
# Test- och mätsystemutveckling, kvalitetssäkring och effektivisering

# Presentation

- **Mattias Ericsson**
  - LabVIEW developer >10 years
  - LabVIEW Partner Program
    - G# Framework
    - G# StarUML
    - Free, open source tool
    - LabVIEW Add-On of the Year for Community 2011
    - *www.ni.com/labviewtools*
    - *www.addq.se/gsharp*

NATIONAL INSTRUMENTS LabVIEW CERTIFIED DEVELOPER

COMPATIBLE WITH NATIONAL INSTRUMENTS LabVIEW GOLD PRODUCT

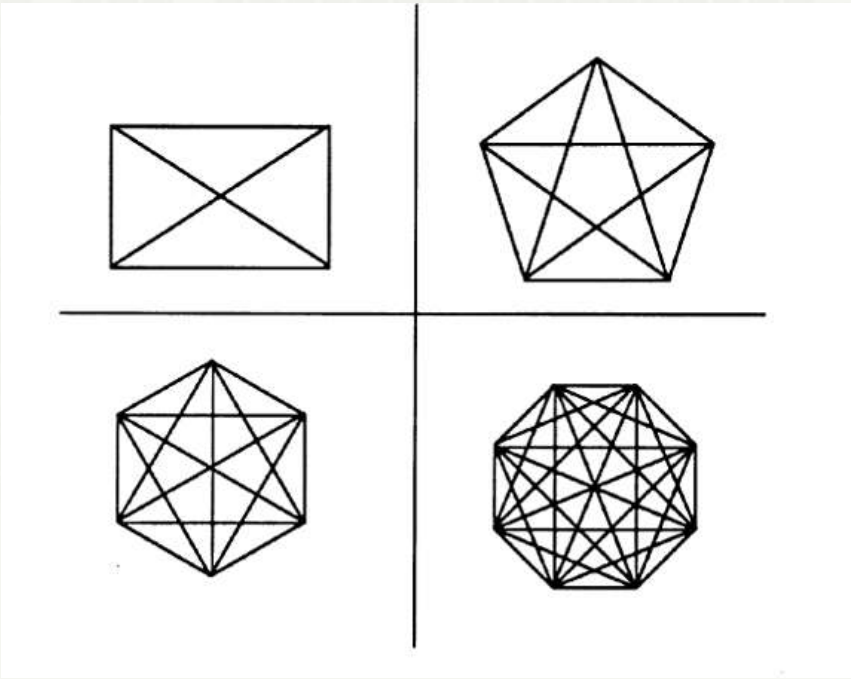
AddQ CONSULTING

# Agenda

- Design Fundamentals
- Analysis
- Use Cases
- Data Models
- Example: Data Modeling

# What is Design All About?

- Managing complexity and dependencies
- Tradeoffs and priorities
- Nondeterministics
- Hard to know when its ”good enough”



*”No one’s skull is really big enough to contain a modern computer program” – Edgar Dijsktra 1972*



AddQ CONSULTING

# The Goal of Design

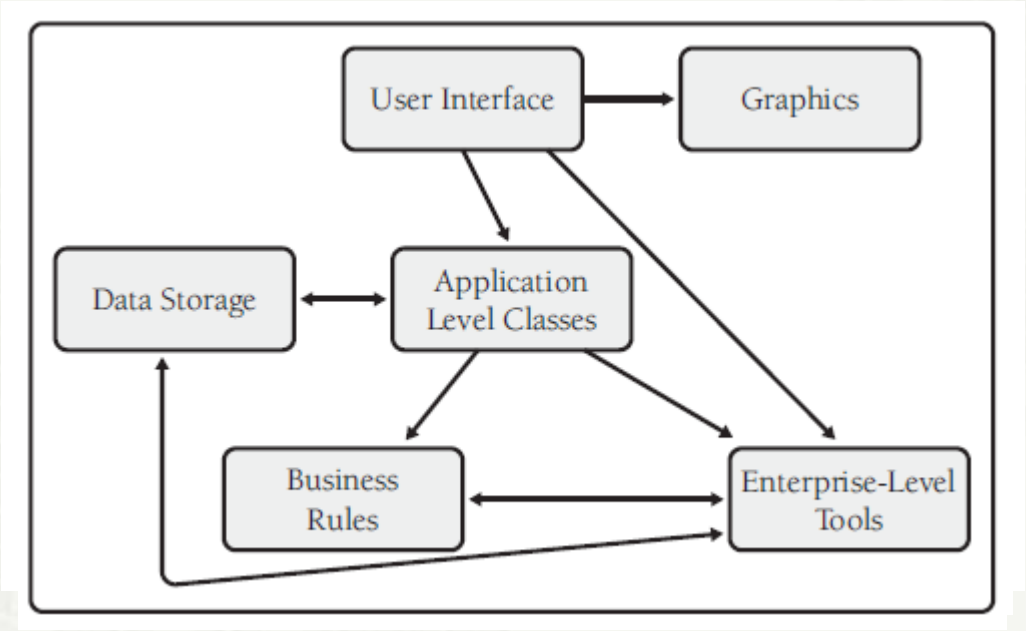
- Solve the problem
- Break down problem into simple pieces
- Minimize complexity
- Design for change
- Loose coupling

