UML:
the **Unified Modeling** or
the **Unstructured Muddling**
Language?

Background, status,
opinions, pros & cons
for COMP 477
by Conrad Weisert (www.idinews.com)
Spring, 2015

---

Disclaimer

- Some of the concepts presented here are fundamental to a course in **Systems Analysis** (COMP 320 is one)
  - We’ve focused here on the concepts and left out many of the details

- Project managers don’t need to master them, but will often need:
  - to know what they’re about
  - to appreciate their strengths and weaknesses,
  - to settle arguments between team members who favor one approach and team members favor another (in particular UML versus SA).

---

**Why is the UML so important today?**

- Wide acceptance among I.S. professionals as preferred set of tools for "analysis and design".
  
  **What's "analysis & design"?**

- Publicized, documented, and promoted by leading gurus, including:
  - Grady Booch
  - James Rumbaugh
  - Ivar Jacobson

- Blessed as a **standard** by the OMG
  (for "modeling")

  **What's that?**

---

**Therefore**

- Any software-development organization (or individual professional) has to recognize the UML in one of these ways:
  - by embracing it as the main components of its systems analysis methodology, or
  - by rejecting it and adopting a practical alternative, or
  - by selectively integrating the UML’s best features into an older mainstream methodology, such as structured analysis (SA).

- Professionals cannot **ignore** the UML.
**1990's: the object revolution**
Object-oriented analysis (OOA) and Object-oriented design (OOD)
- Object-oriented technology (OOT) originated with programming (OOP)
- Supported by programming languages, including:
  - Smalltalk (Goldberg, Park Place, DigiTalk)
  - C++ (Stroustrup, AT&T)
  - Java (Sun)
- Widely viewed as very successful -- now the mainstream of software development

**Characteristics of Object-oriented programming (OOP)**
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

That sounds good. Can we extend that concept to systems analysis (OOA)?
Let's try!

**OOP definitions**
- Abstraction
  [http://en.wikipedia.org/wiki/Abstraction_(computer_science)]
- Encapsulation
  [http://en.wikipedia.org/wiki/Encapsulation_(object-oriented_programming)]
- Inheritance
- Polymorphism
  [http://en.wikipedia.org/wiki/Polymorphism_(computer_science)]

*Not needed for COMP 477, but project managers should know what OOA is all about and appreciate why it's helpful.*

**Early (pre-UML) OOA work (textbooks)**
- Coad & Yourdon: *Object-Oriented Analysis*
- Booch: *Object-Oriented Analysis and Design*
- Rumbaugh: *Object-Oriented Modeling and Design*
- Jacobson: *The Object Advantage*
- others
  - Incompatible terminology and diagramming standards among those!
Important: There's no such discipline as "analysis & design"

- Systems Analysis and System Design
  - are done at different times during a project.
  - demand very different skills.
- Analysis describes the external view of a proposed new system (the WHAT)
- Design describes the internal view of a system (the HOW)

What about all those books?
Which phases of our sample life cycle are they usually done in?

The "three amigos"

- Booch, Rumbaugh, & Jacobson
  - were competitors
  - each promoted his own version of OOA.
- But they all ended up working for Rational Software and now agree on the UML.
- Rational Software sells the dominant C.A.S.E. tools for object-oriented modeling, including Rational Rose.
- IBM has bought Rational Software.

UML has been immune from most criticism

- "It's not a methodology, only a language."
  - Therefore UML promoters reject criticisms on methodology grounds.
- "It's not for system specification, but for system modeling."
  - Therefore UML promoters reject criticisms on understandability grounds.
  - What's "modeling"?
    - Who needs to understand it?

Is the UML a methodology component

- Of course.
- When UML gurus claim it's not, they mean only that it's not a standard process or system development life-cycle.
- Actually it is closely tied to its own standard life cycle!
The UML's Life Cycle

- Since UML is "process-independent", UML gurus claim it's OK to follow any SDLC you like, as long as that life cycle is:
  - use-case driven
  - architecture centric
  - iterative and incremental

How do we estimate an iterative project?

