is generic and will be more or less common across all like businesses. We can think of this as 'plumbing' – it cannot be a source of business differentiation.

So what other data is there that will allow us to differentiate our business? The amount of data that we could collect on any given topic is boundless – how do we determine what additional data is relevant? Decisioning! How we make decisions is the key differentiator of any business – how we decide to accept customers, approve sales, price sales, agree terms and conditions, etc. The important additional data will now be found because it is needed for decision making. We are now ready to do decision analysis; that is, after the initial strategic scoping, and prior to either detailed data or process analysis.

Decision Discovery

Decision analysis is both simple and intuitive because this is the primary activity of the business – this is what the business knows best. A decision is a single atomic value – we cannot have half a decision anymore than we can have half an attribute. So we are looking for explicit single valued outcomes, each of which will be captured as a new data attribute. Let's start with our business object (say, 'insurance policy'), and with a state change that will cause a change in the value of that object (say, 'approve insurance policy'). If we have a documented set of governance policies that describe the business intent, we will interrogate them looking for noun phrases, and related assertions and conditions. Noun phrases can be interpreted directly into the Fact Model. The assertions and conditions will be reduced to a set of operations and declared as decision logic. We should also access the domain expert within the business – this is the person charged with responsibility for strategy directed decision-making, and who can readily answer questions like the following: "What decisions do you make in order to … approve this insurance policy?"

There is a pattern to most commercial decision making that helps structure the decision discovery process (see fig.3). The green squares are the most important in the cycle. We can start analysis with the first of the primary (green) decision components ('Authorization or Acceptance' in Fig.3) – these are the 'will I or won't I' decisions. For our insurance policy, 'will I or won't I' accept this risk? (for other problem domains simply replace the business object and state as required; e.g. for a hospital, 'will I or won't I admit this patient?' etc.). Determining this decision may identify many precursor decisions that need to be considered first. For instance, for our underwriting question we might need to ask:

- What is the inherent risk of the object?
- What is the customer risk?
- What is the geographic risk?

These decisions in turn may give rise to further decisions so that we develop a tree of decisions – this is the decision model for authorization. Now we can move on to the next in the primary class of decisions: at what price? (or cost if a cost centre) – see 'Pricing or Costing' (Fig.3). In this case, the question is 'what decisions do you make to determine the price... of this risk?' Again, this may result in a tree of decisions (for instance, pricing based on various optional covers, pricing for the different elements of risk, channel pricing, campaigns and packages, etc.). Following 'at what price?' we can repeat the process for the 'Terms and Conditions' and then the other pre and post decisions:

- Pre-Processing Check: Do I have sufficient information to start decision making?
- Context Based Validation: Is the supplied data valid?
- Enrichment: What further data can I derive to assist with the primary decision making?
- Product or Process Selection: Do I need to determine one decision path from other possible paths for the primary decision making?

And after the primary decision making...

- Workflow: Now that I have made my primary decisions, what do I need to do next?
- Post-Processing: Am I finished and complete without errors? Are my decisions within acceptable boundaries?