*”Make everything as simple as possible, but not simpler”*
*– Albert Einstein*



AddQ CONSULTING

# Subsystem Dependecies



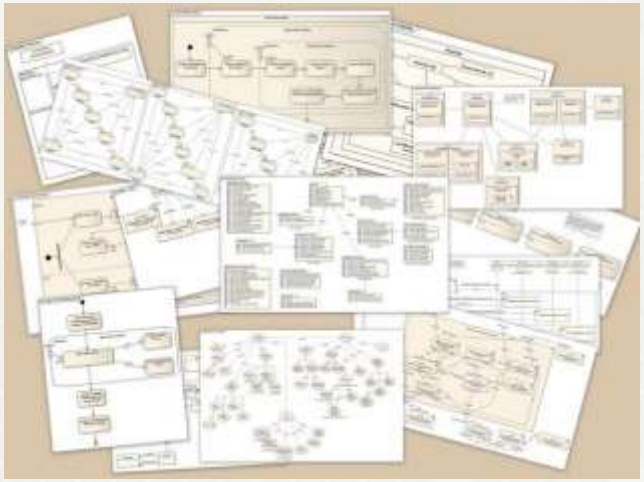


***Dependency graph should be an acyclic graph***

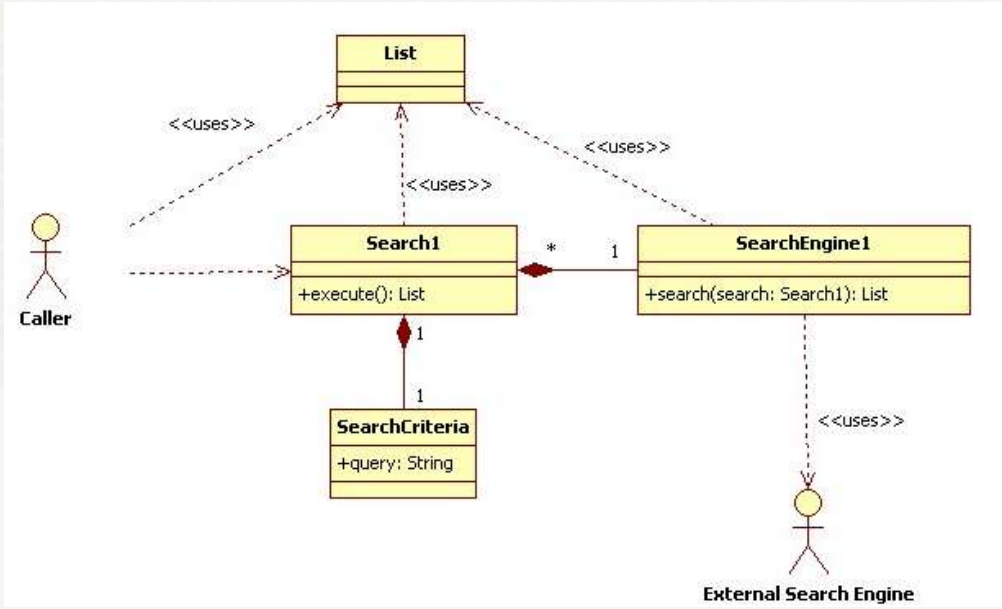


AddQ CONSULTING

# UML



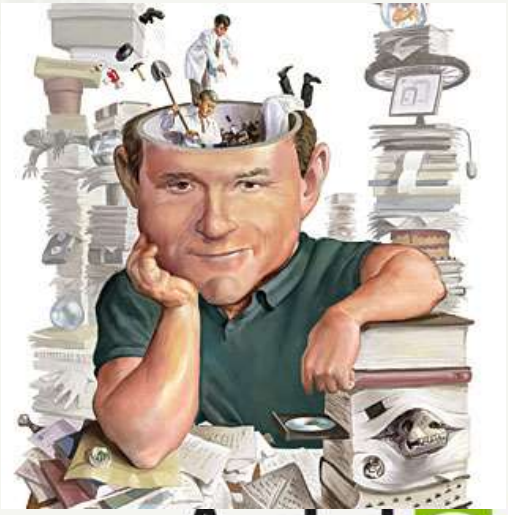
- UML Modelling Language
  - Structural
    - Class Diagram
  - Behavioral
  - Interaction
- Tools
  - ”White Board”
  - StarUML




AddQ CONSULTING

# What about analysis?

- Analysis is all about ***understanding*** the problem!
- Analysis focuses on ***what*** the system does!
- Design focuses on ***how*** the system should be built
- The output of the analysis is
  - Conceptual model of the problem
  - Functional requirement specification
  - Use cases



