Three Innovative Software Engineering Methodologies

Elvis C. Foster, PhD
Associate Professor of Computer Science
Department of Computer Science,
Keene State College, Keene, NH 03435

Thomas O'Dea, former student of Keene State College

Myles Dumas, current student of Keene State College

Presented at the GOCICT Conference, Sullivan University, 2015
[Abridged version available at http://ieeexplore.ieee.org/document/7545104/]

Abstract

Software engineering has come to the stage where speed of development, level of correctness, interoperability, user friendliness, usefulness, and reusability in different projects are very important factors in determining the success of a software engineering venture. Equally important is the use of methodologies for software design. In just over six decades, we have seen the progress of software design from an amorphous set of informal methodologies to structured techniques, formal methodologies, and object-oriented methodologies. In the area of object-oriented methodologies (OOM), the unified modeling language (UML) has made a significant contribution in defining a set of methodologies that can be applied to any software engineering effort.

This paper draws on the UML methodologies and proposes three methodologies that could add richness and additional flexibility to the software engineering experience. They are *system topology charts*, *entity/object specification grid*, and the *extended operation specification*. The system topology charts include an *information topology chart* (ITC) that presents the object types and/or information entities in the way they will be managed in the software system, and a *user interface topology chart* (UITC) that presents operations the way they will appear in the system. The entity/object specification grid (E/OSG) adopts the conventions of the UML class diagram, but expands it to include additional critical information that lead to better software construction. The extended operation specification (EOS) embraces the UML guidelines for the activity diagram, but is flexible enough to include other techniques such as pseudo-code, Warnier-Orr diagram, and collaboration diagram. The methodology also allows for the specification of other critical information not covered in these standard techniques.

Keywords: Software Design Methodologies; Software Documentation; Software Requirements Engineering; Database Specification; User Interface Design.

Elvis C. Foster is Associate Professor of Computer Science at Keene State College, New Hampshire. He holds a PhD in Computer Science from University of the West Indies, Mona, Jamaica. Among his varied interests is a focus on the design and construction of more effective management support systems (MSSs) for the business environment.

1. Introduction

Computer science and software engineering are relative young disciplines when compared to more traditional disciplines. If we date the software engineering discipline from the 1940s, then during the relatively short period, we have witnessed the maturation of the discipline into a mainstream field. There is not a single professional discipline that has not been impacted by software engineering. Professionals in all fields known rely on software systems (developed by software engineers) to be more productive, and to make their work more manageable, meaningful, and worthwhile. So successful has been the impact, it has now become ridiculous to even attempt building a credible academic program without input from computer science (CS) or information technology (IT), the parent fields for software engineering.

Contemporary software engineering has been significantly impacted by three subfields and one related field. The contributing fields are summarized below:

- **Structured Techniques:** Early software engineering methodologies were somewhat amorphous, and relied on intuition and the skill of the software engineering team. Then came structured techniques as an attempt to formalize the discipline, and make it more understandable and learnable. There is a rich heritage of literature on these techniques; two examples are [Martin & McClure 1985] and [Marca & McGowan 1987].
- **Database Systems:** The field of database systems is sometimes seen as a subfield of software engineering. However, it can also be construed as a parallel complimentary field. Whatever the perspective, software systems are often characterized by underlying database systems on which they rely for information support. This field is also blessed with a rich reservoir of literature; for example, see [Date 2004], [Hoffer, Venkataraman, & Topi 2013], and [Kroenke & Auer 2015].
- **Object-Oriented Methodologies:** Since the 1990s, object-oriented methodologies (OOM) have dominated the software engineering discipline. These methodologies draw the best features from structured techniques into a new set of techniques and technologies, leading to improvements in a wide range of areas including software efficiency, design and development efficiency, sophistication, quality, interoperability, understandability, and platform independence. Moreover, OOM as a subfield is flexible enough to peacefully incorporate database systems methodologies in some instances, and replace them in other instances. OOM enjoys widespread acceptance in the software engineering community, so that there is a rich literature reservoir including (but not confined to) works such as [Jacobson 1991], [Martin & Odell 1993], [Rumbaugh et al. 1999], and [Bruegge & Dutoit 2010].
- **Mathematical Modeling:** Being an applied discipline, software engineering draws from various other engineering disciplines, and significantly from the field of mathematics. We use mathematical models to help formulate, explain, and analyze algorithms. In situations requiring complex analysis (for instance forecasting, simulation, numerical analysis, linear programming, graph theory, game theory, etc.) we draw heavily from mathematics, but often expand the adopted mathematical models to suit different scenarios. An excellent example of this is Robert Sedgewick's work on graph algorithms [Sedgewick 2002].

As a discipline, software engineering embraces both standardization and creativity. Through standardization, we have been able to have a pervasive discipline that cuts across cultural and national boundaries, and produce software systems that are not only platform independent,

but immune to cultural and national idiosyncrasies. Despite this emphasis on standardization, software engineering encourages initiative and creativity. We are always seeking to improve on methodologies learned, and where necessary to develop new methodologies for conducting work in an efficient manner. It is this confluence of standardization, initiative, and creativity that makes software engineering such a progressive and exciting discipline.

In keeping with the themes of standardization, initiative, and creativity, this paper proposes three methodologies that offer richness and additional flexibility to the software engineering experience. They are *system topology charts*, *entity/object specification grid*, and the *extended operation specification*. The system topology charts present software system information in a manner that allows for easy planning and management of the system. The entity/object specification grid (E/OSG) adopts the conventions of the UML class diagram, but expands it to include additional critical information that lead to better software construction. The extended operation specification (EOS) embraces the UML guidelines for the activity diagram, but is flexible enough to include other techniques such as pseudo-code, Warnier-Orr diagram, and collaboration diagram. The methodology also allows for the specification of other critical information not covered in these standard techniques.

The paper proceeds with four additional sections. Additionally, throughout the paper, a generic *inventory management system* (IMS) will be used as a case study. Section 2 introduces the system topology charts. To illustrate their usage, an object flow diagram (OFD) will be presented, followed by system topology charts for the system. Section 3 introduces the E/OSG, and illustrates how this methodology can be used to prepare a detailed database specification for the inventory management system (the E/OSG also includes a list of all operations that will be defined on each object type or entity). Next, the EOS is discussed in section 4; the paper shows how information in the E/OSG can be used to define an EOS for each operation that will form part of the user interface of the system. Finally, section 5 provides a summary and some concluding remarks.

2. System Topology Charts

The proposed system topology charts include an *information topology chart* (ITC) that presents the object types and/or information entities in the way they will be managed in the software system, and a *user interface topology chart* (UITC) that presents operations the way they will appear in the system menu(s).

2.1 Information Topology Chart

The *information topology chart* (ITC) shows information levels of the system in a top-down manner — the system is at the highest level and data elements if included, are at the lowest level. In many cases, the information entities (or object types) appear at the lowest level. Data elements are often excluded as they tend to clutter the diagram; besides, there are other equally creative ways to include such details. The ITC presents information to be managed in the system in a logical modular way and therefore allows for easy analysis and identification of omissions or redundancies.

