A Design Quality Learning Unit in Relational Data Modeling Based on Thriving Systems Properties

Leslie J. Waguespack
lwaguespack@bentley.edu
Computer Information Systems Department
Bentley University
Waltham, Massachusetts 02452, USA

Abstract

This paper presents a learning unit that addresses quality design in relational data models. The focus on modeling allows the learning to span analysis, design, and implementation enriching pedagogy across the systems development life cycle. Thriving Systems Theory presents fifteen choice properties that convey design quality in models integrating aspects of aesthetics, the more subjective phenomena of satisfaction; a quality perspective more expansive than that usually found in software engineering, the traditional "objective" notion of metrics. Recent IS curriculum guidelines relegate software development to elective status confining design pedagogy into smaller and smaller pockets of course syllabi. Where undergraduate IS students may once have practiced modeling in analysis, design, and implementation across several courses using a variety of languages and tools, they commonly now experience modeling in two or three courses in at most a couple of paradigms. And in most of these courses their modeling focuses on acceptable syntax rather than achieving design quality in information systems. Learning design quality may once have been an osmotic side effect of development practice, but now it must be a conscious goal in pedagogy if it is to be taught at all. This learning unit is intended as an adaptable framework to be tailored to the coursework and the overall objectives of specific IS programs.

Keywords: design quality, design, relational data modeling, IS curricula, IS pedagogy

1. INTRODUCTION

Over the past decade computing curricula have been repartitioned with the permeation of computing across disciplines and society. (Shackelford, Cross, Davies, Impagliazzo, Kamali, LeBlanc, Lunt, McGettrick, Sloan & Topi, 2005) There are now 5 major guidelines that subdivide curriculum in the computing discipline. (Soldan, Hughes, Impagliazzo, McGettrick, Nelson, Srimani & Theys, 2004, Cassel, Clements, Davies, Guzdial, McCauley, McGettrick, Sloan, Snyder, Tymann & Weide, 2008, Diaz-Herrara & Hilburn, 2004, Lunt, Ekstrom, Gorka, Hislop, Kamali, Lawson, LeBlanc, Miller & Reichgelt, 2008, Topi, Valacich, Wright, Kaiser, Nunamaker, Sipior & de Vreede, 2010) The co-location of IS curricula in schools of business further exacerbates the pressure on pedagogy as accreditation bodies further constrain the scope of coursework by compressing systems development into smaller and smaller pockets of course syllabi. (AACSB, 2010, EQUIS, 2010) Where undergraduate IS students once may have practiced modeling in analysis, design, and implementation across six or more courses in a program using a variety of languages and tools, they commonly now experience modeling in four or fewer courses in at most a couple of paradigms. (Waguespack, 2011a) And in most of these courses their modeling decisions focus on acceptable syntax rather than principles representing and communicating concepts of quality in information systems. Where learning design quality may once have been an osmotic side effect of development practice it must now be a conscious goal in pedagogy if it is to be taught at all.

At the same time industry and academia persist in their lament over the paucity of focus on quality in system design first sounded more than four decades ago (Dijkstra, 1968) and echoing consistently since as in (Denning, 2004, Brooks, 1995, 2010, Beck, Beedle, van Bennekum, Cockburn, Cunningham, Fowler, Grenning, Highsmith, Hunt, Jeffries, Kern, marick, Martin, Mellor, Schwaber, Sutherland, & Thomas, 2010)

This paper presents a learning unit that teaches quality design in relational data models. The focus on modeling allows the learning to span analysis, design, and implementation enriching pedagogy across the systems development life cycle. Thriving Systems Theory presents fifteen choice properties that convey design quality in models integrating aspects of aesthetics, the more subjective phenomena of satisfaction; a quality perspective more expansive than that usually found in software engineering, the traditional “objective” notion of metrics. This learning unit is adaptable to the coursework and objectives of specific IS programs. The paper presents: a brief overview of design quality, properties to assess design choices, the relational ontology; and a discussion of how each of the design choice properties express quality through the use of relational data modeling constructs. Finally, there is a description of how the learning unit has been integrated in data management syllabi with a comment on its efficacy. A parallel treatment of design quality pedagogy applied to the object-oriented paradigm may be found in (Waguespack 2011b).

2. WHAT IS DESIGN QUALITY?

Quality is an elusive concept, shifting and morphing on a supposed boundary between science and art: objective, engineering characteristics versus subjective, aesthetic observer or stakeholder experience. International standards of quality reflect the challenge of defining quality by offering a variety of perspectives (as gathered here by Hoyle, 2009):

- A degree of excellence (Oxford English Dictionary)
- Freedom from deficiencies or defects (Juran, 2009)
- Conformity to requirements (Crosby, 1979)
- Fitness for use (Juran, 2009)
- Fitness for purpose (Sales and Supply of Goods Act, 1994)
- The degree to which the inherent characteristics fulfill requirements (ISO 9000:2005)
- Sustained satisfaction (Deming, 1993)

(Waguespack, 2010c) asserts that the quality of systems revolves around two primary concepts: efficiency and effectiveness defined as follows (New Oxford American Dictionary):

Efficiency [noun]- the ratio of the useful work performed [...] in a process to the total energy [effort] expended

Effectiveness [noun]- successful in producing a desired or intended result

These two concepts appear primarily quantitative and therefore objective. In and of themselves they may well be. Portraying efficiency using a convenient interpretation of “work” and “effort” is genuinely objective. “How many” or “how much” or “how often” often depicts efficiency. But, when we ask “Is it enough?” apparent objectivity fades away.