Add Q CONSULTING

# Analysis Models

- Analysis models are meant for conceptual understanding only

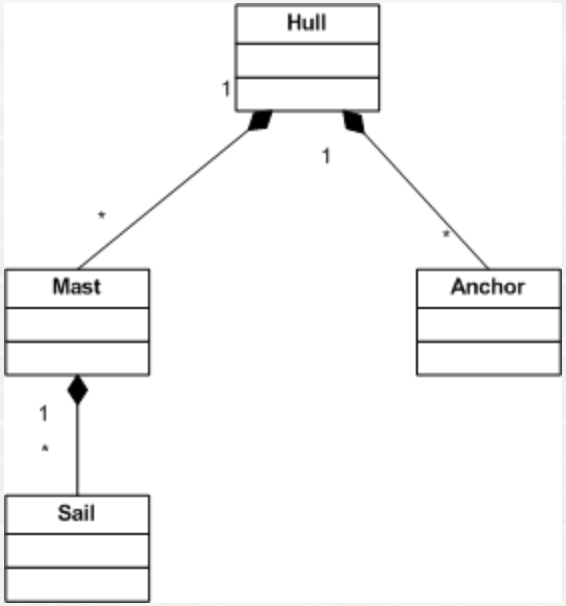
- The subject area to which the user applies a program is the ***domain*** of the software.

***Abstraction that describes a selected aspect of a domain***

*"Assumption is the mother of all fuckups" – Olaf Maassen*

AddQ CONSULTING

# Models







AddQ CONSULTING

# Modeling Principle

- The simplest model is not neccessary the first one you think of
- Use the simplest model possible

***Modelling principle***
*Models are not right or wrong; they are more or less useful*

# Functional Requirements

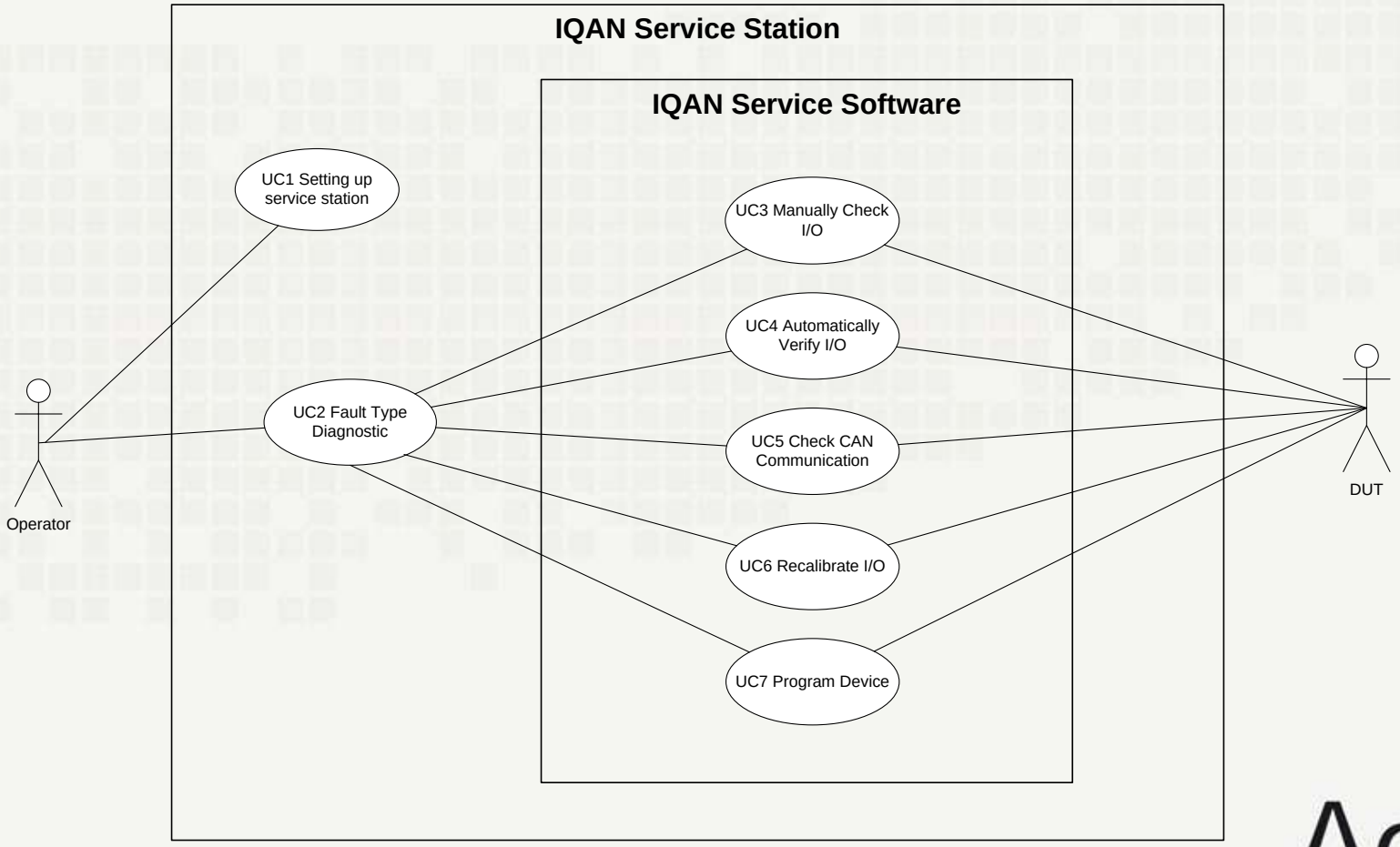
- Capture the intended behavior of the system
  - Services
  - Tasks
  - Functions
- Avoid using GUI image
- *Use cases* have quickly become a widespread practice for capturing functional requirements

