UML:

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

Background, status, opinions, pros & cons

for **APCU**

by Conrad Weisert (**www.idinews.com**)

28 May, 2005

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

  Who are they?

- 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:
  - 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).
- We cannot *ignore* the UML.

Agenda

- Background
  - the role of systems analysis
  - the object-oriented revolution
- UML
  - origins
  - characteristics
- This will not be a UML tutorial

The need for systems analysis

- In the early days of computing, there were only **programmers** and **users**.
- That worked out well for simple applications, but broke down for complicated ones.

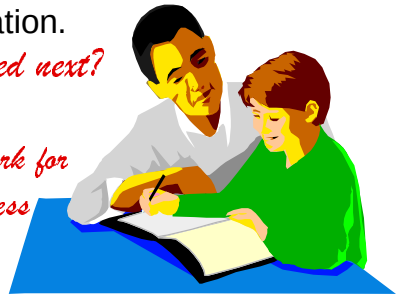
"Simple" and "complicated" in what sense?

1950s origins

- A **programmer** (technically competent problem solver) would confer with a **problem sponsor** (user with funds and authority) on requirements for a new or modified application.

What happened next?

Why didn't that work for complicated business applications?



1960s crises

- Growing demand from business led to
  - larger project teams with > 4 programmers
  - hordes of poorly trained underqualified programmers.
  - major project fiascos
- Desperate organizations "promoted" their most mature and knowledgeable programmers to **systems analysts**.

What did the systems analysts do?

1960s crises (continued)

- The typical 1960's systems analyst:
  - Conferred with the problem sponsor
  - Prepared detailed **flow-charts** and **narratives** (DeMarco's *Victorian Novel* approach) of the application logic
  - Drew up **layouts** for reports and files
  - Assigned coding and testing tasks to the programmers.

Why didn't that approach work?

The 1970's
The *Structured Revolution*

- Systematic approaches to programming began to overcome the lack of design, coding, & testing skills.
- Projects continued to fail, not because of programming problems but often because the programmers were solving the wrong problems!

What was the solution?

-

1980's Maturing of Systems Analysis

- Structured Analysis
  (DeMarco, Gane & Sarson, et al):
  - Clearly separated analysis from design
  - Replaced long-winded narratives by coherent set of interrelated documentation with graphical emphasis
  - Empowered the sponsoring end users to understand external specifications in full detail

The role of the systems analyst

- To capture the sponsoring users' requirements for a new or modified application system and document them in a form that can be fully understood by two audiences:
  - The sponsoring end users, who must approve it
  - The developers who will build it.

The phased system developmentlife cycle (SDLC)

- Phase-limited commitment to prevent loss of control
- Concrete phase deliverables to assure proper foundation for later work
- "Moving window" estimating
-
-

Here's a typical one -- 7 phases. . .

A Typical SDLC

1. Project initiation
2. Business system requirements
3. External system specifications
4. System architecture
5. Construction
6. Installation
7. Project review

Phase 1. Project Initiation

- Prospective **users** and **developers** agree that:
  - some problem might be solved
    or
  - some opportunity might be met using information technology.
- They establish a project, which then determines the problem *scope*.

Phase 2. Business System Requirements

- **Systems analysts** work with the prospective **users**:
  - to specify what business capabilities a new system must support
    and
  - to assess the feasibility of developing (or buying) such a system

This phase is poorly understood and rarely performed.

45

Phase 3. External System Specifications

- **Systems analysts** specify exactly what the proposed new system will do in terms that both the **users** and the **developers** can understand.
- They may also recommend (or specify constraints on) specific equipment, software, development tools, or design approaches.

What are some other names for this phase?

The external system specs.

- Authors:
  - One or more **systems analysts**, often with assistance and participation of user representatives, specialist consultants, etc.
- Audiences
  - The sponsoring **users**
  - The **developers**

Roles, not positions

- The ESS, then, serves as a *contract* between the users and the developers through the remaining phases.

Does that mean that the specifications are frozen at the end of phase 3?