The ITC is particularly useful in providing a global view of the system, including all significant components (subsystems and information entities). At first sight, one may be tempted to compare the technique with other existing techniques such as the HIPO (hierarchy input-process-output) chart (as described in [Kendal & Kendal 2014]), the fern diagram, object flow diagram (OFD), or the object-relationship diagram (ORD) as described in various literature on OOM (for instance [Martin & Odell 1993]). A comparison of the ITC with these methodologies was first provided in [Foster 1999]. For ease of reference, this comparison is summarized here:

- The HIPO chart is a functional representation of processes in a system. On the other hand, the ITC is a conceptual representation of subsystems and constituent information entities (or object types) of the software system.
- The fern diagram is used in object-oriented design (OOD) to illustrate object categorization (including component and inheritance relationships). For very complex systems, fern diagrams have the tendency to be complex, cluttered and difficult to read. The ITC may be used in OOD or function oriented design (FOD), solely to illustrate how information will be categorized for the purpose of management. As such, the technique often incidentally illustrates component relationships, but no attempt is made at representing inheritance, or other types of relationships.
- The OFD is used to illustrate the flow of activities and/or data among objects. The ITC provides no such illustrations.
- The ORD illustrates the relationships that exist among object types; for large, complex systems, it has the tendency of being voluminous and difficult to follow. The ITC typically illustrates component relationships, but that is not its sole or main purpose.

To illustrate the use of the ITC, consider a basic inventory management system that allows a store manager to keep track of supplies, purchases, and sales. Figure 1 provides an OFD for the system, and figure 2 shows a partial ITC, featuring the information entities organized in three subsystems.

Figure 1: Object Flow Diagram for the IMS Project

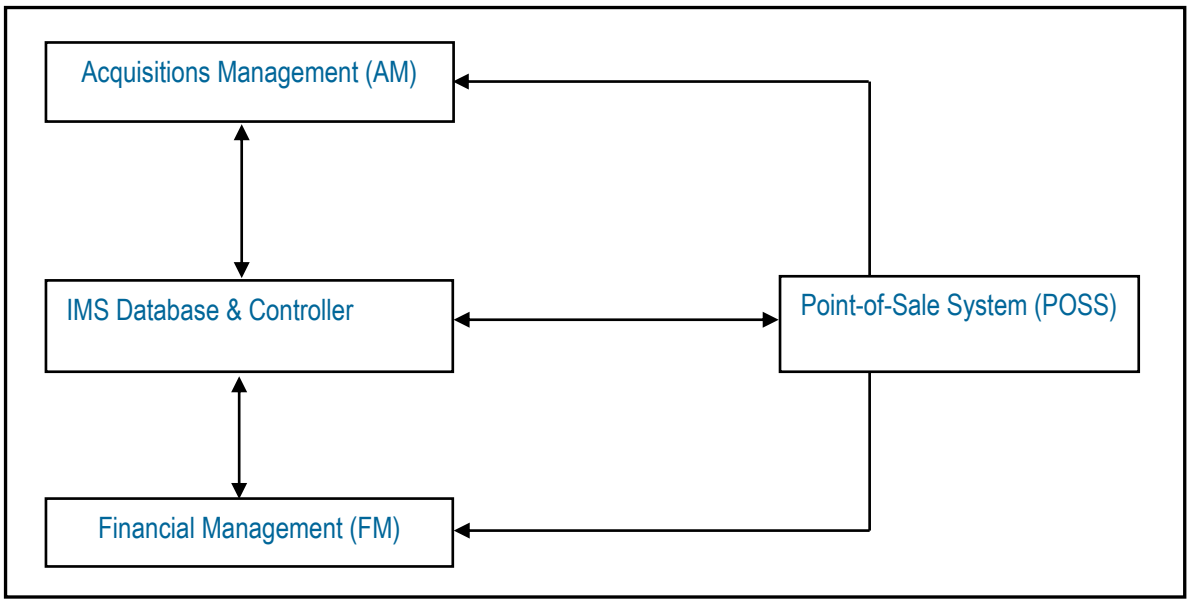
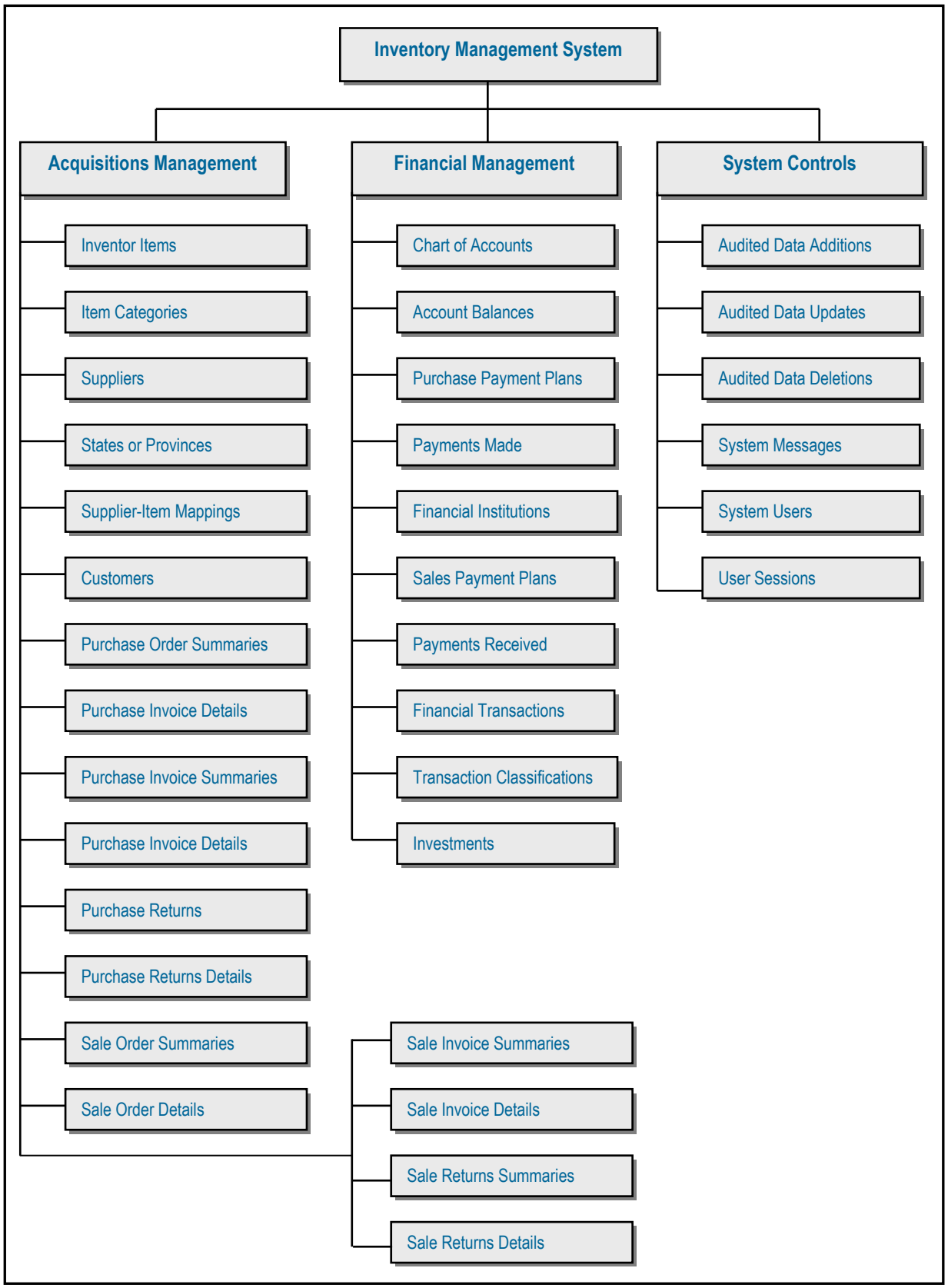