# Use Cases

- A description of steps or actions between a user (or "actor") and a system which leads the user towards something useful

- Each use case focuses on describing how to achieve a goal or a task

# Use Case Example - UML



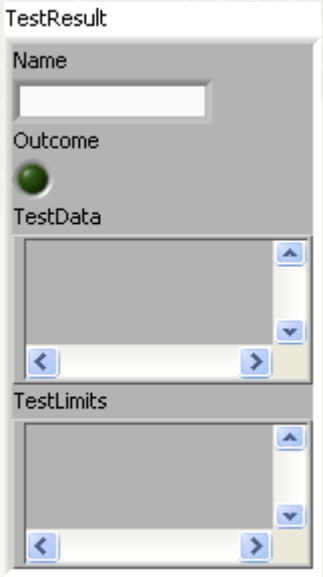
# Use Case Example - Text

| | |
|---|---|
| **Use Case Title** | UC2 Fault type diagnostic |
| **Level** | System |
| **Use Case Goal** | Identify what kind of fault the unit has and determine how to proceed. |
| **Summary** | The operator gets a faulty unit for repair and a first diagnostic must be perform to determine the type of problem in order to decide what the next step is. This diagnostic is performed as a sequence of checks to perform. |
| **Preconditions** | Use Case 1 success. |
| **Success End Condition** | Operator knows how to proceed. |
| **Failed End Condition** | Operator can't identify the problem with the device. |
| **Primary Actor** | Operator |
| **Trigger** | Operator |
| **Main Success Scenario** | 1. Check fault report.<br>2. Visually inspect the unit. Check mechanics.<br>3. Connect DUT to service station.<br>4. Active DUT power and check LEDs.<br>5. Check current consumption.<br>6. Open CAN communication.<br>7. Check status, product ID, firmware and error counters.<br>8. Check I/O status.<br>9. Reset error counters.<br>10. Add operator comment to service report.<br>11. Disable power.<br>12. Disconnect device from station.<br>13. Mark device with service label. |
| **Extensions** | 4a. If error code -> Lookup error code.<br>5a. Abnormal current consumption -> Disable power.<br>6a. Run use case UC5 Check CAN Communication<br>8a. Run use case UC3 Manually Check I/O<br>8b. Run use case UC4 Automatic Verify I/O |

Add CONSULTING

# Data Models

- A data model explicitly determines the structure of data.
- *Example: LabVIEW Cluster*

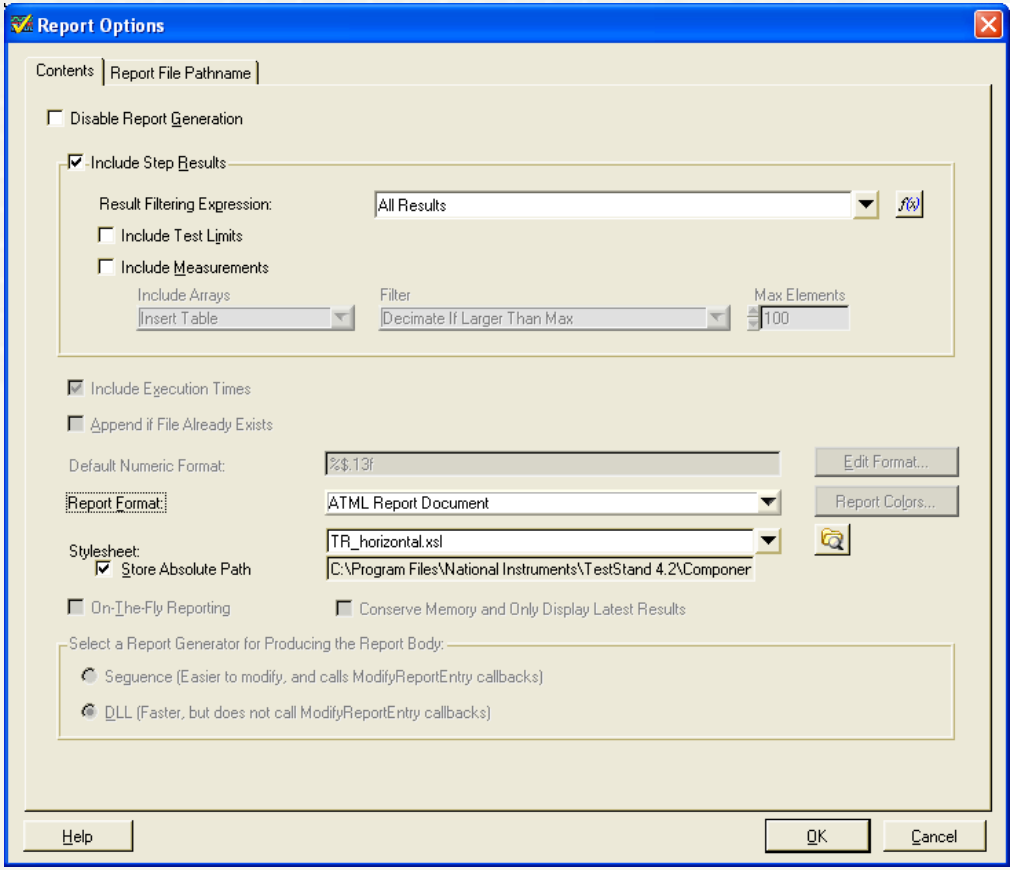



AddQ CONSULTING

# Example: Modeling Test Results

- ATML IEEE Std 1636.1 Test Result Description

```
<TestResults>
    <Personnel>
    <UUT>
    <ResultSet>
        <TestGroup>
            <Test>
                <TestResult>
                    <Outcome>
                <Outcome>
            <TestGroup>
            <SessionAction>
        <Outcome>
    <TestStation>
```