The critical point

- The end of phase 3 ("External System Specification", or . .) is **the most important point** in a project's life cycle.
- It is the last point:
  - at which changes can be made without huge cost.
  - where the sponsoring end users can be expected to understand the deliverables in full detail
  - that is (or should be) independent of.
    - choice of operating platform(s)
    - make or buy choice
    - development tools and methodologies
    - etc.

Weisert's Rule # 2 on project failure:

- Inadequate **external design** is one of the two most common reasons why projects fail; i.e. the content is one or more of these:
  - incomplete,
  - erroneous, inconsistent,
  - incomprehensible to the intended audiences,
  - mixed up with *internal* design.

Especially this

Question

- Is is desirable (acceptable) for the systems analyst to prepare two seaparate versions of the phase 3 deliverables?
  - one in business-oriented language for the sponsoring end users
  - the other in technical language for the developers

Phase 4. Computer System Architecture

- **Technical specialists** establish a structure for a computer-based system to implement the ESD.
- In parallel **systems analysts** and **users** begin preparing plans for testing, training, and installation
- *What are some other names for this phase?*

Phase 5. Construction

- **Technical specialists** build the system according to the overall architecure.
  - For **in-house development**, they:
    - design, code, and test programs
    - establish data bases
  - For **purchased components**, they:
    - order, test, and install software products
  - For **all application systems** they:
    - prepare operational documentation,
    - package the system for routine operation, and
    - conduct integrated system testing.

What are some other names for this phase?

Phase 6. Installation

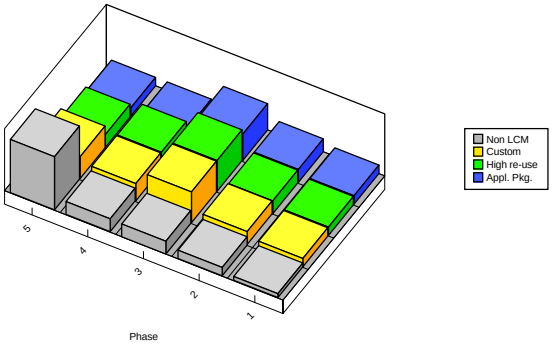
- **Systems analysts**, **programmers**, and **users** cooperate to bring the already tested and documented components into operational ("production") status.
- Their activities include:
  - **Training**, user personnel
  - **Acceptance testing**
  - **Conversion** (or "start up") to initialize the live database and lead into routine operation.

What are some other names for this phase?

Phase 7. Project review

- The I.S. organization assesses the project, both improve its own future effectiveness and to propose improvements to the newly-installed system. They evaluate:
  - How well the new system is meeting the user organization's needs
  - the operational cost, performance, and reliability of the system.
  - the cost and duration of the project.
- This phase is often omitted. *Why?*

Typical phase cost ratios



1990's: the object revolution

- Object-oriented technology (OOT) originated with *programming* (OOP) and design (OOD)
- Supported by *languages*:
  - 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 the concept to systems analysis (OOA)?

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

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 **What**
- *Design* describes the internal view of a system **How**

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

30

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"?

Is the UML a methodology component

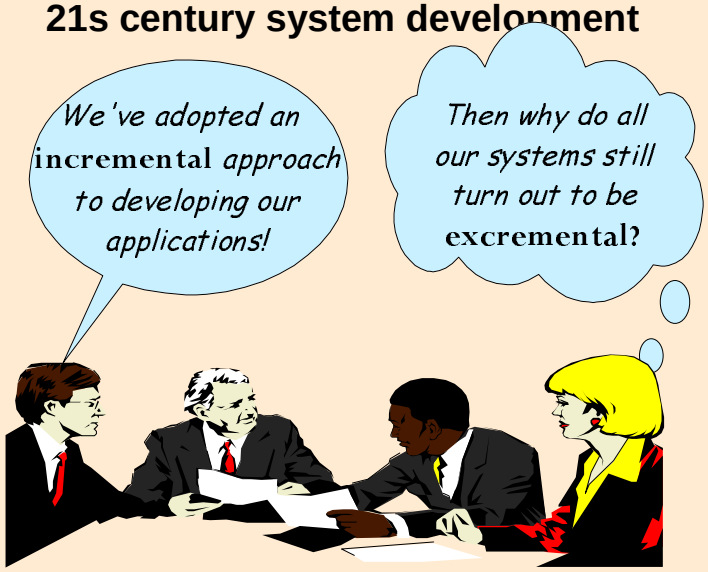
- Of course.
- When UML gurus claim it's not, they mean only that it's not a standard *process* or system development ***life-cycle***.
- Actually it is closely tied to its own standard life cycle!
  -

