Unified Markup Language and Software Design

Introduction
Blueprints are a type of drawing that is based on standard form, layout, scaling, icons and annotations. Blueprints permit complex projects to be described in a visual language that is easy for all the parties involved in a construction project from client to contractor to grasp. They create a coordinate space in which the high-level abstraction of a construction project can be mapped to low-level concrete issues of the construction, such as where to place light switches.

They provide an easy-to-understand basis for reaching an agreement about construction projects, and form the framework for construction contracts. Construction projects are often complex and involve many parties from different backgrounds. The client, for example, is not a single person but a collection of individuals from owners to regulators to neighbors, each of whom has a different set of concerns and expectations, from appearance and cost to impact on neighborhoods and zoning issues. The contractors and trades must understand exactly what they are expected to do in order to project the costs of the project. The architects, engineers and designers create the blueprints to represent the agreements they are planning to make with all parties. We know that people with differing responsibilities can recognize, discuss, change and finally agree upon these matters paying attention primarily to only those graphic elements and annotations that correspond to their interests.

Blueprints and Unified Modeling Language
The Unified Markup Language (UML) is intended by its creators to become a blueprint for a software engineering projects. It is "Unifiied" because in its current form it is a merger of several different software design modeling languages. UML is still evolving. It is likely that it will eventually be incorporated as a diagramming type in blueprints. Blueprints themselves are evolving to become computer simulations of a construction project. You should as a software engineer, understand UML’s strengths and weaknesses and help it evolve so that we can create a common graphical language in which to discuss and recognize best practices.

UML for Software Design
In Software Design, we use only four diagram types:

Class Diagrams
The only name in a class diagram is the name of the corresponding class. Class diagrams describe two specific relationships between classes. These relationships are: IS-A and HAS-A.

A class C has an IS-A relationship to a second class A if it says so in its class declaration. For example, the following class statements describe an IS-A relationship between classes C and B in which C IS-A B.

Please send feedback to alex.morrow@olin.edu
If class D has a HAS-A relationship to class Skittle, this will be reflected in an attribute present in any instance of D. Continuing the example,

```python
class D(C):
    def __init__(self):
        self.skittles = []
    def register(self, skittle):
        self.skittles.append(skittle)

class Skittle(object):
    def __init__(self, dobj):
        dobj.register(self)
```

would set up a HAS-A relationship between objects of class D and objects of class Skittle.

These relationships are shown in class diagrams by the type of arrow used to connect the related classes and the annotations positioned next to the arrow.

**Object Diagrams**

The only names in object diagrams are attributes. An object diagram always names the class from which the object was constructed but putting the class name above it on the right hand side.

Object Diagrams describe an instantaneous state of class instances or objects within a run-time environment. The instantaneous state is the state at some point in the running of the program, either after a series of function or method calls have completed or during a specific set of function or method calls.

**Stack Diagrams**

The only names in stack frames are local variables.

Stack Diagrams describe the state of the methods and functions of an application as they are evaluating function or method call. The consist of a stack of stack frames, each of which is an instance of a function object or method object. A stack frames is an instance of a function objects or method object. When function or method objects are referred to by only one name (usually a global name), this name is used to decorate the stack frame to the left of the frame. The class name for a stack frame is "stack frame."

**Module Diagrams**

The only names in module diagrams are global variables.
The Module Diagram describes the state of global variables. The only global variables users normally encounter in interactive sessions and those in the __main__ module. A module diagram is an object diagram in which the attributes are global variables and the class name is __main__ or the module name. We normally only use the __main__ module in Software Design diagrams, but be aware that every module added to a run-time environment by an include statement (but not by a from x include y statement) has its own global variables.

Class Diagram Examples

- **Class Diagrams (IS-A)**

  **Inheritance**
  
  Class names (usually global)

  ```python
  class Deck(object):
      ...
  
  class Hand(Deck):
      ...
  ```
Inheritance

Class names (usually global)

class Deck(object):
    

class Hand(Deck):
    

class RummyHand(Hand):

RummyHand.__mro__
{
    <class '__main__.RummyHand'>,
    <class '__main__.Hand'>,
    <class '__main__.Deck'>,
    <type 'object'>
}

ClassName.__mro__ gives the method resolution order
contains (references to)

class Deck(object):
    def __init__(self):
        cards = []
    def register(self, card):
        cards.append(card)

Note that the Deck __init__ and register methods are inherited by Hand.

HAS-A relations may go in both directions (Turtles do this in TurtleWorld).

properties

contains (references to)

class Deck(object):
    def __init__(self):
        cards = []
    def register(self, card):
        cards.append(card)

class Hand(Deck):
    def count_points():
        ...

Note that the Hand inherits the Deck __init__ and register methods. Therefore, if it does not override these methods, a Hand instance will also have a "HAS-A multiplicity of" relationship with Card instances. In an object diagram, the Hand instance would have an attribute card.

• Class Diagrams (HAS-A)

• Class Diagrams (Combined)
Python Object Diagrams

Object Diagram Examples

Stack Diagram Examples

Stack Diagram Frames

Local Names

frame

NewtonsMethod

i3 33

f  lambda q:q **.5

nm1

A frame is a type of object
It is an instance of a function or method body
Function bodies are usually pointed to by
global names, but can be pointed to by local name
Functions themself do not have names. They are referred to by names.
You can think of function bodies in lambda form:
def foo(x): return x**.5 is equivalent to the  f  lambda q: q**.5
Module Diagram Example

**Global Names**

- Module dictionary

---

**Module Diagram**

```python
__main__
```

- `a` → ‘abc’
- `xx` → 1.4535e3