- By the way here's one ("Objectory"© -- Jacobson):
  1. inception phase
  2. elaboration phase
  3. construction phase
  4. transition phase

Objectory (UP) phases

- **Inception** phase:
  - Like traditional "project initiation" + "business requirements"
  - Defines:
    - project scope
    - initial justification

- **Elaboration** phase:
  - Like traditional "functional specifications" or "external design"
  - "During the elaboration phase you need to get a good handle on the requirements and their relative priorities." - Fowler p. 17

Objectory phases (continued)

- **Elaboration** phase (another view):
  - "... usually uses one iteration cycle and involves the following activities:
    - Elaborate the specification of the effort from the previous phase.
    - Baseline and completely delimit scope, objectives, and requirements.
    - Baseline the business case by solidifying the vision, solidifying success criteria, and mitigating the highest risks. Baseline the plan with a schedule.
    - Distribute the requirements among multiple iteration cycles within the construction phase.
    - more . . .

Objectory phases (continued)

- **Elaboration** phase (continued):
  - Focus on understanding or forming a notion of the problem to determine the requirements that the problem imposes on its solution . . ., establishing and verifying the foundation for the overall solution (architectural design) and distributing the requirements among the iteration cycles of the construction development phase.
    - Alhir, p. 23
Clearing it all up

"During the elaboration phase, as we have already noted, we build the architecture. We identify the use cases that have a significant impact on the architecture. We realize these use cases as collaborations. It is in this way that we identify most of the subsystems and interfaces -- at least the ones that are architecturally interesting. Once most of the subsystems and interfaces are identified, we flesh them out, that is, write the code that implements them. Some of this work is done before we release the architectural baseline and it continues throughout all of the workflows."

- Ivar Jacobson

Objectory phases (continued)

**Construction** phase (Fowler p. 15):

- "... consists of many iterations, in which each iteration builds production quality software ... that satisfies a subset of the requirements ... Each iteration contains all the usual life-cycle phases of analysis, design, implementation, and testing. In principle you can start at the beginning: Pick some functionality and build it, pick some other functionality, and so forth. However it is worthwhile to spend some time planning."

Wait! There's more.

Objectory phases (continued)

**Construction** phase (UML UG p. 34):

- "... the third phase of the process, when the software is brought from an executable architectural baseline to being ready to be transitioned to the user community. Here also, the system's requirements and especially its evaluation criteria are constantly reexamined against the business needs of the project, and resources are allocated ... to actively attack risks to the project."

Which version do we like better?
**Objectory phases (concluded)**

- Summary:
  - A little of this and a little of that in each phase.
  - Emphasis on system **architecture** from the start poses serious problems:
    - obscures the **business** problem or opportunity
    - blurs distinction between analysis and design
  - Descriptions by gurus permeated by **vagueness** and lack of rigor ("get a handle on", "form a notion of", etc.)

**A common policy for application system development in organizations**

- *We will develop custom software only when it is shown that no suitable packaged application software product exists.*

- Not a variant, but the mainstream in many, probably most, organizations today
- This policy is foreclosed when requirements "emerge" by iteration.

**Five views of a system**

![Diagram of five views of a system]

- **Design view**
- **Implementation view**
- **Process view**
- **Deployment view**
- **Use-case view**

From UML Users' Guide, p. 31

*What ties them all together?*
*Who makes sure they're consistent?*

**Modeling different views**

"When you model a system from different views, you are in effect constructing your system simultaneously from multiple dimensions. . . . If you do a poor job of choosing these views or if you focus on one view and the expense of all others, you run the risk of hiding issues and deferring problems that will eventually destroy any chance of success." — UML UG p. 98

*How do you decide?*
A central component

**use-cases**

- Time-sequenced narrative or diagram of *what happens* ("course of events") when a user initiates some action
- Widely associated with object-oriented analysis (OOA), mainly through the efforts of Ivar Jacobson.
- But there's nothing at all **object oriented** about them! You can practice:
  - OOA without use-case scenarios
  - use-case scenarios without OOA
- "I can't imagine a situation now in which I would not use use-cases.” - Fowler, p. 51

**use-case role in a system specification**

- The glue that ought to tie everything else together.
- Compare with De Marco's data-flow diagrams
- A large system may have hundreds of use-cases (Jacobson says 15 is typical)
  - How do we know when it's complete?
  - How do we know they're mutually consistent?

**Documenting a use-case**

- Narrative description !
  - or
- Diagram

**use-case narratives**

- "Use cases are best described using simple language to facilitate understanding. They are described in episodic, narrative form." - Jacobson, p. 178

- Compare with De Marco's "Victorian novel" approach to system specification.
Use-case diagram

Public library system

What's missing here?