Figure 2: Information Topology Chart for the IMS Project



The ITC is useful during the investigation and analysis, as well as the design phases of the SDLC (software development life cycle). Additionally, once created, it serves as an effective system documentation tool, as well as a means of identifying system problems that may exist. Among the benefits provided by the methodology are the following:

- The ITC is a useful design and documentation aid.
- The technique is easy to learn, involving minimal use of symbols.
- The technique is useful in conceptualizing (the entire) system scope.
- The technique is useful in illustrating how information (or object types) will be managed.
- The technique is applicable to the object-oriented (OO) paradigm, but is also applicable to the more traditional function-oriented (FO) paradigm.

2.2 User Interface Topology Chart

The *user interface topology chart* (UITC) is logically constructed from the ITC, and is comparable to Schneideman's *Object-Action Interface* (OAI) model for user interfaces [Schneiderman et. al. 2010]. It shows the operational levels of the software system in a top-down manner: the system is represented at the highest level; subsystems (may) appear at the intermediate levels; actual operations are represented at the lowest level. It is similar to a HIPO chart, except that it favors an OO approach to software design.

The UITC presents operations of the system in a logical manner, showing interrelationships, and how they fit in the overall system architecture. It also presents the end user with a panoramic perspective of the entire system. Hence, as the name suggests, it is useful is portraying a blown out static picture of the (menu driven or graphical) user interface of the system. Figure 3 illustrates a partial UITC for the IMS project.

The UITC provides the following benefits:

- Like the ITC, the UITC is a useful design and documentation aid.
- Like the ITC, the technique is easy to learn, involving minimal use of symbols.
- The technique is useful in conceptualizing (the entire) system scope from an operational perspective.
- The technique is useful in illustrating how various system operations comprising the system come together in a coherent whole.
- The UITC can be used to help identify the area in the software system where there is a problem that needs to be isolated and addressed.

Figure 3: Partial User Interface Topology Chart for the IMS Project

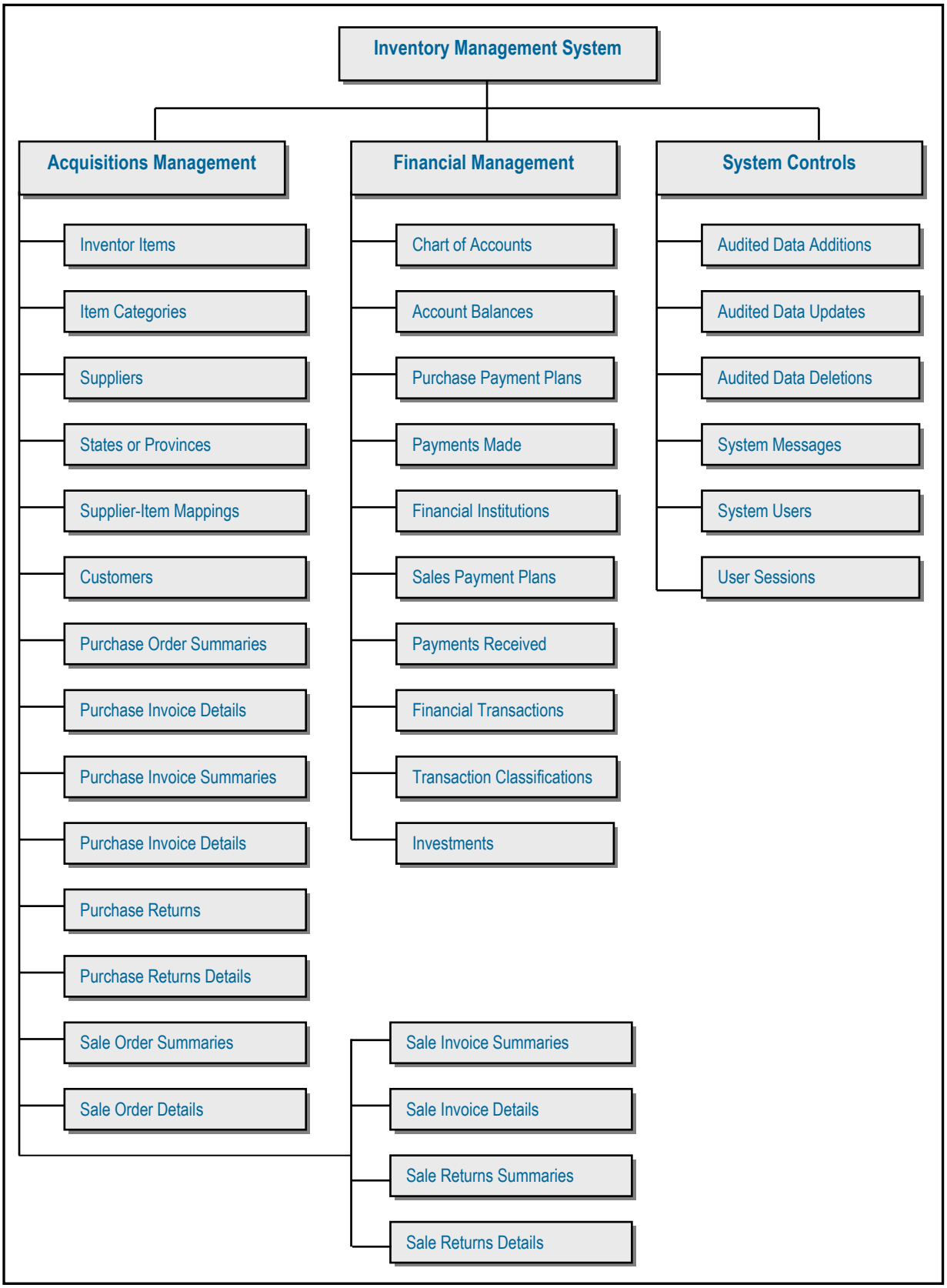
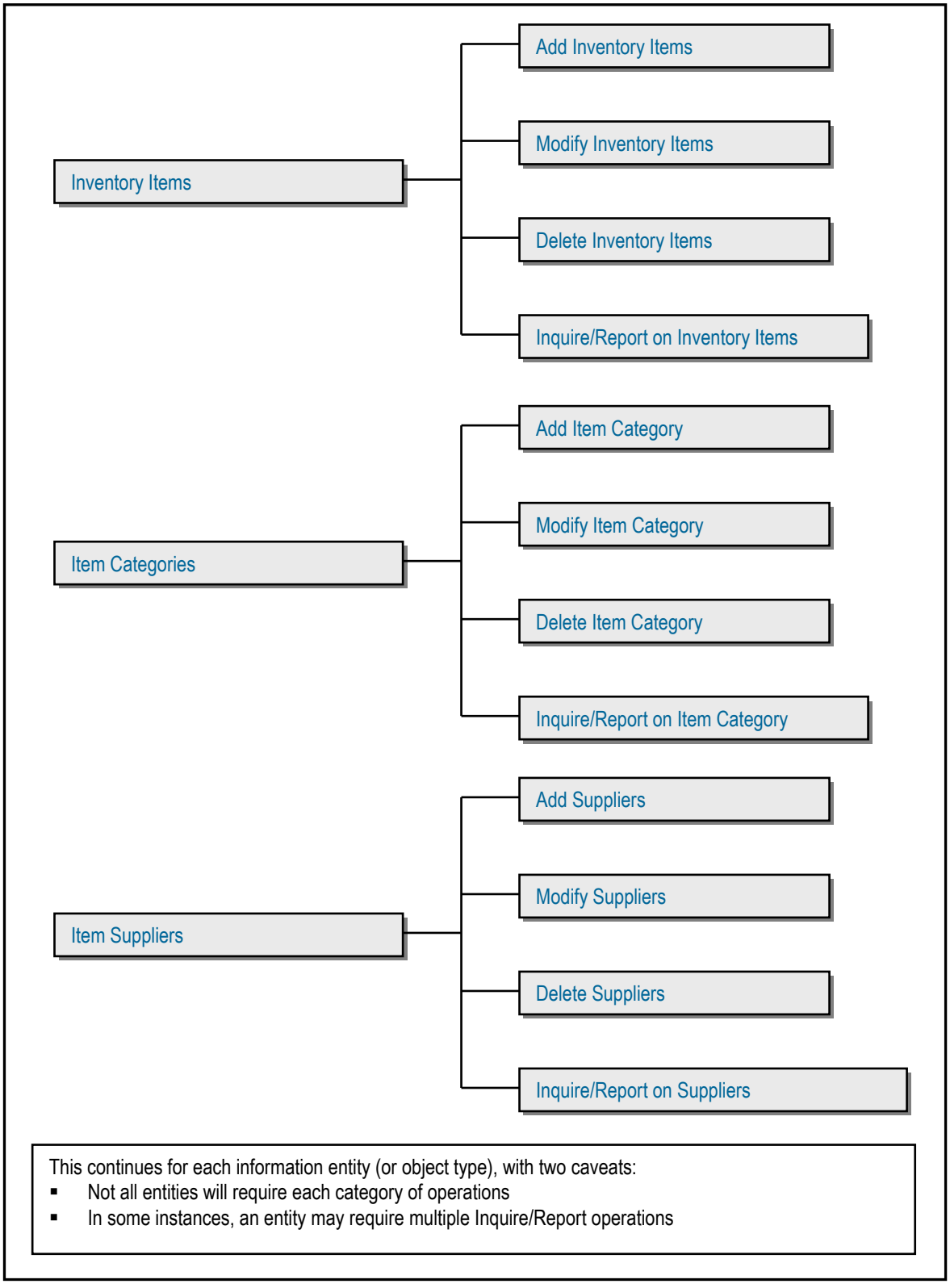