Likewise, the supposed objectivity of “effectiveness relies upon the tenuous phrase, “desired or intended result” defined as

Intention [noun]- have (a course of action) as one’s purpose or objective; plan

Effectiveness (like efficiency) is a correspondence between a system and its stakeholders’ intentions. Assessing effectiveness depends on comparing “what is” to “what is intended.” While the former may be expressed quantitatively the latter presents challenges: clarity of conception, mode of representation, scope of contextual orientation, and fidelity of communication to name but a few. Indeed the notion of effectiveness is complicated when we contemplate identifying and quantifying the stakeholder(s) intentions objectively.

The indefiniteness or imprecision that characterizes stakeholder intention(s) is generally not a concern if an observer is asked to assess the beauty of something – an assessment generally conceded to be subjective.

A detailed or even explicit intention is not expected in assessing beauty – beauty is most often perceived as an experience of observation rather than a system analysis.

Most people commonly accept beauty as subjective and exempt from specific justification or explanation – “Beauty is in the eye of the beholder.” and “You’ll know it [beauty] when you see it.” This absence of or difficulty in forming a quantitative justification of beauty is often the basis for categorizing artifacts or processes as products of art rather than of engineering. And therein lies the presumption that the aspects of design quality that we label objective and those we label subjective are somehow dichotomous. They in fact teeter between objectivity and subjectivity depending on the degree of granularity that observers choose to employ in inspecting not only the artifact but also their own disposition toward satisfaction relative to it.

3. AN ARCHITECTURAL INTERPRETATION OF QUALITY DESIGN

We will never be able to absolutely define design quality because of the relativistic nature of satisfaction in the observer experience. But, our students must still face design choices. So, as IS educators we must provide a framework for them to develop and refine their individual perceptions and understanding of systems quality. The taxonomy of design choice evaluation proposed in Waguespack (2008, 2010c), the 15 *choice properties*, is just such a framework. (See Appendix A.) Choice properties derive from Christopher Alexander’s writings on design quality in physical architecture. (Alexander, 2002)

Choice properties address the process of building, the resulting structure, and the behavior of systems as cultural artifacts. Every design decision, choice, contributes to the aggregate observer experience: either positively or negatively. Each choice exhibits the 15 properties with varying strengths or influence that impact the resulting observer satisfaction. The confluence of property strength results from the coincidence of the designer’s choice with the collective intention of the stakeholders. The combination of all choices with their respective property strengths results in the overall, perceived design quality. Many of the properties are design characteristics long recognized in software engineering (i.e. modularization, encapsulation, cohesion, etc.). But several reach beyond engineering to explain aesthetics, the art (i.e. correctness, transparency, user friendliness, elegance, etc.). An example of the effectiveness of choice properties in explaining the design quality of production systems is reported in (Waguespack, Schiano & Yates, 2010b).

4. THE ONTOLOGY OF THE RELATIONAL PARADIGM

Illustrating design decisions in the relational paradigm can be a challenge. The idiosyncrasies of data modeling syntax often obscure the intention and/or the result of a design decision. For that reason the learning unit presented here uses a paradigm description independent of programming language, the relational ontology, found in (Waguespack, 2010a) and excerpted in Appendix B. The graphical outline of the ontology is Figure 1 below.

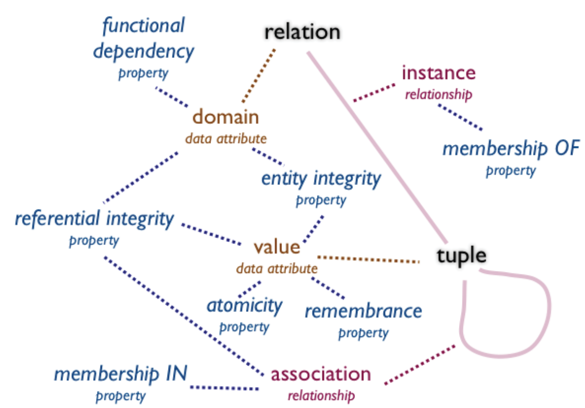


Figure 1 – Relational Ontology

The ontology captures the elements of the relational paradigm eschewing the obfuscation that usually occurs with programming language syntax examples. At the same time an experienced IS teacher can readily translate the ontological elements into a relevant programming or modeling dialect.

5. CRAFTING RELATIONAL MODELING CHOICES THAT STRENGTHEN PROPERTIES OF DESIGN QUALITY

This section, the heart of the learning unit, enumerates the 15 choice properties as defined in Waguespack (2010c) illustrating how modeling choices in the relational ontology can express design quality. In this space-limited discussion one choice property often references another reflecting the confluent nature of the design quality properties as Alexander defines them in physical architecture. (Alexander 2002)

Stepwise Refinement (as the name implies) is an approach to elaboration that presumes a problem should be addressed in stages. The stages may represent degrees of detail or an expanding problem scope. (Birrell and Ould 1988) In either case quality evidence of *stepwise refinement* is demonstrated by the cogent and complete representation of a design element at whatever level of detail or scope is set at each stage. To achieve this representation the modeling paradigm must support abstraction that allows generalization of the scope of interest and then the elaboration of that scope from one stage to the next.

In the *relational* paradigm the choice of *attributes* along with their interdependencies forms entities depicting facts. Each *relation* depicts a cohesive, encapsulated and distinct segment of knowledge. Each instance of that knowledge depends on its distinguishable *identity*: tuple by tuple. The scope of knowledge included in any particular model is constructed by the aggregation of these distinct segments interwoven through their explicit relationships. A whole model is built up stepwise as the "subset of the universe" chosen for the model (its *intension*) is systematically surveyed, cataloged and defined in the collection of *relations*. Each *relation*'s integrity is achieved through its independent *correctness* separate and distinct except for those relations with which is maintains foreign key relationships. But the *correctness* of the whole proceeds from the stepwise assembly of the entire set of *relations* that together describe the reach of a model's responsibilities.