Use-case documentation

Typically, you'll first describe the flow of events for a use case in text. As you refine your understanding of your system's requirements, however, you'll want to also use interaction diagrams to specify these flows graphically.

- UML UG p. 224

Lots more kinds of UML diagram

- Class diagrams
- Object diagrams
- Package diagrams
- Activity diagrams
- Interaction diagrams
  - sequence diagrams
  - collaboration diagrams
- State-transition diagrams
- Deployment diagrams

More have been added to recent versions!

Class diagrams

- "Static design view" of a system
  - One point of similarity between UML and other versions of OOA (e.g. Coad & Yourdon)
- For each type of data item, specifies:
  - Its attributes (components / "has-a" hierarchy)
  - Its behavior (methods / interface)
- For related classes:
  - Inheritance ("is-a" hierarchy)
  - Entity-relationships

Logical database design
Some UML issues

- Always assumes custom architecture and in-house software development; little provision for evaluating, selecting, and integrating application program packages.
- There's no clear starting point:
  - for creating
  - for reading
- There's no definite way of knowing when the model is complete.
- Relies on the system modeler to keep multiple independent diagrams and other documents consistent with one another!

How does anyone really know?

2015 conclusions

Conrad Weisert

- Based on what I've discovered so far, UML's weaknesses outweigh its strengths.
- In examining actual UML projects in organizations
- I've so far found two kinds:
  - the utterly trivial
  - the utterly atrocious
- I've found no example yet where user representatives claim that they understood exactly what they were going to get before they approved the project!
- But people keep telling us that it works great!

So what should we do about it?

The Requirements Crisis

- Large projects that try to follow UML / UP often experience a serious deficiency in gathering, organizing, understanding, and approving the users' requirements.
  - Abandonment of structure, in particular:
    - Where do we begin?
    - How do we know when we're done?
  - Overwhelming detail
    - Compare with DeMarco's "Victorian Novel" approach to system specification

How did we come to abandon requirements documentation structure?

A chronology from ca. 1991:

- Object-oriented analysis (OOA) is good.
- UML (Booch-Rumbaugh) is standard for OOA.
- Sponsoring users and other non-technical audience can't understand UML reqs. specifications.
- Jacobson adds use-cases to UML.
- Users can't understand use-cases either.
- Unstructured "requirements lists" substituted.

What are those?

- Above all, reject / ignore anything associated with "traditional" or "structured" systems analysis (SA) ("Dinosaur" methodologies)
The fallback: Unstructured requirements lists

- When it became more and more clear that UML documentation couldn't be understood by sponsoring user representatives, projects needed an alternative way to communicate what the proposed new application system was going to do.

- Solution: Requirements lists
  - "The system shall . . . 
  - (Dozens or hundreds of those)
  - Sounds reasonable to the sponsors. Have we seen that before?

But whatever you do, don't

- Start developing (or commit to purchasing) software (phases 4 & 5 in our sample life cycle) until you're absolutely certain that the sponsoring users:
  - understand exactly what they're going to get,
  - know and approve credible estimates of when they're going to get it and what it will cost.

- Be intimidated by claims about "modern" or "up to date" approaches that don't clearly support the above.

A reasonable 2014 methodology for documenting the results of systems analysis

- Continue using Structured Systems Analysis (e.g. per De Marco, 1978 with sensible updates) as the framework.

- Substitute O.O. class diagrams, for some data dictionary specs
  - encapsulated behavior
  - where appropriate, exploit inheritance. What else?

Sources

- Not a recommended bibliography, but just sources of quotes for this presentation from vigorous UML advocates:
  - Ivar Jacobson: The Object Advantage, AW, 1994, 330 pages
  - Si Alhir: UML in a Nutshell, O'Reilly, 1998, (275 pages)
Major Dramatic Breakthroughs

- Every 3 or 4 years someone devises a major dramatic breakthrough (MDB) in software development.
- Some MDBs are evolutionary, others revolutionary.
- Each MDB claimed productivity gain between 1.5x and 10x.

Therefore . . .

MDB cumulative impact

- Therefore (conservatively), we are now developing programs with less than 1/1000 the effort required in 1960.
- Or
- We're now routinely developing programs 3 orders of magnitude bigger and more complex.

Why didn't that happen?

- Half-hearted adoption
- Guruism
- Obfuscation and intimidation by insiders.
- Overhyped fad MDBs -- Some may even be harmful! Which ones?