Figure 3: Partial User Interface Topology Chart for the IMS Project (continued)



3. Entity/Object Specification Grid

Many software systems are characterized by underlying databases that provide information support. The database specification may be in different forms, depending on the available resources. In an OO environment where you have the use an OO-CASE tool that supports UML (Unified Modeling Language), it may simply be a detailed ORD (object-relationship diagram) where each object type is represented as a UML class diagram. In an FO or hybrid environment, it may simply be a detailed ERD (entity-relationship diagram) where the attributes of each entity are included on the diagram.

For large, complex software engineering projects (involving huge databases with tens of information entities or object types), unless a CASE or RAD tool which automatically generates the ERD/ORD is readily available, manually drawing and maintaining this important aspect of the project becomes virtually futile. In such cases, an *entity/object specification grid* (E/OSG) is particularly useful. Depending on the context and environment in which it is used, the grid may be referred to as an OSG or an ESG. The grid contains the following components:

- Descriptive name of the entity (or object type)
- Implementation name of the entity (or object type) — typically indicated in square brackets
- Reference identification for each entity, to facilitate easy referencing
- Descriptive name, implementation name (in square brackets) and characteristics (in square brackets) for each attribute
- References (implying relationships) to other entities in the system (indicated in curly braces)
- Comments on the entity and selected attributes
- Indexes (including primary key or candidate keys) to be defined on the entity
- Operations to be defined on each entity (or object type)
- Optionally, implementation names of operations are be indicated in square brackets next to respective operations

The convention for specifying attribute characteristics is to use a letter to represent the nature of the data (A for alphanumeric, N for numeric and M for memo) followed by numbers representing the length and precision (for decimals). Figure 4 provides an illustration of a partial E/OSG for the IMS project. The ESG for three entities are included in the figure. In actuality, there would be one for each entity (or object type) comprising the system. Also note the special data attributes that reference other entities in the figure (E1.3, E1.10, E3.5, and E3.11). In order to determine when to introduce such references, one needs to apply standard principles of database design (as covered in resources such as [Date 2004] and [Kroenke & Auer 2015]).

The E/OSG provides the following advantages:

- The approach is inexpensive, not requiring acquisition of any CASE or RAD tool. Rather, a simple text editor can be used.
- The methodology allows the designer to develop a comprehensive specification of the software system, linking the critical requirements for each information entity (or object type) with the related operational requirements.
- The technique is easy to learn, since it does not require use of multiple symbols.
- The methodology provides useful documentation of the software system, and is therefore useful not only during the design phase, but for subsequent stages in the life cycle of the system.

Figure 4: Partial E/OSG for the IMS Project

E1. Inventory Master [AMInvMaster_BR] consists of the following:

Attributes:

1. Item Code [**AMItmCd**] A8
2. Item Name / Description [**AMItmDes**] [A30]
3. Item Category [**AMItmCat**] **{Refers to E2}** [A4]
4. Quantity on Hand [**AMQty**] [N(7,2)]
5. Reorder Quantity [**AMLowQty**] [N(7,2)]
6. Last Unit Price [**AMLastPrice**] [N(9,2)]
7. Average Unit Price [**AMAvgPrice**] [N(9,2)]
8. Last Selling Price [**AMPrevPrice**] [N(9,2)]
9. Current Selling Price [**AMPrice**] [N(9,2)]
10. Account Number [**AMAcctNum**] **{Refers to E19}** [N10]
11. UPC Code [**AMUPC**] [N12]
12. SKU Number [**AMSKU**] [A6]

Comments:

1. This entity stores information about inventory items
2. Item Code will consist of a 4 byte alphabetic code combined with a 4 digit sequential number.