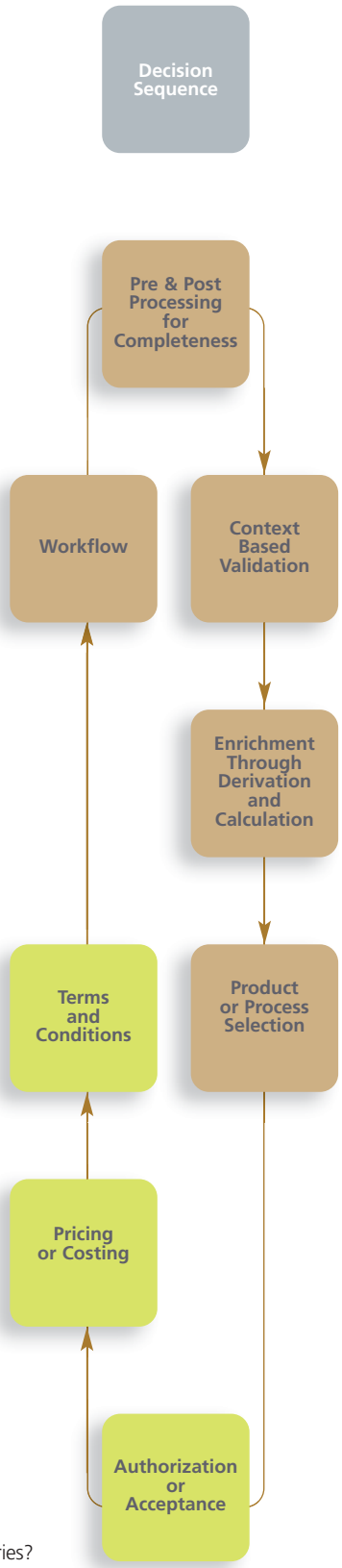


Figure 3.
The Decision Sequence

Normalization

Data normalization is a semi-rigorous method for modeling the relationships between atomic datum. It is 'semi-rigorous' because normalization is a rigorous process that is dependent on several very non-rigorous inputs, including:

- Scope: Determines what is relevant – we don't normalize what is not relevant;
- Context: Determines how we view (and therefore how we define) each datum;
- Definition of each datum: This is highly subjective yet drives the normalization process.

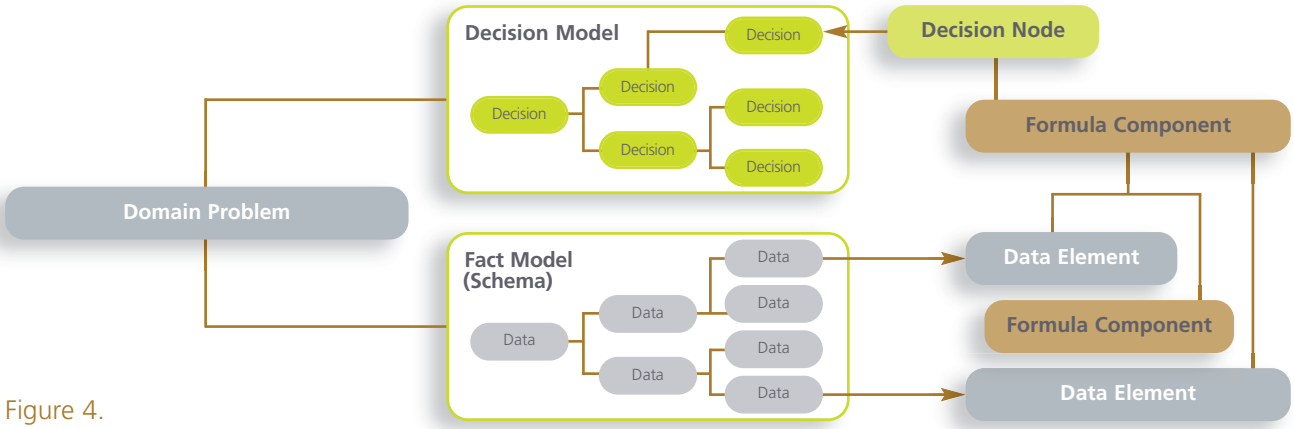


Figure 4.
Decision-centric Development

Data normalization is based on making a number of subjective assertions about the meaning and relevance of data, and then using the normal forms to organize those assertions into a single coherent model. Normalization with regard to decisions is similar. Each decision derives exactly one datum, and is 'atomic' in the same way that the datum is. Similarly, each decision has relationships to the other decisions in the model. The decisions are related by both sequence and context. In this regard, context plays a similar role to the 'primary key' in data normalization. Some of the inter-decision relationships within a model include:

- All decisions have a context that is derived from the normalized placement of its output datum.
- Each decision definition may precede and/or follow exactly one other decision definition.
- Each decision may belong to a group of decisions that share the same context.
- In a direct parallel of 4th normal form, unlike decisions that share context should be separately grouped.
- A group of decisions itself has sequence and may also be grouped with other decisions and groups according to shared context.