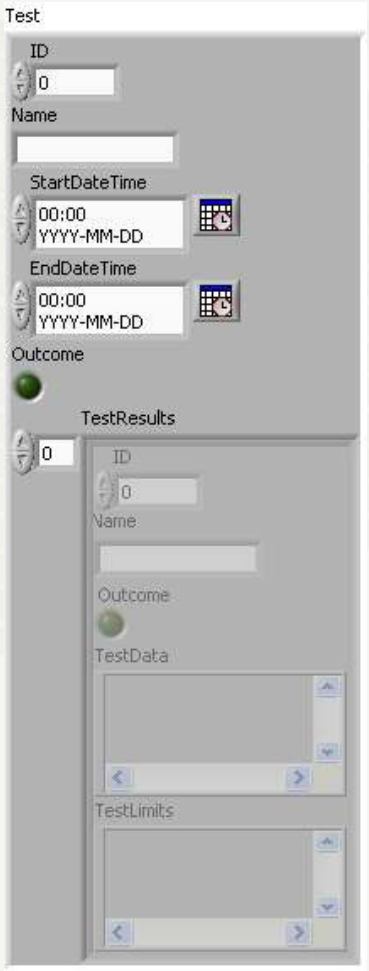
```xml
<?xml version="1.0" encoding="UTF-8"?>
<TestResults xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="SimpleATMLTestResult.xsd">
    <Personnel>
        <SystemOperator ID="administrator" name="administrator"/>
    </Personnel>
    <ResultSet ID="14" name="DemoReport.xml" startDateTime="2011-05-11T14:46:45.109" endDateTime="2011-05-11T14:46:46.542">
        <Outcome value="Failed"/>
        <TestGroup name="Demo.seq#MainSequence" ID="14" startDateTime="2011-05-11T14:46:45.109" endDateTime="2011-05-11T14:46:46.542">
            <Outcome value="Failed"/>
            <SessionAction ID="15" name="Pre-test" startDateTime="2011-05-11T14:46:45.207" endDateTime="2011-05-11T14:46:46.433">
                <ActionOutcome value="Done"/>
            </SessionAction>
            <Test ID="16" name="Battery Voltage Test" startDateTime="2011-05-11T14:46:55.944" endDateTime="2011-05-11T14:46:55.944">
                <Outcome value="Failed"/>
                <TestResult ID="17">
                    <TestData>
                        <Datum value="7.2797690596062" type="c:double" nonStandardUnit="volt"/>
                    </TestData>
                </TestResult>
            </Test>
            <Test ID="18" name="Register Test" startDateTime="2011-01-31T17:43:23.861" endDateTime="2011-01-31T17:43:23.863">
                <Outcome value="Passed"/>
            </Test>
            <Test ID="19" name="Instruction Set Test" startDateTime="2011-01-31T17:43:23.880" endDateTime="2011-01-31T17:43:23.882">
                <Outcome value="Passed"/>
            </Test>
        </TestGroup>
    </ResultSet>
    <TestStation>
        <SerialNumber>
            MERICSSON
        </SerialNumber>
    </TestStation>
    <UUT UutType="hardware">
        <SerialNumber>
            123
        </SerialNumber>
    </UUT>
</TestResults>
```