Indexes:

1. Primary Key: attribute [**1**] (Constraint Name is **AMInvMasterPK**)
2. AMInvMasterNX2 on attribute [**2**]
3. AMInvMasterNX3 on attributes [**3,1**]
4. AMInvMasterNX4 on attributes [**3,2**]
5. AMInvMasterNX5 on attribute [**11**]
6. AMInvMasterNX6 on attribute [**12**]

Valid Operations:

1. **Add** Inventory Item [**AMInvMaster_AO**]
2. **Modify** Inventory Item [**AMInvMaster _MOO**]
3. **Delete** Inventory Item [**AMInvMaster _ZO**]
4. **Inquire** Inventory Item [**AMInvMaster _IO**]
5. **Report** Inventory Item [**AMInvMaster _RO**]

Figure 4: Partial E/OSG for the IMS Project (continued)

E2. Item Category [AMItemCat_BR] consists of the following:

Attributes:
1. Category Code [**AMItemCat**] [A4]
2. Category Description [**AMCatDescr**] [A30]

Comments:
1. This entity stores information about Item Categories
2. Category Code will consist of a 2 byte alphabetic code combined with a 2 digit sequential number.

Indexes:
1. Primary Key: attribute [**1**] (Constraint Name is **AMItemCatPK**)
2. AMItemCatNX2 on attribute [**2**]

Valid Operations:
1. **Add** Item Category [**AMItemCat_AO**]
2. **Modify** Item Category [**AMItemCat_MO**]
3. **Delete** Item Category [**AMItemCat_ZO**]
4. **Inquire** Item Category [**AMItemCat_IO**]

E3. Supplier [AMSupplier_BR] consists of the following:

Attributes:
1. Supplier Code [**AMSuppCode**] [A8]
2. Supplier Name [**AMSuppName**] [A30]
3. A Address Line 1 [**AMAddr1**] [A30]
4. Address Line 2 [**AMAddr2**] [A30]
5. State or Province Code [**AMState**] {**Refers to E4**} [A4]
6. Zip Code [**AMZip**] [N8]
7. Telephone Number(s) [**AMPhoneNo**] [N10]
8. Fax Number [**AMFaxNo**] [N10]
9. Email Address [**AMEmail**] [A30]
10. Contact Person [**AMContact**] [A30]
11. Account Number [**AMAccountNo**] {**Refers to E19**} [N10]
12. Ordering Preference [**AMOrderPref**] [A30]

Comments:
1. This entity stores information about Suppliers
2. Supplier Code will consist of a 4 byte alphabetic code combined with a 4 digit sequential number.

Indexes:
1. Primary Key: attribute [**1**] (Constraint Name is **AMSupplierPK**)
2. AMSupplierNX2 on attribute [**2**]
3. AMSupplierNX3 on attribute [**11**]

Valid Operations:
1. **Add** Supplier [**AMSupplier_AO**]
2. **Modify** Supplier [**AMSupplier_MO**]
3. **Delete** Supplier [**AMSupplier_ZO**]
4. **Inquire** Supplier [**AMSupplier_IO**]
5. **Report** [**AMSupplier_RO**]

This continues for each entity (or object type) comprising the system …

4. Extended Operation Specification

The *extended operation specification* (EOS) approach was developed out of a need for a one-stop resource that provides all the critical information about an operation that will lead to its development with negligible or no setback. In this approach the software engineer records important requirements about an operation in such a manner as to offset or minimize the disadvantages associated with other alternate methodologies. The basic idea is to provide enough detail about the required operation, so that a software developer that pulls the spec should have little or no problem in writing the operation (program). The technique is applicable to both OOD and the more traditional FOD.

The EOS requires the specification of operational requirements in the following areas:

- **Operation Biography:** This includes the operation's descriptive name, implementation name, a brief description, operation classification, complexity rank, related system and subsystem, and spec author.
- **Inputs:** This includes all files and other resources from which the operation pulls information.
- **Outputs:** This includes all files to which the operation writes data, as well as other forms of media outputs (such as monitor display, print, and audio).
- **Validation Rules:** All data validation and integrity constraints that must be upheld are specified here.
- **Special Rules/Notes:** This includes formulas or special procedures/guidelines that are required.
- **Operation Outline:** This outlines the basic logic or algorithm that the operation should implement. This may include any combination of pseudo-code, Warnier-Orr diagram, activity diagram, or collaboration diagram.

Observe that while the approach prescribes a structure for the specification of operational requirements, it does not inhibit creativity. The software engineer has the flexibility of framing the operation outline in a manner that suits the prevailing circumstance(s). Relating this to the IMS project, each operation mentioned on the E/OSG would have its own EOS. By way of example, figures 5 – 7 provide examples of EOSs for various operations relating to the management of inventory items.

The following are some benefits associated with the EOS methodology:

- It allows the software engineer to pack all the relevant information about an operation into one spec so that development is easy.
- It provides information that allows the project manager to make intelligent work assignments during software development.
- Under the operation outline section, the software engineer has the flexibility of using any combination of pseudo-code, Warnier-Orr diagram, activity diagram, or collaboration diagram to clearly specify the required logic of the operation.
- Important information such as I/O requirements, categorizations, etc. can be included in the spec.
- The whole process of specifying an EOS for each operation can be automated by developing a software system for that purpose.

Figure 5: EOS for Inventory Addition

Operation Biography:
System Name: Inventory Management System
Subsystem Name: Acquisitions Management
Operation Name: **AMInvMaster_AO**
Operation Description: Facilitates addition of items to the Item Master table.
Operation Category: Mandatory Complexity Rank: 6 of 10
Spec. Author: E. Foster Date: 7-10-2010

Inputs:
New Item Form
AMInvMaster_BR — Inventory Master File (E1)
AMItemCat_BR — Item Categories (E2)
FMChrtAccts_BR — Chart of Accounts (E19)

Outputs:
AMInvMaster_BR — Inventory Master (E1)

Validation Rules:
1. Item Code must not previously exist
2. Category Code must already exist in **AMItemCat_BR**
3. Blank Item Name not allowed
4. Account Number must already exist in **FMChartAccts_BR**

Special Notes:
When a new Item is added or purchased, the Quantity on Hand and the Quantity Owned are adjusted.

Operation Outline (Pseudo-code):

```
START
WHILE (User wishes to continue)
        Accept Key Field(s);
        Check Record Absence or Existence in file AMInvMaster_BR;
        IF      (Record Absent)
                Accept Non-key Fields;
                Validate Non-key Fields based on Validation Rules;
                WHILE (Any Error Exists),
                        Re-display Non-key Fields for possible Update;
                        Display appropriate error message(s);
                        Validate Non-key Fields based on Validation Rules;
                END-WHILE;
                Re-display full Record for confirmation;
                IF      (Confirmation Obtained)
                        Write New Record to file AMInvMaster_BR;
                        Write New Record to audit file for Additions;
                ENDIF;
                ELSE    Inform the User that nothing was saved;
                END-ELSE;
        ENDIF;
        ELSE    Display Message ('Record already exists');
        Check if User wishes to quit and set an exit flag if necessary;
END-WHILE;
Generate Edit-List;
STOP
```

Figure 6: EOS for Inventory Modification

Operation Biography:
System Name: Inventory Management System
Subsystem Name: Acquisitions Management
Operation Name: **AMInvMaster_MO**
Operation Description: Facilitates modification of items to the Item Master table.
Operation Category: Mandatory Complexity Rank: 6 of 10
Spec. Author: E. Foster Date: 7-10-2010

Inputs:
AMInvMaster_BR — Inventory Master File (E1)
AMItemCat_BR — Item Categories (E2)
FMChrtAccts_BR — Chart of Accounts (E19)

Outputs:
AMInvMaster_BR — Inventory Master (E1)

Validation Rules:
1. Item Code must not previously exist
2. Category Code must already exist in **AMItemCat_BR**
3. Blank Item Name not allowed
4. Account Number must already exist in **FMChartAccts_BR**

Special Notes: None.