Decision Driven Data Design

We have suggested that decision and fact models can both evolve from strategy directed decision analysis. Following the initial discovery and elaboration of the primary business objects, we worked through the decision discovery process by analyzing the strategy defined policies and any identifiable business intent, which in turn identified new data attributes (the decision outputs). If we locate these decisions around the data constructs in the Fact Model, we can build an integrated decision/data model (see Fig.4). Following the discovery of the decisions, we can then elaborate them with formulas. Formulas provide additional detail regarding the consumption of data by decisions, thereby driving further demand for data. If the system cannot provide this data, then by definition the business cannot make the decision and the business objective of the model cannot be achieved. In this way, decisioning can be shown to drive the demand for data. The interaction between data and decisions helps the normalization and modeling processes for both. This combined model is self-validating because when complete all data should be produced and/or consumed by one or more decisions; and all decisions must have context within and consume data from the Fact Model. This helps to ensure the overall rigor of the analysis. Note that the fact/data models used in decision modeling are subsets of the domain data model – they need to contain only the data required by the decisioning that is currently in focus. They are, in effect, normalized models of the value creating transactions rather than a model of the business domain. As noted in Part 2, the XML schema standard is a useful standard for describing normalized transactional data. The set of fact models converge to define the primordial data model that will underpin all further project related data analysis. It can be easily overlaid onto the business object model to synthesize a more complete domain model that can then be further extended to include all of the 'plumbing' requirements (e.g. security, users, history, logs etc). The resultant model will only contain the data actually required by the business – there is no second guessing of the data requirements as often occurs in a traditional data analysis approach, with significant and positive implications for development cost.

Decision Driven Process Design

Decisioning also drives process. For a decision to execute, the decision model must be supplied with its fact model by a process. This data, and therefore the process, cannot be defined until the decision requirements are known. Then, for the decision model to have any effect, a process must enact some response to the decisions made. While it is possible to define and build processes in anticipation of the decisioning that will drive them, it is sensible to analyze the decisioning in order to determine the range of inputs and outcomes, and then to normalize the process responses to them. Again this has a positive effect on development cost and complexity.

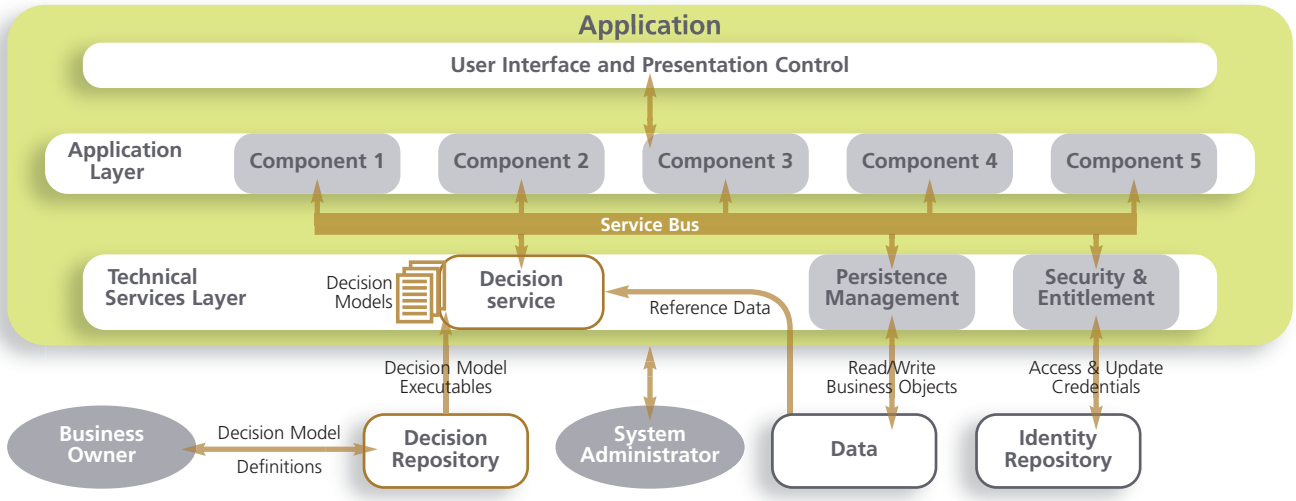
Bespoke discretionary processes do not occur in a vacuum. Such processes should only exist to support value creation for the business – as described by the decision analysis. Regulation and industry practice may require additional processes, but these are non-discretionary by definition and may not add value. Bespoke processes that are found to exist that do not have this fundamental decision driven requirement are not good subjects for primary analysis – certainly their mere existence does not make the process requirement a necessity, and they should always be considered to be candidates for removal or re-engineering. There may be many process options for supplying data and responding to decisions made. In particular we should look for opportunities for direct integration with external systems to create integrated industry solutions. This process re-engineering opportunity offers significant value, but may be missed if analysis of existing processes is

