Session 5: The Data Dictionary

- relationship to systems analysis methodologies
- relationship to project management
- data definition vs. data representation
- taxonomy of data types

Ways of documenting detailed requirements
- Flowcharts + record layouts
- "Victorian novel"
- Structured analysis
- Object-oriented analysis
- UML with use-cases
- Discrete requirements list
- Incremental approach / stories

No matter which one(s) you choose:
  - There's one component you should always have.
  - It goes with any of them.

The Project Data Dictionary

- A repository of rigorous definitions of every data item mentioned in the requirements documents.
  - Its unique name
  - Its type
  - Its precise meaning, not the representation.
    (What it is, not what it looks like)
  - Its attributes in the real world

A data dictionary may be maintained

- With a specialized software product
  - An independent data-dictionary program
  - Part of a C.A.S.E. tool
  - Some are also data directories

- With a general software product
  - A spreadsheet processor
  - A database manager
  - A word processor

- Manually
  - Stack of index cards or forms

What is it?
Many projects don't!

- If a project doesn't create and maintain a data dictionary, it may define data items:
  - through inline documentation or footnotes
    - This is particularly common with use-cases and user stories.
    - We call that an implicit data dictionary.

  What's wrong with that?

  - or not at all!

Important!!

- The lack of a project data-dictionary is a common cause of failure of application development projects.

  What do we mean by "failure"?
  a. serious schedule overrun
  b. serious cost overrun
  c. never satisfies the users
  d. combination of the above

An implicit data-dictionary example

"Customers who place more than $10,000 business per year and have a good payment history or have been with us for more than 20 years are to receive priority treatment."

- a business rule from a respected systems analysis textbook

  We saw that before!

  Are there data items there that should be defined?

  If so, should the business rule be rewritten?

Obvious questions

"Customers who place more than $10,000 business per year and have a good payment history or have been with us for more than 20 years are to receive priority treatment."

- For how many years must a customer have purchased more than $10,000 worth of merchandise? How recently?

- What is a good payment history?

- During how many years during that twenty-year period and how much must a customer have spent to have been with us?

- What does priority treatment mean?
  How should those questions be answered?
Computer programs manipulate two kinds of data

- **Application domain data**
  - Exist in the real world
  - Usually known to users / sponsors
  - May be persistent or transient

- **Program data**
  - Have no real-world existence
  - None of the users’ business
  - Usually transient

A data dictionary is concerned only (or mainly) with application-domain data. Why?

A taxonomy of data types

- The 3 fundamentally different kinds of data
- Some basic subtypes of those 3.

This is all independent of any programming language.

Data items: 3 kinds

1. **Elementary** items are
   - not composed of other data items
   - sometimes called "fields" or "elements"
     (when part of a composite item).
   - defined in terms of their real world meaning

2. **Composite** items are:
   - composed of other data items, which may be either elementary or composite,
   - sometimes called "structures", "records", "blocks", "data flows",
   - also called "entities" or "subjects" when they play a primary role in an application system
   - defined mainly in terms of their components.

Data items: 3 kinds

3. **Container** items are
   - structures that hold other data items, usually either elementary or composite (sometimes other containers).
   - either:
     - static (staying the same size and shape throughout their life span), or
     - dynamic (growing, shrinking, or reconfiguring)
   - either:
     - homogeneous (all elements are of the same type), or
     - heterogeneous (multiple kinds of data can be elements)
   - defined in terms of their behavior
Elementary data items

- Every *elementary* data item belongs to one (and only one) of these basic types.
  - **discrete** (or coded or enumerated)
    - possible values belong to a finite set
  - **numeric**
    - some arithmetic operation is meaningful
  - **text** (or character string)
  - **logical** (or Boolean or switch or indicator)
    - 2 possible values (T/F, Y/N, on/off, 0/1, present/absent, . . .)

- Are there any others?
- How do C#, Java, etc. designate them?

Which of the four elementary subtypes do these belong to?

- MaritalStatus
- EmployeeName
- InterestRate
- TelephoneNumber
- DueDate
- BookTitle
- CreditLimit
- Sex
- Weight (of a shipment)
- Velocity
- City
- State
- ZIP code
- Color (of a product)
- Price (of an item)
- CreditApproved

Attributes (or properties) of elementary data items

- Attributes of **numeric** data items:
  - unit of measure
  - range
  - precision
  - scale

- Attributes of **text** data items:
  - length
  - internal format, delimiters

- Attributes of **discrete** data items:
  - number of possible values (both current and potential)
  - coding structure

Avoid false numerics

- Many *discrete* data items are represented by a sequence of numeric digits.

  Examples?

- That doesn't make them numeric.

  Why not?

  - But many of them have "number" as part of their name, e.g. accountNumber

- We emphasize what a data item *is*, not what it *looks like*. Some old-fashioned tools take the opposite view.