Operation Outline (Pseudo-code):

```
START
WHILE (User wishes to continue)
        Accept Key Field(s);
        Check Record Absence or Existence in file AMInvMaster_BR;
        IF      (Record Present)
                Retrieve Record and update Audit Log Fields (with before-values);
                Display Non-key Fields for possible Update;
                Validate Non-key Fields based on Validation Rules;
                WHILE (Any Error Exists),
                        Re-display Non-key Fields for possible Update;
                        Display appropriate error message(s);
                        Validate Non-key Fields based on Validation Rules;
                END-WHILE;
                Re-display full Record for confirmation;
                IF      (Confirmation Obtained)
                        Update Audit Log Fields (with current-values);
                        Write New Record to audit file for Updates;
                        Update Record in file AMInvMaster_BR;
                ENDIF;
                ELSE    Inform the User that nothing was saved;
                END-ELSE;
        ENDIF;
        ELSE    Display Message ('Record does not exist');
        Check if User wishes to quit and set an exit flag if necessary;
END-WHILE;
Generate Edit-List;
STOP
```

Figure 7: EOS for Inventory Inquiry

Operation Biography:
System Name: Inventory Management System
Subsystem Name: Acquisitions Management
Operation Name: **AMInvMaster_IO**
Operation Description: Facilitates inquiry on inventory items
Operation Category: Mandatory Complexity Rank: 8 of 10
Spec. Author: E. Foster Date: 7-10-2010

Inputs:
AMInvMaster_BR — Inventory Master File (E1)
AMItemCat_BR — Item Categories (E2)
FMChrtAccts_BR — Chart of Accounts (E19)

Outputs: Monitor/Printer

Validation Rules: None

Special Notes:
1. It will be possible to query Items via any of the following access paths:
1.1 By Category and Item Name
1.2 By Item Category & Code
1.3 By Account Number
1.4 UPC Code or SKU number
2. Each option will invoke one of four sub-operations (**AMInvMaster _I1, AMInvMaster _I2, AMInvMaster _I3,** or **AMInvMaster _I4**).
3. Utilizes the logical views (**AMInvMaster _LV1, AMInvMaster _LV2, AMInvMaster _LV3,** and **AMInvMaster _LV4)**, each of which joins **AMInvMaster_BR** with **AMItemCat_BR** and **FMChrtAccts_BR**.

Operation Outline (Pseudo-code):

```
START:  /* Inquire */
WHILE (User Wishes to Continue)
        Present the User with the options mentioned above;
        Depending on the User's choice, Invoke one of the sub-operations;
END-WHILE;
STOP.
```

Outline for AMInvMaster_I1:

```
START
WHILE (User Wishes to Continue)
        Prompt user for Item Name;
        Starting at that point in AMInvMaster _LV1, Load a Virtual Data Collection Object with all records
                until End-of-File;
        Display the Virtual Data Collection Object;
END-WHILE;
STOP.
```

{The other sub-operations will be similar}

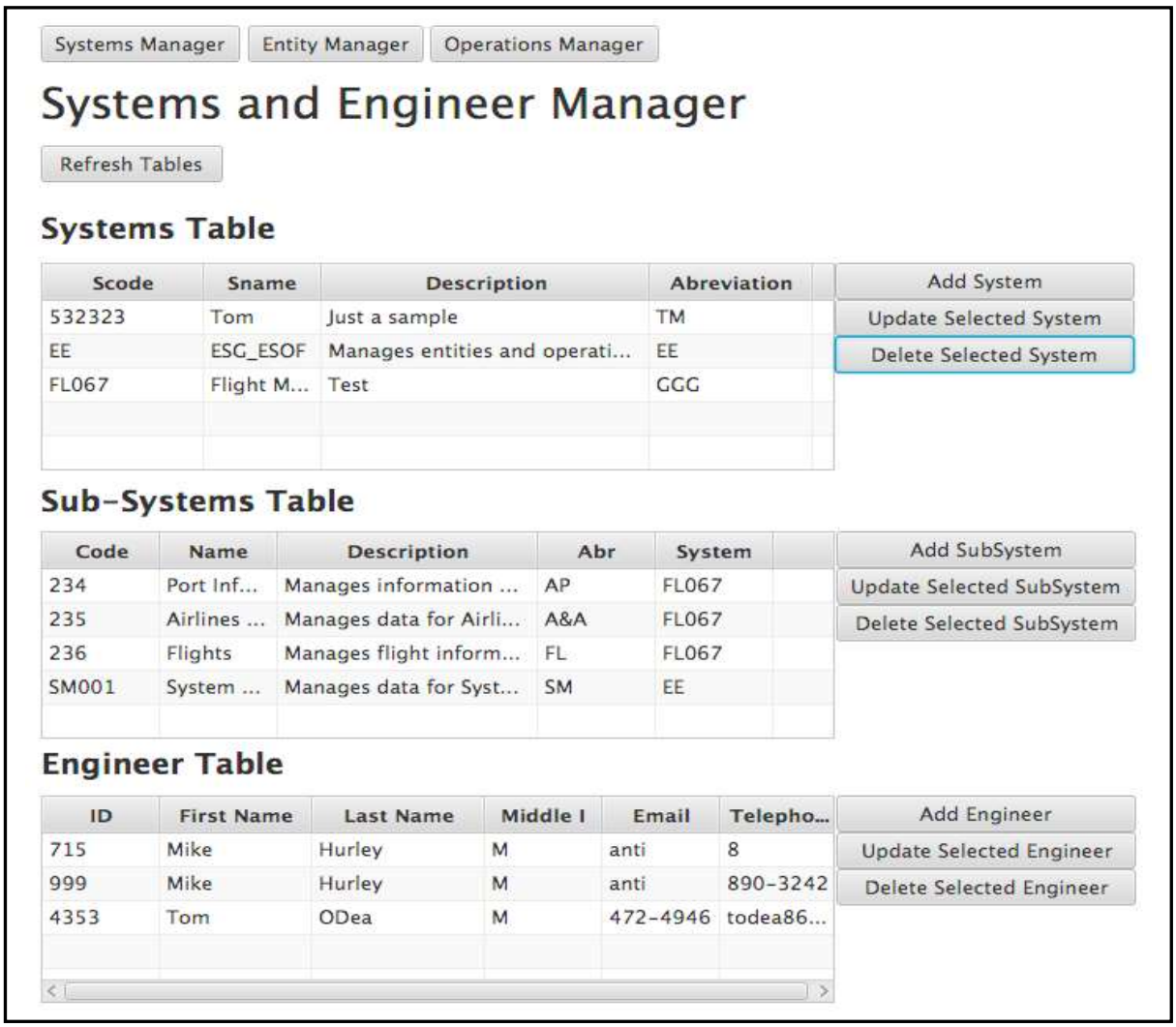
5. Recent Additional Work

Recently, two undergraduate students at Keene State College have conducted work on a guided software engineering project, which has resulted in the design, construction, and testing of a software prototype to automatically generate the ESG and EOS for any software engineering project. The project is called the ESG-EOS Facilitator (ESG-EOSF, abbreviated EE). The prototype operates like a CASE tool, allowing users (who typically would be software engineers and/or developers) to specify the requirements of a software system as inputs to the prototype. This information is then used to automatically generate the ESGs and EOSs for the various entities and operations comprising the target software system.