conducted as the primary analysis. For this reason we do not consider processes to be attractive first order development artifacts – they are in fact at the tail-end of the analysis chain, the glue that binds events, data, and decision making to a platform and its devices.

Handover and the SDLC

We can achieve a verified and tested decision model, and its integrated and co-dependent fact models, with relatively modest effort – often only a fraction of the cost of traditional approaches. All technology architecture and design options remain open. We have, in fact, defined and constructed a 'requirements model' of the core functionality of the system from the business perspective without constraining the technology options.
Further, this 'requirements model' is testable and can be proven to be complete, consistent and correct according to a candidate set of business test cases. Even better, we can retain the separate identity of this critical requirement over the long term, even across multiple system implementations – there need never be another legacy system. Using a biological analogy, this is the "DNA" that defines how the organization is operated irrespective of how it is implemented. Implementation is now of little interest to those business users who are responsible for decisioning – we have a clear and well defined handover point to begin the traditional development cycle. It is feasible, even desirable, to simply hand over the decision models and fact models to systems designers as their SDLC starting point.
It is worth re-emphasizing a critical point: Decision models live outside of any project or process – decision analysis precedes the SDLC entirely. By definition the subsequent system design must be able to supply and receive data that complies with the decisioning schemas. And the system must provide appropriate process responses to the decisions made. While these processes remain undefined, this is secondary analysis and is tightly bounded by the decision design that precedes it, which is in turn secondary to the specification of business intent as described by business strategy. Therefore, process analysis has been de-risked, and at the same time, it offers rich opportunities for business process re-engineering. The traditional SDLC approaches can then be used to design, build and/or reuse various software components (the plumbing) as appropriate to support the decisioning requirements. It is the developer's task to provide an infrastructure within which the decisioning can occur, as shown in Figure 5.

Figure 5. **Decision Oriented Architecture**



Conclusion

The decision-centric development approach represents a significant advance on traditional development methodologies. It focuses on a 'missing link' between business strategy and operational systems – the decision model. The decision model is a new and important requirements artifact that is accessible and understood by both business and development practitioners, giving rise to a previously unobtainable level of shared understanding that can help bridge the gap between business strategies and the systems that support them. The decision model gives the business 'hands-on' control over the definition and implementation of its most critical IP – its decision making know-how.

About IDIOM

Established in 2001, **IDIOM** is a pioneer in the development of decision automation concepts and approaches, and in their practical application to the design and development of systems of all sizes.

IDIOM has applied its "decision oriented" tools and approaches to improve systems development performance and business agility in projects in Europe, Asia, North America and Australasia, across such diverse domains as insurance, health, government, telecoms and logistics.

IDIOM leverages this experience in developing and marketing the "**IDIOM** Decision Manager", the industry leading, purpose built decision automation tool.

IDIOM Decision Manager is a proven, pragmatic and cost effective tool for capturing, managing and deploying business decision making know-how.

IDIOM Decision Manager is used by business users to define and control intelligent business processes through generation of small footprint, non-intrusive decision making software components.

IDIOM is a regular participant at the EuroBizRules and International Business Rules Forums. **IDIOM** was accredited to the Gartner 'Business Rules Engine' Magic Quadrant in 2005.

About the Author

Mark Norton has 30 years development experience, mostly on enterprise scale, mission critical systems in finance, insurance, government and health administration.

In 2001 Mark established Idiom Ltd. to develop and market tools and techniques in support of the decisioning concept. Through joint project participation with integration partners, these tools and techniques have been personally developed and tested on dozens of projects to deliver the pragmatic and conceptually sound decision centric development approach as described in this article.

Contact:
mark.norton@idiomsoftware.com
+64 21 434669

www.idiomsoftware.com