Cohesion is a quality property reflecting a consistent responsibility distribution in a field of system components. (Zuse, 1997) Each *relation* serves a separate, *cohesive* role in the responsibility of representing domain knowledge. *Relations* reflect *identity* as they distinctly capture and represent concepts in the form of facts collected to represent cogent, clearly defined information. The *tuples* within *relations* similarly represent cogent, unambiguously defined instances of reality patterned after the *attribute* structure of their containing *relation* while by virtue of their *entity integrity* they remain distinct from any other *tuple* therein. The population of *tuples* in a *relation* reflects the ebb and flow of experience that the *relation* captures in the dynamics of the represented reality (the *extension*). The *attribute* structure of the *relation* as a template for each of its *tuples* ensures that the experience remains comparable and thus understandable regardless of the number of instances that experience produces. *Functional dependency* and its role in *normalization* assure that each relation represents an unambiguous and atomic division of knowledge in the modeling space. The result is a collection of distinct knowledge experiences bound together by a structure that both explains the significance of each instance and enables the analysis of that experience in terms of the whole reality that the *relation* captures.

Encapsulation is a design quality isolating and insulating instances of domain knowledge. In the *relational* paradigm the individual *relation* assumes the responsibility for capturing and defining the "reality," the "facts," the modeler chooses to instill in a model. The modeler's *intension* is represented in the structure of facts that each of its instances must be able to remember. Each instance of the *relation* remembers by way of the *data attribute value* set in each *tuple*. An important part of the reality captured in each *tuple* is its individuality and the uniqueness of the information that it remembers in its *data attribute values*, its entity integrity. The truthfulness of individual *tuples* can thus be independently established as an *encapsulated* division of "reality." (Scott, 2006) This individuality is determined solely by the values *encapsulated* therein dependent on no other information or relationships as characterized by *Second Normal Form*.

Extensibility is the property of design quality most important in pursuing systems with sustainability essential to cost of ownership economy. Extensibility juxtaposes the potential for new functionality with the effort required to achieve it. (van Vliet, 2008). Although each *relation* (down to the individual *tuple*) represents an independent depiction of reality in a relational model, more complex information is realized and extended through the relationships that associate *relations*. *Associations* permit the depiction of more elaborate descriptions of a model's responsibilities. *Associations* depict correspondence, interdependence or even ownership of concepts between and among *relations*. These *associations* are employed through the *relational operators* that combine or collect facts resident in multiple *relations* and render them correlated, organized and/or extracted as a consistent but distinct representation of knowledge contained in the model.

Modularization along with *cohesion* expresses "divide and conquer" problem solving augmented by the flexibility of configuring and reconfiguring model elements. *Modularization* also supports *scale* permitting the composition of subsystems of varying scope that hold details in abeyance until they require focus. (Baldwin and Clark, 2000) Enlightened module partitioning exposes the solution structure envisioned by the modeler and publishes intentions for further extension by separation of concerns and isolation of *accidents of implementation*. (Brooks, 1987) By the nature of depicting model knowledge in a collection of individual *relations* that knowledge is subdivided and compartmentalized. The process of *normalization* assures that the *intension* depicted by individual *relations* and combinations of *relations* through their *associations* are not ambiguous, redundant or inconsistent. The compartmentalization of knowledge not only affords stakeholders a clearer view of relations individually, but also exposes the opportunities to safely recombine that knowledge through relational operations. This *cohesion* that distinguishes each relation's role in the *intension* of the model also segregates the concerns that accomplish the model's responsibilities and permits attention to be focused on relevant subsets within the overall model's complexity.

Correctness in software engineering is often narrowly defined as computing the desired function. (Pollack, 1982) Thriving Systems Theory frames this property upon two outcomes: 1) validation, the clarity and fidelity of the represented understanding of system characteristics, and 2) verification, the completeness and effectiveness of model feature testing both individually and in composition.

Validation depends on the fidelity of the unfolding process; that through the stages of *stepwise refinement* the "essence" of system characteristics are brought forward maintaining their integrity. (Brooks, 1987) *Modularization* aids in cataloging and focusing on individual essential characteristics. *Correctness* is the only choice property that directly supports itself! *Correctness* must be a priority at each stage as shortcomings grow more and more expensive to rehabilitate as models evolve.

Verification depends on the effective testability of each choice to certify it as "consistent with stakeholder understanding." *Modularization* enables the verification of individual choices or relations. Then relying on the *correctness* of individual relations verification can turn to the certification of relationships resulting from *composition of function*.

Entity integrity, *referential integrity* and *normalization* directly support a relational model's fidelity to the modeler's *intension*. *Entity integrity* assures that the uniqueness of each depiction of reality (*extension*) is enforced by the structure of the relation, *intension*, (the *attribute* set, their respective *data attribute domains* and the respective *functional dependencies*). The specification of that subset of *attributes* that will always contain a unique (combination of) value(s) defines the discriminating characteristics of that knowledge (the *primary key*) – the conformance to which is easily tested and thus protected. *Referential integrity* assures not only that *data attribute values* conform to the *intension* of their *relation's data attribute domain* but, further to the modeled *intension* of *associations* between *tuples* including the ownership relationship between *relations*. *Normalization* extends the assurance of fidelity (model to the modeler's *intension*) by assuring that the interrelationship among *data attribute values* not only supports *entity integrity* and *referential integrity*, but also inhibits the accidental loss of model knowledge (*anomalies*) through the action of *relational operators*.