The software engineer using the ESG-EOSF is allowed to specify the requirements of the software system being designed under three captions, namely, *Systems Manager*, *Entities Manager*, and the *Operations Manager*:

The **Systems Manager** component allows for the definition of software systems being designed as well as their respective component subsystems. Figure 8 provides a screen-shot from this component. In the figure, you will notice specifications for a Flight Management System (FMS) — another software system on which the ESG-EOSF is being tested.

Figure 8: Screen-shot from the ESG-EOSF Systems Manager Component



The **Entities Manager** component allows for the definition and specification of entities comprising previously defined systems and/or subsystems. Entities are specified in terms of predefined criteria mentioned in section 3. Figures 9 and 10 provide representative screen-shots, again the aforementioned FMS being used as a case study. This information is then used to generate the ESGs for the software system being designed.

Figure 9: Screen-shot from the ESG-EOSF Entities Manager Component

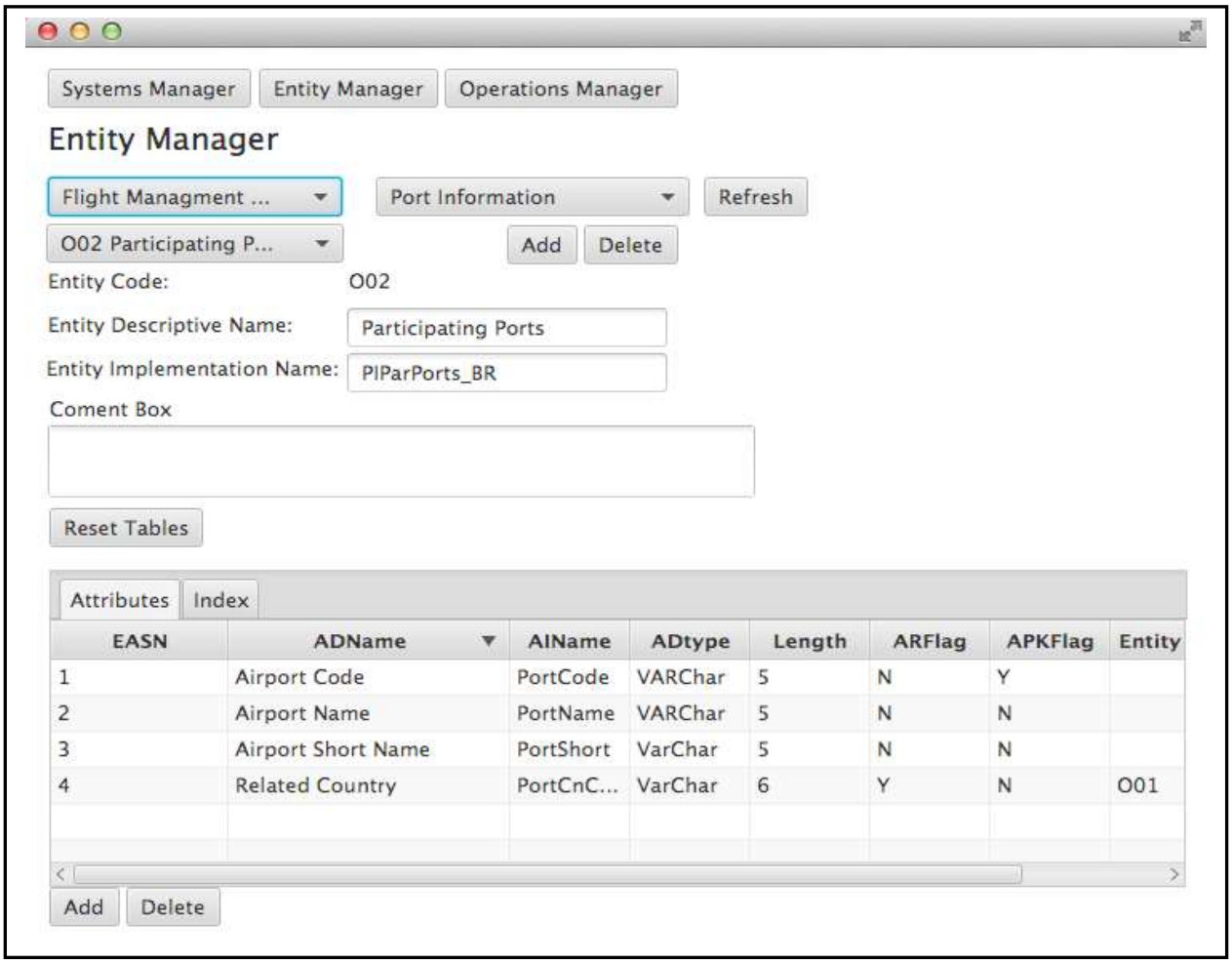
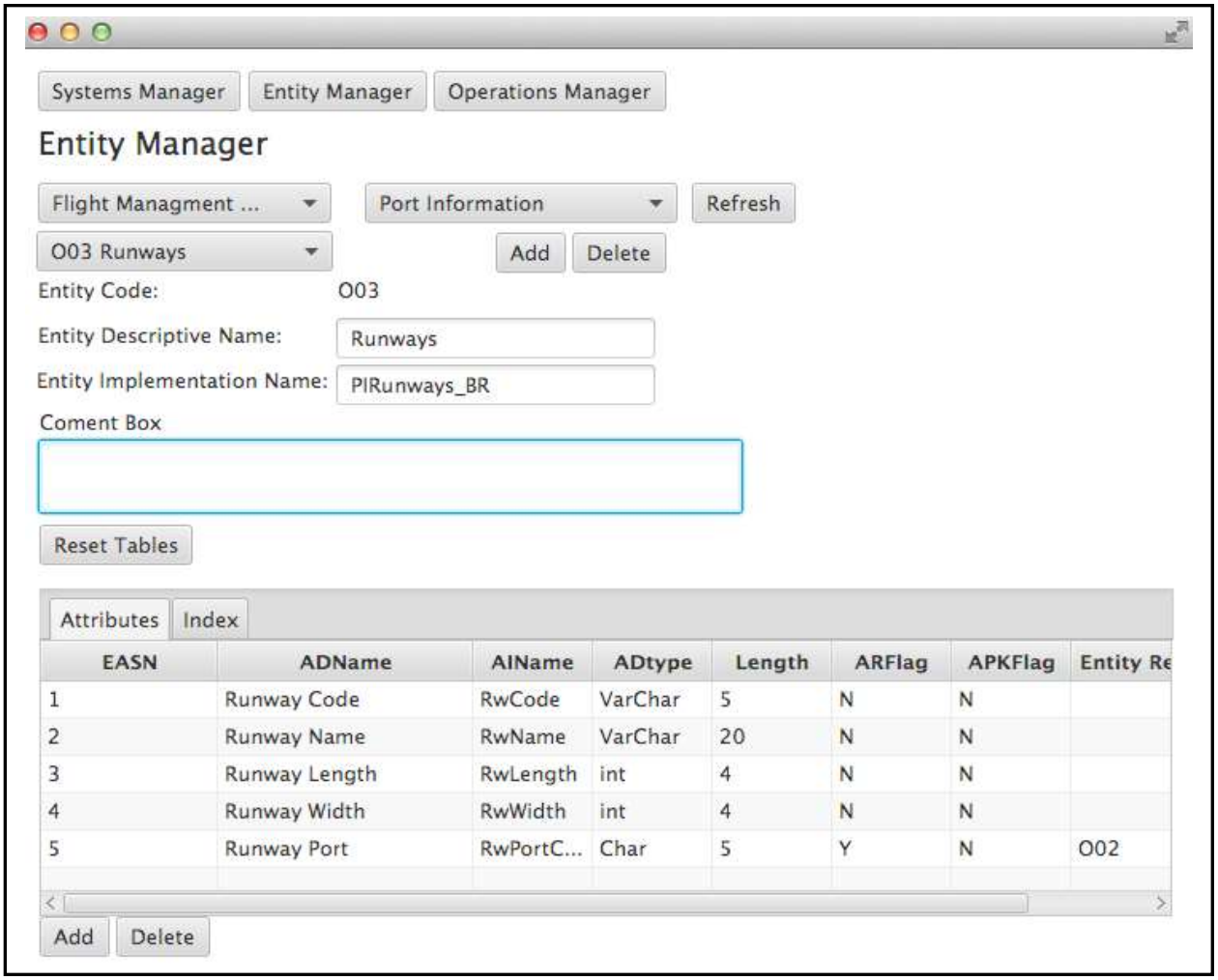