# DEMO

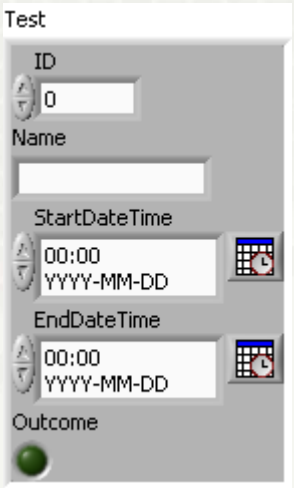
- LabVIEW Data Model
  - Using Clusters

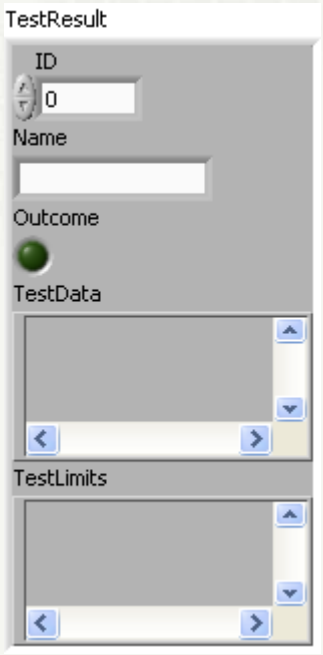
Add CONSULTING

# Cluster vs. Entity Model



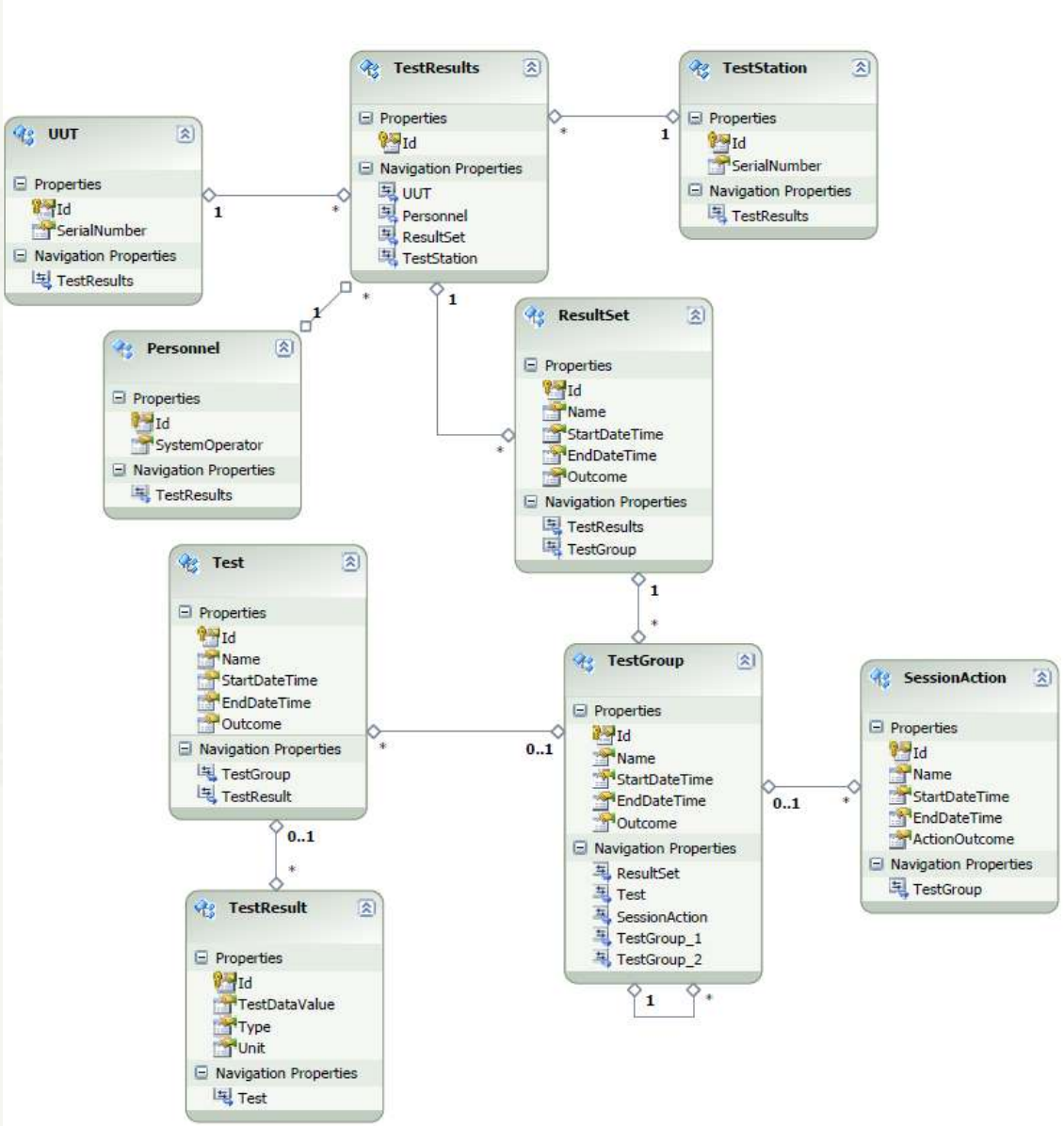

- Split up the clusters to *entities.*




AddQ CONSULTING

# Entity Model




AddQ CONSULTING

# Data Relations

- Cluster -> Data within data (cluster within clusters)
  - + Easy and fast to model
  - - Unflexible, relations are static
  - - Accessing data cause LabVIEW data copies
- Entity -> References between entities
  - + Flexible, any entity could be related to any entity
  - + Easy and good performance accessing data, no data copy
  - - Bit more complex to get it working