Transparency is evident structure, revealing how things fit and work together. (Kaisler, 2005) The relational paradigm facilitates *transparency* in two obvious respects. Inspecting the relevant *data attribute values* is sufficient to assess every aspect of integrity whether *entity integrity* or *referential integrity*. These same continuously accessible values form the basis of all relationships among *data attribute values* or among *relations*. The consistency of each and every *data attribute value* can be certified. At any time before or after any and every relational database operation we can verify concurrence with the time independent definition of *intension* given by the *data attribute* set and their respective *data attribute domains* along with the designation of *candidate* and *foreign keys*. There are no implied or hidden definitions of *association* or dependence. Every aspect of *tuple* or *relation* fidelity is discerned through self-evident information. The result of any *relational operator* is determined solely by the *data attribute values* of the *relations* involved.

Composition of Function - As a fundamental tool for managing complexity humans regularly attempt to decompose problems, issues or tasks into parts that either in themselves are sufficiently simple to permit direct solution or can through recursion be subdivided successively until they become sufficiently simple. This is a defining aspect of *modularization*.

Composition of function as a property of design quality is realized in model features that facilitate the extension or retargeting of the model in the future. It is the capacity to combine simple features to build more complicated ones (Meyer, 1988).

Each *relation* in a relational model represents a fundamental aspect of *intension* in the modeler's depiction of reality. *Association* and the use of *relational operators* effect that fundamental *intension* deriving an answer to any query we may invent based on that fundamental knowledge. The result of every *relational operation* is itself a *relation*. The modeler's ingenuity and discipline in forming queries carefully that yield results, *relations*, that are themselves consistent with the integrity constraints of the model creates the potential of an endless cascade of query result as input to another query and so on. This is the direct result of the mathematical formalism upon which the relation model is based – the predominating strength of the relational paradigm. The form in which these queries may be posed to a relational system is constrained only by the choice of mathematical representations (e.g. tuple calculus or domain calculus) or transformations (e.g. relational algebra or relational calculus) to the underlying relational definition.

Identity is at the root of recognition and is another property of design quality not usually defined in software engineering. In the physical world *identity* is literal based upon direct sensorimotor experience: by sight or touch and in some cases by sound or smell – a human experience of the "real" world. In the relational paradigm this human experience is applied directly by collecting those *attributes* that completely describe how any particular instance is unique – the combination of *attributes* that comprise the *primary key*. (Khoshafian and Copeland, 1986) The *primary key* serves to anchor the knowledge that surrounds it – those additional *attributes* that further describe the *tuple* which it uniquely *determines* –those *attribute values* that are *functionally dependent* upon the *primary key*. No *tuple* is permitted to exist in the relational universe (*extension*) unless it has a *primary key – entity integrity*. Ownership as it is manifest through *foreign key associations* is also anchored on the *primary key* of the owner *tuple*.

Scale's affect on design quality is reflected in common idioms: "You can't see the forest for the trees!" and "Let's get a view from 10,000 feet." They reflect the importance of context in recognition and decision-making. *Scale* captures the modeling imperative that all choices must be kept in perspective because it is not sufficient to consider a choice only in the microcosm of itself, as it must also participate in the connectedness of the whole. By achieving scale, a system designer provides differing granularities of comprehensibility to suit the requirements of a variety of observers (Waguespack, 2010).

In many cases the only familiarity that is needed in a relational model is the *intension* – the collection of *relation* definitions with their *attribute* sets defining their respective *attribute domains* and the *associations* among the *relations*. The knowledge structure and semantic relationships that may be mined through *relational operators* sufficiently defines any derivation of information representations that queries may be formulated to elicit. In terms of *scale* any relational model (*intension* or *extension*) may be expanded to incorporate additional knowledge. The modeler achieves this by grafting new knowledge onto existing *relation* structure through the addition and/or alignment of *data attribute domains* and *associations*.

User Friendliness is another property of design quality more often considered aesthetic. It is a combination of: ease of learning; high speed of user task performance; low user error rate; subjective user satisfaction; and, user retention over time (Shneiderman, 1992). Its impact may be easiest to consider in its absence. A modeling choice that is "unfriendly" to stakeholders is confusing, hard to comprehend, unwieldy, and perhaps worst of all, of indeterminate *correctness*. That which defies understanding cannot be determined to be correct. Satisfaction is cumulative. The sensitivity to the stakeholders' conceptions of the essence of the system to be modeled is key to the stakeholders' sense of comfort, familiarity, and expectation.

There is *elegance* in the succinctness and simplicity that arises from properly isolating domain knowledge in the respective *relations*. The use of user/client/customer familiar naming of *relations* and *attributes* and the choice of the commonly used, domain based *attribute values* lends a comfort level to the representation of problem domain experience. The relational model also enables the derivation of contained knowledge at levels of granularity much higher than the individual *tuple* or *relation*. This is because *relational operations* on *relations* produce *relations* as their result. Information derived from a relational database can be presented as if it were simply retrieved from a single physical *relation*. This illusion is easily achieved in relational programming languages that support the definition and storage of queries that may then be referenced themselves as *relations* without the users' notice (*i.e. in ANSI SQL the "create view" syntax*). The facility of such extensions to apply relational operations so discretely creates virtually unlimited opportunities and permits what might otherwise be a complex and daunting algorithm of derivation to be completely ignored by the users.

Patterns describe versatile templates to solve particular problems in many different situations (Gamma et al., 1995). *Patterns* is the property of design quality that channels change (unfolding). A pattern foreshadows where and how change will need to be accounted for. Patterns of the form popularized in (Coplein, 1995) document commonly encountered design questions offering carefully considered advice and cautions.