Figure 10: Another Screen-shot from the ESG-EOSF Entities Manager Component



The **Operations Manager** component facilitates definition of operations for various systems and/or subsystems, and specification of the requirements for these operations. Similar to the Entities Manager component, operations are specified based on predefined criteria mentioned in section 4. The specified information is then used to generate the required EOSs for operations comprising the software system being designed.

The results from this effort are encouraging, though additional work and refinements are needed. The findings from this additional work will be the subject of a future publication.

6. Summary and Concluding Remarks

This paper has proposed three methodologies that could add richness and additional flexibility to the software engineering experience. They are the system topology charts, entity/object specification grid (E/OSG), and the extended operation specification (EOS). The system topology charts discussed are the information topology chart (ITC) that presents the object types and/or information entities in the way they are managed in the software system, and the user interface topology chart (UITC) that presents operations the

way they appear in the menu hierarchy of the software system. The E/OSG adopts the conventions of the UML class diagram, but expands it to include additional critical information that lead to better software construction. The EOS embraces the UML guidelines for the activity diagram, but is flexible enough to include other techniques such as pseudo-code, Warnier-Orr diagram, and collaboration diagram. The methodology also allows for the specification of other critical information not covered in these techniques.

These proposed methodologies been documented in [Foster 2014] with additional illustrations. Moreover, they have been tested with software engineering students on various projects, and have produced very encouraging results. One such project is the inventory management system (IMS), which was used as a case study in the paper. One student was successful in employing these methodologies (along with others) to develop and implement such a system for his family business. Another group of students have applied these methodologies in the development of a prototype for a National Football League Management System (NFLMS) to keep track of activities and issues related to teams, coaches, players, and games. A third group has successfully applied the methodologies in the development of a prototype for a generic National Sports League (NSF), which can then be tailored for any national sports program. A fourth group has developed a working prototype for a Flight Management System (FMS) that could be configured for any airport.

The methodologies have been successfully to several software engineering projects not mentioned in this paper. However, the final initiative worthy of mention here is the project to automate the methodologies discussed herein. A group of students has successfully developed a prototype that attempts to automate the ESG and EOS methodologies as described earlier; the user interface developed allows a software engineer to key in basic information about a software engineering project, and this information is used to generate a corresponding ESGs and EOS instances. This latter initiative is currently being refined, and will be the subject of a subsequent article.

It must be emphasized that these methodologies are not applicable in all situations. They are best suited in situations where any combination of the following conditions holds:

- The software system being developed requires an underlying database
- The underlying database is very large and/or complex
- An appropriate CASE or RAD tool is not readily available
- It is desirable to provide detailed database specifications so that the required database can be easily constructed
- It is desirable to provide detailed operation specifications that a software developer can use to construct the required operations without any significant problems

It is anticipated that as more people use these methodologies, they will be further refined to produce even better results.

References

[Bruegge & Dutoit 2010] Bruegge, Bernard and Allen H. Dutoit. 2010. *Object Oriented Software Engineering Using, Patterns, and Java,* 3rd ed. Boston: Pearson.

[Date 2004] Date, Christopher J. 2004. *Introduction to Database Systems* 8h ed. Menlo Park, California: Addison-Wesley.

[Foster 1999] Foster, Elvis C. *Labour Market Information System*. PhD Dissertation, Mona, Jamaica: Department of Mathematics and Computer Science, University of the West Indies, 1999. See section 4.4.1.

[Foster 2014] Foster, Elvis C. 2014. *Software Engineering: A Methodical Approach*. New York: Apress Publishing.

[Hoffer, Venkataraman, & Topi 2013] Hoffer, Jeffrey A., Ramesh Venkataraman, & Heikki Topi. 2013. *Modern Database Management* 11th ed. Boston: Pearson.

[Jacobson 1991] Jacobson, Ivar. 1991. *Object-Oriented Software Engineering*. New York, NY: ACM.

[Kendal & Kendal 2014] Kendal, Kenneth and Julie Kendal. 2014. *Systems Analysis and Design,* 9th ed., Boston: Pearson.

[Kroenke 2015] Kroenke, David M & David Auer. 2015. *Database Concepts 7th ed*. Boston: Pearson.

[Marca & McGowan 1987] Marca, David A. and Clement L. McGowan. 1987. *Structured Analysis and Design Technique*. Upper Saddle River, NJ: Prentice Hall.

[Martin & McClure 1985] Martin, James and Carma McClure. 1985. *Structured Techniques for Computing*. New York, NY: McGraw-Hill.

[Martin & Odell 1993] Martin, James and James Odell. 1993. *Principles of Object Oriented Analysis and Design*. Englewood Cliffs, New Jersey: Prentice Hall.

[Rumbaugh et al. 1999] Rumbaugh, James, Ivar Jacobson, and Grady Booch. 1999. *The Unified Modeling Language Reference Manual*. Reading, MA: Addison-Wesley.

[Schneiderman et. al. 2010]: Schneideman, Ben.Catherine Plaisant, Maxine Cohen, and Steven Jacobs. 2010. *Designing the User Interface: Strategies for Effective Human-Computer Interaction 5th ed*. Boston: Pearson.

[Sedgewick 2002] Sedgewick, Robert. 2002. *Algorithms in C++: Part Five, Graph Algorithms* 3rd ed. Boston, MA: Addison-Wesley.