# Back to ATML again

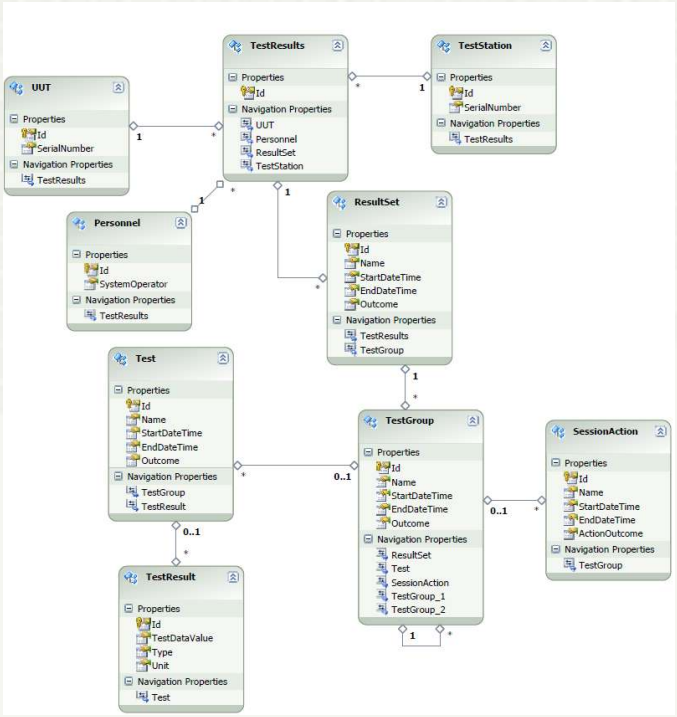
- “Recursive” or “self reference” relationships
- Clusters can’t be used!
- Entities works!

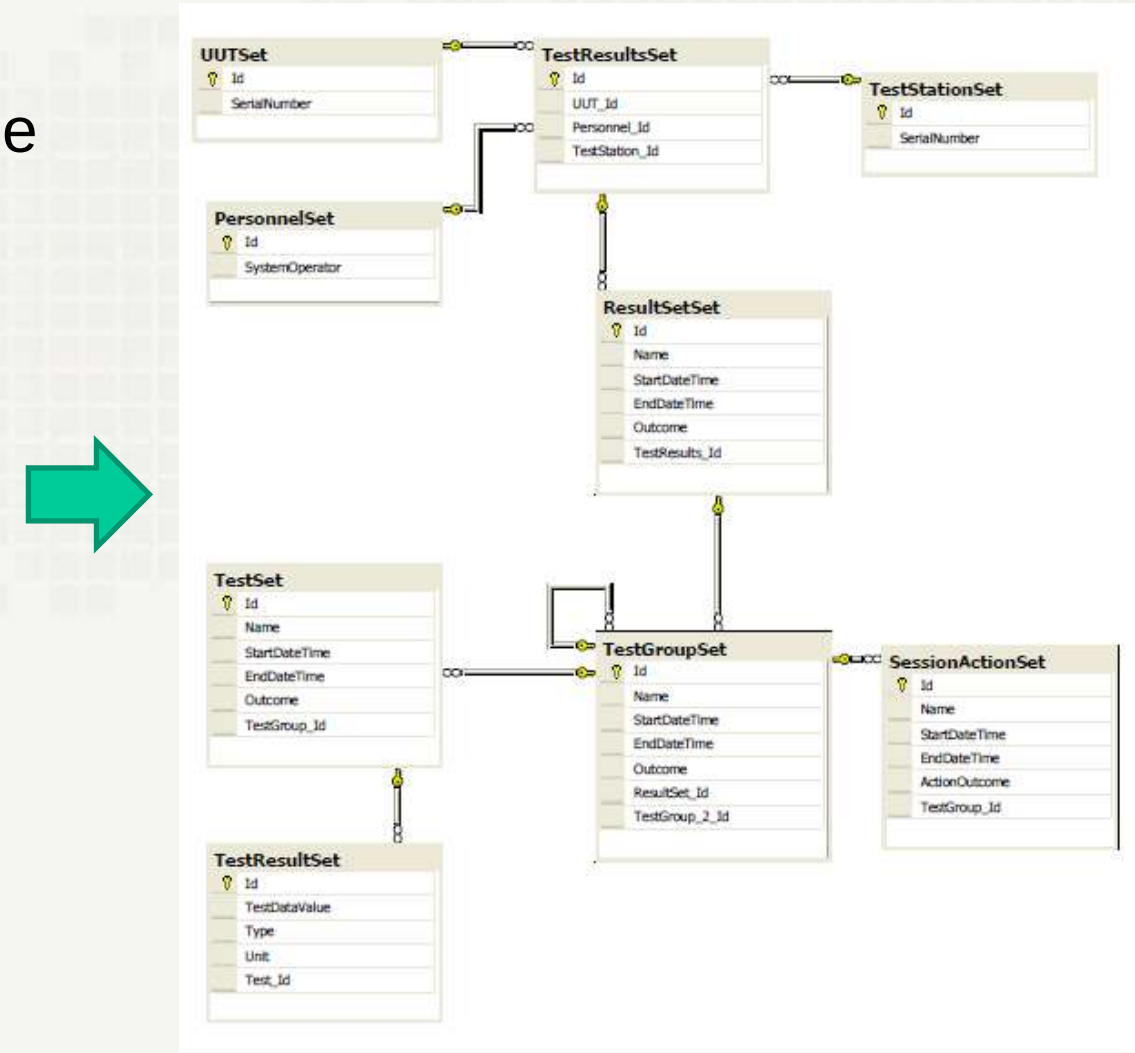
<TestResults>
<Personnel>
<UUT>
<ResultSet>
<TestGroup>
<Test>
<TestResult>
<Outcome>
<Outcome>
<TestGroup>
<SessionAction>
<Outcome>
<TestStation>