The most predominant *pattern* found in relational models is the regularity of structure that is embodied in the *tuples* that populates *relations*. This regularity assures that the same "questions" may be posed to each and every instance in a *relation* to elicit a consistently meaningful result. The *tuples* may be readily compared one to another and ordered that their factual content may be exhibited in a useful exposure of multiplicity. At the next level of structure we find the *foreign key* relationship where an *association* between *relations* is constructed by choosing *attributes* in the two *relations* that proceed from the same *attribute domain*. The *pattern* is further emphasized by the property of *referential integrity*. This *pattern* of connecting facts between and among *relations* permits the stepwise assemblage of higher and higher levels of derived information. The *association* enables the traversal of a network of concepts and facts that are both defined by and operationally enabled by the *foreign key* construct. The use of these *patterns* by the relational model designer provides the opportunity to lay out domain knowledge in a predictable and usable mapping.

Programmability in software engineering is often considered a feature rather than a property of design quality – the capability within hardware and software to change; to accept a new set of instructions that alter its behavior (Birrell and Ould, 1988). It is closely allied with *extensibility* and addresses the need for models to welcome the future. What largely separates information systems from other human-made mechanisms is the degree of adaptability that they offer to deal gracefully with change. Unlike most appliances that support a very narrow range of use (albeit with great *reliability*), contemporary information systems are expected to provide not only amplification of effort as in computation, but also amplification of opportunity in terms of different approaches to business or organizational questions. Contemporary information systems are expected to demonstrate that they can reliably accommodate change. As with *extensibility,* successful accommodation of change relies on an understanding of the fundamental options governing the structure and behavior within a particular domain.

What sets *programmability* apart from *extensibility* is a facility that permits altering the systems behavior without having to reconstruct choices – this versatility is not accidental but architectural.

Returning again to the use of *relational operations* to compose higher and higher levels of information we see individual *relations* as building blocks that may be arranged (assembled through *relational operations*) to yield any reasonable arrangement or derivation of information that the underlying *relations* may possess. This is possible because of the individual *identity* that each *relation* fosters in its *tuples* and because of the predictable *reliability* that proceeds from the consistency and safety of *relational operations* that is guaranteed in a set of *normalized relations*. The extent of information mining that may be attempted is limited almost solely by the programmers' imagination.

Reliability is a property of design quality more often associated with implementation than design. It is the assurance that a product will perform its intended function for the required duration within a given environment (Pham, 2000).

There is an overarching simplicity that results from the fact that all of the properties of integrity are based upon *data attribute values* that may be readily inspected before or after any *relational operation*. *Intension* is expressed in modeled expressions of integrity constraints that are domain specific. The synchronization between the *intension* and *extension* of the model is easily tested because of this simple *transparency*. *Reliability* is assured if valid *relational operations* are applied consistent with model integrity constraints and thus will always yield consistent ("truthful") information.

Reliability in design reflects an austerity that confines design elements to the essentials of the stakeholder's intentions. When design or implementation decisions involve additional constructs due to technology or compatibility, these *accidents of implementation* must be clearly delineated so as not to imply that they are essence rather than accident. This clear distinction will protect future system evolution from mistaking accidental "baggage" as stakeholder intentions.

Elegance is perhaps the epitome of subjective quality assessment that clearly sets choice properties of design quality apart from traditional software engineering metrics. "Pleasing grace and style in appearance or manner," that's how the dictionary expresses the meaning of "elegance." (Oxford English Dictionary)

> "A designer knows he has achieved perfection not when there is nothing left to add, but when there is nothing left to take away." (Raymond, 1996)

Models composed of choices that are consistent, clear, concise, coherent, cogent, and transparently correct exude *elegance* and nurture cooperation, constructive criticism and stakeholder community confidence. These are models that confess to their own shortcomings because their clarity obscures nothing, even omissions. These are models that satisfy stakeholders. They appear "intuitively obvious."

Elegance is achieved largely through the relational model when *relations* are modeled with a minimum of extraneous or redundant information. Indeed eliminating redundancy is common mantra of relational modeling. The laying out of basic facts divided into distinct *encapsulated* containers of knowledge and the subsequent composition of higher levels of derived information effects a sense of economy of form and abundant opportunity for exploring and extracting the knowledge that a database so fashioned accommodates.

Elegance largely proceeds from the efficient and effective representation of *essential* system characteristics along with those features emerging out of design decisions, *accidents of implementation*, that are laid out with equal clarity for separate consideration. This is the *field effect* of the beneficial, integrated, mutual support of strong choices described in Thriving Systems Theory. (Waguespack, 2010c)

6. INTEGRATING THE DESIGN QUALITY LEARNING UNIT IN A RELATIONAL MODELING SYLLABUS

The design quality discussion provides a quality vocabulary for one-on-one consultations between teacher and student as each develops their relational models. In this one-on-one context each student's specific design decisions may be discussed and evaluated in relationship to the design quality properties, an opportunity for individualized, reinforced learning and/or suggested improvements.

The deeper subtleties of design quality present a challenge for some students particularly in a compressed format. The "light doesn't go on" right away for all students. However, the integration of the ontology and design quality property based vocabulary establishes a touchstone that returning students report helps them "to name" the "quality elements" they rediscover in succeeding coursework and professional practice.

In your own curricular situation the distribution of learning unit elements may span more than one course (some addressed in database programming, requirements engineering, or database design, etc.), be rearranged to suit your modeling tools, or be adjusted to your course sequencing with context-appropriate examples. Regardless, the learning unit components are flexible and robust enough to suit various specific program needs.

7. ACKNOWLEDGEMENTS

Thanks to helpful referees. Special thanks are due my colleagues David Yates and Bill Schiano at Bentley University for their insightful discussions and comments on these ideas. And thanks to the students who have labored through the development of this learning unit.