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
- By the way here's one ("Objectory"© -- Jacobson):
  - **inception** phase
  - **elaboration** phase
  - **construction** phase
  - **transition** phase

Rational has renamed it.

21s century system development



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

Is this where we do most of the systems analysis?

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 reuirments that the problem imposes on its solution . . ., establishing and verifying the foundation for the overll 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 evalutation 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 (continued)

- **Transition** phase:
  - Like the traditional "installation phase" + ongoing maintenance + more "iterations".

Objectory phases (concluded)

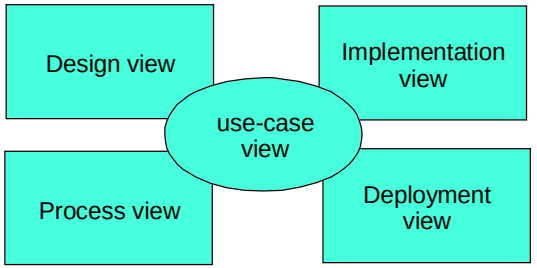
- Summary:
  - A little of this and a little of that in each phase.
  - Emphasis on system ***architecture*** from the start poses serious problems:
    - obscures the ***business*** problem or opportunity
    - blurs distinction between analysis and design
  - Descriptions by gurus permeated by **vagueness** 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 software product exists.*

- Not a variant, but the mainstream in many, probably most, organizations in 2005
- This policy is foreclosed when requirements emerge by iteration.

Five views of a system



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, your 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?

use-case

- 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 consistent?*

15

Documenting a use-case

- Narrative description !

  or

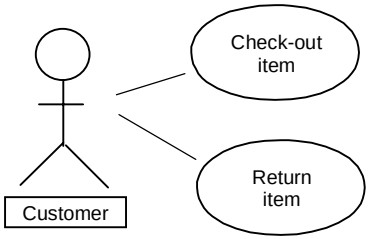
- Diagram

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
(continued)

- *"Typically, you'll first describe the flow of events for a use case in text. As you refine your undersanding 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

What's the problem with that?

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 (?)

- 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

2005 conclusions

Conrad Weisert

- Based on what I've discovered so far, UML's weaknesses far outweigh its strengths.
- In examining actual UML projects in organizations I've so far found two kinds:
  - the utterly trivial
  - the utterly atrocious
- But people keep telling me 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 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.
- Reject / ignore anything associated with "traditional" or "structured" systems analysis (SA)

The Waterfall Approach

- There's no such thing.
- The term was coined as a pejorative by people trying to discredit the traditional SDLC.
- It implies extreme inflexibility:
  - Once the deliverables from phase N are approved, you can't return to phase N or an earlier phase!
  - Until the deliverables from phase N are approved, you can't start on phase N+1!

A recommended 2005 methodology for documenting the results of systems analysis

- Continue using **Structured Systems Analysis** (e.g. per De Marco, 1978) as the framework.
- Substitute O.O. **class diagrams**, for data dictionary
  - encapsulated behavior
  - where appropriate, inheritance.

- Continue following a disciplined SDLC.

Sources

- Not a recommended bibliography, but just sources of quotes for this presentation from vigorous UML advocates:
  - Ivar Jabobson: ***The Object Advantage***, AW, 1994, 330 pages
  - Martin Fowler & Kendall Scott: ***UML Distilled***, AW, 1997, 180 pages
  - Grady Booch et al: ***The Unified Modeling Language User Guide***, AW, 1999, 475 pages
  - Si Alhir: ***UML in a Nutshell***, O'Reilly, 1998, (275 pages)