AddQ CONSULTING

# Entity vs Data bases

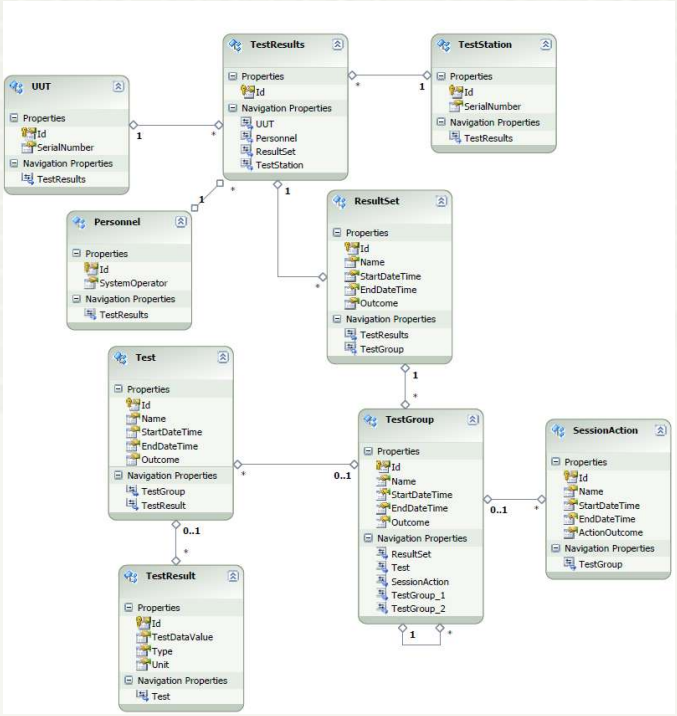
- Easy to map entity classes to data base tables



# Entity vs XML

- Easy to map entity classes to XML elements

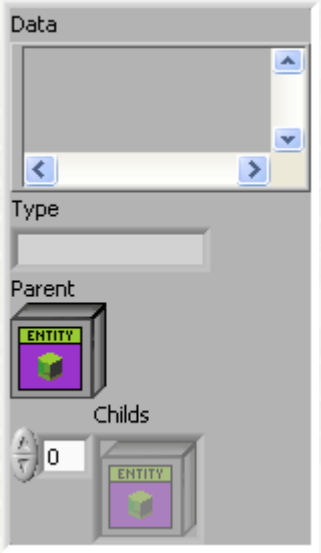


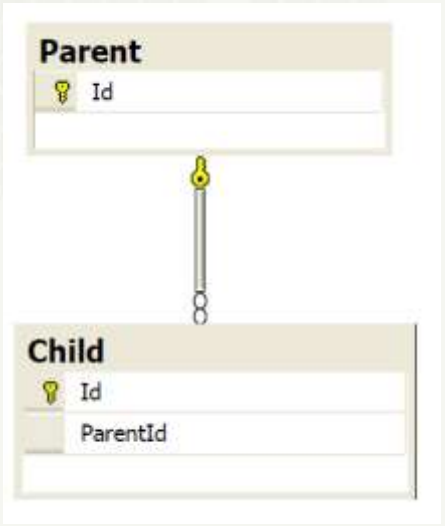

```
<?xml version="1.0" encoding="UTF-8"?>
<TestResults xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="Simpl
    <Personnel>
        <SystemOperator ID="administrator" name="administrator"/>
    </Personnel>
    <ResultSet ID="14" name="DemoReport.xml" startDateTime="2011-05-11T14:46:45.109" endDateTime="2011-
        <Outcome value="Failed"/>
        <TestGroup name="Demo.seq#MainSequence" ID="14" startDateTime="2011-05-11T14:46:45.109" endDate
            <Outcome value="Failed"/>
            <SessionAction ID="15" name="Pre-test" startDateTime="2011-05-11T14:46:45.207" endDateTime=
                <ActionOutcome value="Done"/>
            </SessionAction>
            <Test ID="16" name="Battery Voltage Test" startDateTime="2011-05-11T14:46:55.944" endDateTi
                <Outcome value="Failed"/>
                <TestResult ID="17">
                    <TestData>
                        <Datum value="7.2797690596062" type="c:double" nonStandardUnit="volt"/>
                    </TestData>
                </TestResult>
            </Test>
            <Test ID="18" name="Register Test" startDateTime="2011-01-31T17:43:23.861" endDateTime="201
                <Outcome value="Passed"/>
            </Test>
            <Test ID="19" name="Instruction Set Test" startDateTime="2011-01-31T17:43:23.880" endDateTi
                <Outcome value="Passed"/>
            </Test>
        </TestGroup>
    </ResultSet>
    <TestStation>
        <SerialNumber>
            MERICSSON
        </SerialNumber>
    </TestStation>
    <UUT UutType="hardware">
        <SerialNumber>
            123
        </SerialNumber>
    </UUT>
</TestResults>
```

AddQ CONSULTING

# Relation Comparision

- **Entity Model**
  - Parent <-> Childs
  - Dual way relation
- **Database**
  - Parent <- Child
  - One way relation
- **XML**
  - Parent -> Child
  - One way relation




```
<Parent>
        <Child>
        </Child>
</Parent>
```

AddQ CONSULTING

# DEMO

- LabVIEW Data Model
  - Using Entity Classes

AddQ CONSULTING