8. REFERENCES

AACSB (2010). Eligibility Procedures and Accreditation Standard for Business Accreditation. Retrieved July 16, 2010 from http://www.aacsb.edu/accreditation/AAACSB-STANDARDS-2010.pdf

Alexander C, (2002). The Nature of Order An Essay on the Art of Building and the Nature of the Universe: Book I - The Phenomenon of Life, Berkeley, California: The Center for Environmental Structure, p. 119

Baldwin, C. Y., and Clark, K. B. (2000). Design Rules, Volume 1: The Power of Modularity. The MIT Press, Cambridge, MA.

Beck K., Beedle M., van Bennekum A., Cockburn A., Cunningham W., Fowler M., Grenning J., Highsmith J., Hunt A., Jeffries R., Kern J., Marick B., Martin R.C., Mellor S., Schwaber K., Sutherland J., & Thomas D. (2010). *Manifesto for Agile Software Development*. Retrieved July 12, 2010 from agilemanifesto.org

Birrell, N. D., and Ould, M. A. (1988). A Practical Handbook for Software Development. Cambridge University Press, Cambridge, UK.

Brooks F. P. (1987), "No Silver Bullet: Essence and Accidents of Software Engineering," Computer, Vol. 20, No. 4, pp 10-19.

Brooks, F. P. (1995). The Mythical Man-Month: Essays on Software Engineering (2ed). Addison-Wesley, Boston, MA.

Brooks, F. P. (2010). The Design of Design: Essays from as Computer Scientist. Addison-Wesley, Pearson Education, Inc., Boston, MA.

Cassel L., Clements A., Davies G., Guzdial M., McCauley R., McGettrick A., Sloan B., Snyder L, Tymann P., & Weide B.W., (2008). Computer Science Curriculum 2008 An Interim Revision of CS2001. Association of Computing Machinery (ACM), & IEEE Computing Society (IEEE-CS)

Coplien J and Schmidt D (Eds) (1995). Pattern Languages of Program Design, Addison-Wesley, Reading, MA, USA

Crosby,P. B., (1979) Quality is Free, McGraw-Hill, New York, NY, USA.

Dijkstra, E. (1968). “GOTO Statement Considered Harmful.” *Communications of the ACM*, 11(3), 147-148

Diaz-Herrara, J.L., & Hilburn, Thomas B. (eds.) (2004). Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, IEEE Computing Society (IEEE-CS), Association of Computing Machinery (ACM)

Deming, W. E. (1993), The New Economics for Industry, Government, Education (2ed), Cambridge Press: MIT, Cambridge, MA, USA

Denning, P. J. (2004). “The Great Principles of Computing,” *Ubiquity*, 4(48), 4–10

EQUIS (2010). *EQUIS Standards and Criteria*. Retrieved July 16, 2010 from http://www.efmd.org/attachments/tmpl_1_art_041027xvpa_att_080404qois.pdf

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA.

Hoyle, D. (2009). ISO 9000 Quality Systems Handbook. Butterworth-Heinemann (Elsevier); 6 ed. Burlington, MA, USA

ISO 9000 (2005), http://www.iso.org/iso/qmp

Juran, J. M., (1999). Quality Control Handbook (6ed), McGraw-Hill, New York, NY, USA

Kaisler, S. H. (2005). Software Paradigms. Wiley-Interscience, Hoboken, NJ.

Khoshafian, S. N., and Copeland, G. P. (1986). “Object identity,” *Proceedings of ACM Conference on Object Oriented Programming Systems Languages and Applications*, Portland, OR, November 1986, 406-416.

Lunt, B.M., Ekstrom, J.J., Gorka, S., Hislop, G., Kamali, R., Lawson, E., LeBlanc, R., Miller, J., & Reichgelt, H. (eds.) (2008). Information Technology 2008: Curriculum Guidelines for Undergraduate Degree Programs in Information Technology, Association of Computing Machinery (ACM), IEEE Computing Society (IEEE-CS)

Meyer, B. (1988). Object-oriented Software Construction. Prentice Hall, New York, NY.

Pham, H. (2000). Software Reliability. Springer, Berlin, Germany.

Pollack, S. (Ed.). (1982). Studies in Computer Science. Mathematical Association of America, Washington, DC.

Raymond, E. S. (1996). The New Hacker's Dictionary*,* 3rd ed. The MIT Press, Cambridge, MA.

Sales and Supply of Goods Act 1994, Ch 35, Legislation of Her Majesty's Government, The National Archives, UK, http://www.legislation.gov.uk/ukpga/1994/35/introduction

Scott, M. L. (2006). Programming Language Pragmatics*,* 2nd ed. Morgan Kaufmann, Maryland Heights, MO.

Shackelford, R., Cross, J.H., Davies, G., Impagliazzo, J., Kamali, R., LeBlanc, R., Lunt, B., McGettrick, A., Sloan, R., & Topi, H., (2005). Computing Curricula 2005: The Overview Report, Association for Computing Machiner (ACM), The Association of Information Systems (AIS), The Computer Society (IEEE-CS)

Shneiderman, B. (1992). Designing the User Interface: Strategies for Effective Human-Computer Interaction*,* 2nd ed. Addison-Wesley, Reading, MA.

Soldan, D., Hughes, J.L.A., Impagliazzo, J., McGettrick, A., Nelson, V.P., Srimani, K., & Theys, M.D. (eds.) (2004). Computer Engineering 2004: Curriculum Guidelines for Undergraduate Degree programs in Computer Engineering, IEEE Computer Society (IEEE-CS), Association for Computing Machinery (ACM)

Topi, H., Valacich, J.S., Wright, R.T., Kaiser, K.M., Nunamaker, J.F. Jr., Sipior, J.C., & de Vreede, G.J. (eds.) (2010). IS2010: Curriculum Guidelines for Undergraduate Degree Programs in Information Systems, Association for Computing Machinery (ACM), Association for Information Systems (AIS)

Van Vliet, H. (2008). Software Engineering: Principles and Practice, 3rd ed. Wiley, Hoboken, NJ.