  Which ones?
Attributes of *composite* data items

- List of component data items with rules for including them
- Can often be defined by a language-dependent commented structure definition (C `struct`).
- De Marco's language-independent notation for specifying a composite data item:
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>is composed of</td>
</tr>
<tr>
<td>+</td>
<td>followed by</td>
</tr>
<tr>
<td>( )</td>
<td>optional</td>
</tr>
<tr>
<td>{ }</td>
<td>iteration</td>
</tr>
<tr>
<td>[ ]</td>
<td>alternatives</td>
</tr>
</tbody>
</table>

Example using DeMarco notation:

```
EMPLOYEE INFO = NAME
   + DATE OF BIRTH
   + MARITAL STATUS
   + {DEPENDENT INFO}^o
   + DATE HIRED
   + [HOURLY WAGE
     MONTHLY SALARY]
   + (PREVIOUS EMPLOYER)
   + etc.
```

Examples of container data

- Static structures
  - Arrays

- Dynamic structures:
  - lists, stacks, queues
  - trees, graphs
  - dynamic arrays

- External structures:
  - files, data bases
  - display / interface (GUI) objects

Terminology update

- Java uses the term *container* in a different and narrower sense, to mean a GUI screen object (e.g. a window) that can contain other GUI objects.

- Java now uses the term *collection* in the more general sense, where we've been using *container*.

- Use whichever term you prefer as long as the context is clear.
Taxonomy summary:

the top of the tree

Data item

Elementary item
- Discrete item
- Numeric item
- Text item
- Logical item

Composite item
- Records
- Entities
- Other static tree structures

Container item
- Arrays, vectors, matrices, tables
- Lists, stacks, queues
- Trees, Graphs
- Files, databases
- Windows, boxes

Which ones are suited to being represented as object-oriented classes?

Examples

- Discrete
  - COLOR (of a product)
  - ACCOUNT NUMBER
  - STATE (in an address)
  - MARITAL STATUS
  - BRANCH OFFICE CODE
  - CREDIT CARD TYPE
  - TELEPHONE NUMBER

- Numeric
  - HOURS WORKED (of an employee)
  - TEMPERATURE (of a substance)
  - QUANTITY ORDERED (of a product)
  - DATE SHIPPED (of an order)
  - MASS (of a body)
  - SPEED (of an object)
  - DEPARTURE TIME (of a train)

- Text
  - NAME (of an employee)
  - DESCRIPTION (of a product)
  - CITY (in an address)
  - DUNNING LETTER

- Logical
  - CREDIT APPROVAL (for an order)
  - NEW CUSTOMER FLAG
  - UNION MEMBER (for an employee)
  - AUDIT TRACE OPTION

- Entity / subject
  - EMPLOYEE
  - VENDOR
  - PRODUCT
  - VEHICLE

- Other
  - HOME ADDRESS (of an employee)
  - CREDIT HISTORY (of a customer)
  - SUBASSEMBLY (of a product)
  - CURRENT ORBIT (of a satellite)

Avoid false composites

- Don't confuse a mixed unit or structured representation of an elementary item with a true composite item.

- For example, these should be treated as elementary, not composite, items:
  - Date = year + month + day
  - Person Name = first + middle + last
  - Time = hours + minutes + seconds

Data classes and inheritance

- Each basic data type can be divided into subtypes or data classes.

- Each such class can in turn be further divided into subclasses to any level.
**Data classes and inheritance**

- Each class **inherits** the properties of its parent classes:
  - data representation, esp. internal, and attributes
  - associated functions and operations
- This **inheritance principle** can greatly simplify the definition of both data items and the functions or processes that operate on them (even without any object-oriented tools.)

**Example**

- We can define `billingAddress` as an instance of `MailingAddress`, where `MailingAddress` is a **class** that defines the structure and content of an address acceptable to post offices.
  - Derived subclasses could include `USStreetAddress`, `USBoxNoAddress`, `CanadaAddress`, etc.
- It would then be redundant (and wrong) to define attributes of the individual data item `billingAddress`, such as
  - number of lines for the street address portion and their maximum length
  - format of a ZIP code

**Classes and data items**

- In a class hierarchy, properties (or attributes) of a class T are inherited by both:
  - subclasses of T
  - specific data items (instances, **objects**) of type T
- It is, therefore, unnecessary and undesirable (why?) to specify attributes in the dictionary definition of every data item.
  - Nevertheless, many older tools (COBOL, some data-dictionary systems, etc.) demand that we do so!

> Would the "year-2000 crisis" have occurred, if everyone had understood this principle?

**Classes (or types) versus data items (or instances or objects)**

- Inexperienced systems analysts and application designers sometimes confuse these two very different things.
- For example, these are appropriate names for **classes**, but not for **data items**.
  - `temperature`
  - `date`
  - `address`
- Data items need more specific names!
Ambiguous data names

- **temperature**: What is it the temperature of?
  - When (starting temperature or current)?

- **date**: What is it the date of or for?
  - shippingDate
  - orderDate
  - expirationDate
  - etc. etc.

- **address**: What is it the address of?
  - mailing address
  - billing address
  - home address