Waguespack, L. J. (2008). "Hammers, Nails, Windows, Doors and Teaching Great Design," *Information Systems Education Journal*, 6 (45). http://isedj.org/6/45/. ISSN: 1545-679X

Waguespack (2010a). The Relational Model Distilled to Support Data Modeling in IS 2002. *Information Systems Education Journal*, 8 (3). http://isedj.org/8/3/. ISSN: 1545-679X. (A preliminary version appears in The Proceedings of ISECON 2009: §3133. ISSN: 1542-7382.)

Waguespack, L. J., Schiano, W. T., Yates, D. J. (2010b). "Translating Architectural Design Quality from the Physical Domain to Information Systems," *Design Principles and Practices: An International Journal*, 4, 179-194

Waguespack, L. J. (2010c). Thriving Systems Theory and Metaphor-Driven Modeling, Springer, London, U.K.

Waguespack, L. (2011a). "Design, The "Straw" Missing From the "Bricks" of IS Curricula," *Information Systems Education Journal,* 9(2) pp 101-108. http://isedj.org/2011-9/ ISSN: 1545-679X

Waguespack, L. (2011b). "A Design Quality Learning Unit in OO Modeling Bridging the Engineer and the Artist," Proceedings of the Information Systems Educators Conference, Wilmington, N.C., USA, v28, n1625, 14p.

Zuse, H. (1997). A Framework of Software Measurement. Walter de Gruyter, Berlin, Germany.

Editor's Note:

This paper was selected for inclusion in the journal as a ISECON 2012 Distinguished Paper. The acceptance rate is typically 7% for this category of paper based on blind reviews from six or more peers including three or more former best papers authors who did not submit a paper in 2012.

Appendix A – Choice Properties (Waguespack 2010c)

	Choice Property	Modeling Action	Practical Action Definition
1	Stepwise Refinement	elaborate	develop or present (a theory, policy or system) in detail
2	Cohesion	factor	express as a product of factors
3	Encapsulation	encapsulate	enclose the essential features of something succinctly by a protective coating or membrane
4	Extensibility	extend	render something capable of expansion in scope, effect or meaning
5	Modularization	modularize	employing or involving a module or modules as the basis of design or construction
6	Correctness	align	put (things) into correct or appropriate relative positions
7	Transparency	expose	reveal the presence of (a quality or feeling)
8	Composition of Function	assemble	fit together the separate component parts of (a machine or other object)
9	Identity	identify	establish or indicate who or what (someone or something) is
10	Scale	focus	(of a person or their eyes) adapt to the prevailing level of light [abstraction] and become able to see clearly
11	User Friendliness	accommodate	fit in with the wishes or needs of
12	Patterns	pattern	give a regular or intelligible form to
13	Programmability	generalize	make or become more widely or generally applicable
14	Reliability	normalize	make something more normal, which typically means conforming to some regularity or rule
15	Elegance	coordinate	bring the different elements of (a complex activity or organization) into a relationship that is efficient or harmonious

Appendix B - Relational Green Card (Waguespack 2010a)

THE RELATIONAL "GREEN CARD" NOVEMBER 11, 2008

The Relational Paradigm

Without a Language or Syntax!

What is the relational world all about?

The Relational Ontology

This ontology is consistent with the practice in computer science and information science categorizing a domain of concepts (i.e. individuals, attributes, classes and relationships). This ontology of the relational paradigm of data modeling minimizes the vestiges of implementation languages and methodologies to expose the core nature of relational concepts.

1. Individuals

The most concrete concept in the relational paradigm is the ***tuple***.

1.1. Tuple

A *tuple* corresponds 1-1 with a single concept of reality that it represents. A *tuple* collects the facts that identify it as a single concept and the facts most closely identified with it.

2. Attributes

Attributes are those characteristics (facts) that describe a *tuple*. In the relational paradigm *attributes* define data characteristics - each of which has a static and dynamic form. A prescribed set of *attributes* defines what is called the ***structure*** of a *tuple*. From inception to extinction the *structure* of a *tuple* is immutable. The number of *attributes* in a *tuple* is called its ***degree***.

2.1. Data Attribute

Data attributes store information (data) in the *tuple* and implement the property of ***remembrance***. *Remembrance* is manifest in each *attribute* dynamically as "what *is* remembered," a particular ***data attribute value*** for each *tuple* derived from a ***data attribute domain*** that statically defines "what *can* be remembered," the possible values of the *attribute*.

3. Classes

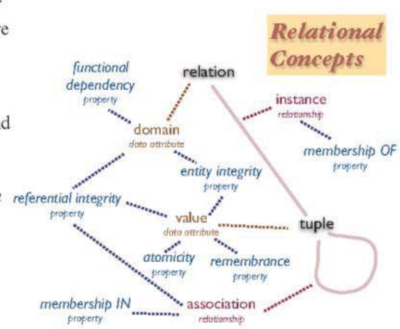
The relational paradigm groups individuals into a collection called a ***relation***. The *relation* corresponds directly with its mathematical antecedent where *attribute* values within each *tuple* reflect a correspondence with the coincidence of facts in the "real world," a correspondence (*attribute* relationship) that is shared by every *tuple* in that *relation*.

3.1. Relation

The ***relation*** concept combines both a definition of *structure* and the collection of *tuple(s)* based on that *structure*. A *relation* is defined as a fixed set of *data attributes*. Every *tuple* is an ***instance*** of a specific *relation* and shares the same static *structure* defined by that *relation* with every other *tuple* of that *relation*. The *relation* concept thereby fuses the existence of the *tuples* to that of their *relation*; *tuples* cannot exist independent of their defining *relation*. *Tuples* are said to be ***members of*** their *relation*. *Tuples* are added to or deleted from their *relation*. The order of attributes in a *relation* is insignificant except that the order is consistent for all *tuples*. *A relation* is also commonly called a ***table*** and each of its *instances*, a ***row***. The collection of every *data attribute value(s)* for a particular *data attribute* in a *table* is called a ***column***.

4. Relationships

Relationships in the *relational* paradigm are based on the property of *remembrance* and the juxtaposition of *data attribute values* in one or more tuples in the same or across *relations*.



4.1. Behavioral Relationships

The behavioral relationships are all based upon the *data attribute value(s)* and which values are permitted to coexist in and across *tuples* and *relations*.

4.1.1. Functional Dependency

In a *relation* a *data attribute* is ***functionally dependent*** when its *data attribute value* is always the same in any *tuple* for a given value in a second *data attribute*. In other words, the value of the first *data attribute* ***is determined by*** the value of the second (called the ***determinant***). *Functional dependency* expresses the informational integrity of *relations*.

4.1.1.1. Entity Integrity

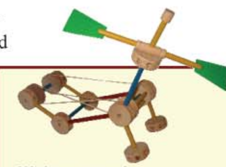
Entity integrity defines the two-fold quality of *tuple* uniqueness in a *relation*: a) every *tuple* in a *relation* is distinct in some *data attribute*

©2008, Les Waguespack, Ph.D. PAGE 1

value(s) from every other *tuple* in that *relation* or symmetrically, b) there is a designated subset of *data attributes* (*column(s)*) called the ***primary key*** such that the *data attribute value(s)* in that *relation* is distinct for all *tuples* and no values may be ***null*** (a value which is unknown and incomparable to any other value). There may be more than one subset of *data attributes* with the value characteristics of the *primary key* (each called a ***candidate key***) but only one is designated as the *primary key*.

4.1.2. Association

An ***association*** is a relationship between *tuples* in the same or different *relations*. *Tuples* are intrinsically separable by way of *entity integrity*. At the same time, humans are compelled to categorize their experience of things in the physical world by superimposing groupings that collect *tuples* into sets. *Tuples* become members in a group based upon *data attribute value(s)*. This property is called ***membership IN***. This property also permits humans to identify a *tuple* that is not in a set (i.e. discrimination). (*Membership IN an association is distinct from membership OF a relation which is intrinsic by way of instance relationship.*)



Without syntax?

Every *language* that is invented to express concepts carries with it the understanding and the biases of the inventor. Depending on his/her purpose(s) those biases simplify certain tasks performed with the language but may obscure the underlying concepts.

Programming language design must deal with the feasibility of automated translation and interoperability with other programming languages and operating systems. Compromises and assumptions are chosen to make the resulting language efficient, effective and marketable.

The goal of this description of the entity-relationship paradigm is to succinctly make the concepts understandable - an ambitious task to say the least!

- Professor Waguespack

4.1.2.1. Relational Operations

Membership IN is realized through **relational operations** keying on relation structure and values. Each relational operation produces a real or virtual relation as its result. The **selection** operation retrieves *tuple*(s) based upon a **selection predicate** testing data attribute value(s) to determine whether each *tuple* is or is not in the set. Selection predicates are based on any boolean comparison including constant values or values referenced in *data attribute value(s)*. The **projection** operation copies all the *data attribute value(s)* for a particular *column(s)*.

Association between *relations* (or a *relation* and itself) is based upon relating (matching) *data attribute values* in *tuples* of one *relation* with those of another. The ***join*** operation pairs every combination of *tuples* from one *relation* with those of another *relation* and copies the *data attribute values* from the pairs where the pairing satisfies a *selection predicate*. This *relational operation* is called *join* because facts from two sources are joined in the result.

4.1.2.2. Join Compatibility

Join compatibility requires that the values involved in comparisons (i.e. *selection predicates*) whether constants or *data attribute values* derive from the same *data attribute domain*.

4.1.2.3. Referential Integrity

When *relations* are devised such that a *tuple* in one *relation* predisposes the existence of (***owns***) *tuple(s)* in another, the *data attribute(s)* of the second required to *join* the *relations* is called a ***foreign key***. Referential integrity asserts that any value found in the *data value attribute(s)* of a *foreign key* must appear in a *tuple* of the first *relation* as the value of a *candidate key* or itself be *null*.

4.1.3. Normalization

Relational model consistency depends on the semantic concurrence of the behavioral relationships and the objectives of the database modeler, the ***intension***, (rather than the accident of a *relation's* contents at any particular instant, its ***extension***). The integrity properties defined above enable the database modeler to devise a structure and behavior of *relations* that avoid semantic discord called ***anomalies***, the unintended loss or modification of information by relational operations. *Relations* designed to avoid certain kinds of *anomalies* are said to be ***normalized*** or in ***normal form***. ***Normalization*** is the arrangement of *data attributes* and their relationships among *relation* structures to prevent particular *anomalies*.

4.1.3.1. First Normal Form

First Normal Form asserts that every *data attribute value* is ***atomic***, indivisible in value or form and may not be operated upon except as a whole and single value.

4.1.3.2. Second Normal Form

Second Normal Form is first normal form and asserts that every *data attribute value* not in the primary key is ***fully functionally dependent*** upon the *primary key*. ("Fully" means applying to every *data attribute* of the *primary key*.)

4.1.3.3. Third Normal Form

Third Normal Form presupposes first and second normal forms and asserts that no *data attribute* outside the *primary key* is ***transitively dependent*** upon the *primary key*. ("Transitively" means an attribute(s) *functionally dependent* upon an attribute *functionally dependent* upon an attribute (. . .) *functionally dependent* on the *primary key*.)