

The Process
of
Building LANDIS-II

Jimm Domingo
University of Wisconsin-Madison
July 13, 2004

Software Development Before LANDIS-II

- Tools and Process – LANDIS 4.0
 - Unified Modeling Language (UML)
Tutorial from Trireme International
 - Unified Process Model (UPM)

I'd like to start with the software-development process prior to LANDIS-II.

Bob Cumming introduced a couple tools and practices from the field of software engineering.

First, there was the industry-standard notation called Unified Modeling Language which he introduced using a tutorial from Trireme International.

Second was the methodology referred to as the Unified Process Model in the overview documentation for LANDIS 4.0.

I considered Bob's efforts to improve how the LANDIS community developed software a major accomplishment of his time on the project.

I'd like to build on those efforts today, by starting with the UML tutorial and a few observations about it.

It is clearly aimed at developers.

There's a wealth of information packed into those 120 pages.

I must confess I've only read about 2/3's of it myself so far.

If you've finished it, I commend you for your perseverance.

Given its target audience and its length, I wouldn't recommend the tutorial for new people who join the project who aren't developers.

There are better means that are shorter & more concise that aim at users & customers involved in a software-development process.

The tutorial does have a good synopsis of the origin of UML within the context of the unification efforts that happened in our field in the 1990s.

Unification of Notation in Software Methodologies

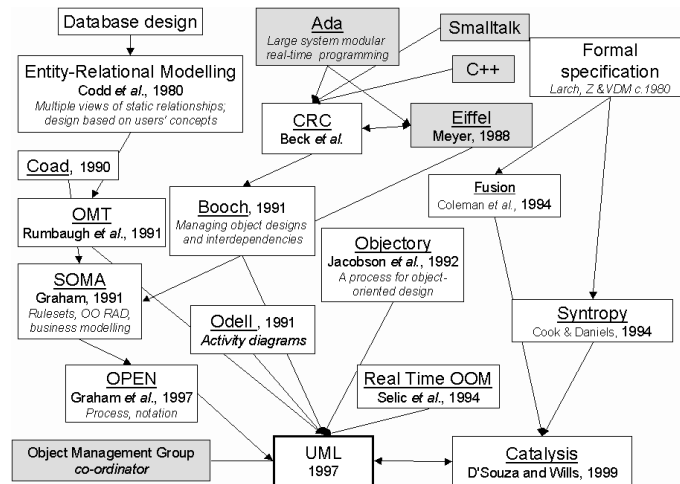


Figure 1, UML – a tutorial, ©1999 Trireme International

This figure is first one in the tutorial.

It illustrates one major aspect of that unification effort – to standardize the notation used by various methodologies.

Different methodologies used different notations, which was really challenging because a symbol would have different meanings in different notations.

This challenge exemplifies the difficulties with having so many methodologies.

This fragmentation in the field and the associated conflict was why this period, late 80's, early 90's, was known as the "Method Wars".

This figure illustrates some of the methodologies & programming languages that influenced of what would become the standard notation known as UML

Figure is by no means exhaustive, but it conveys the sense of how crowded the methodology field was.

There were early attempts at unification (e.g Fusion in the middle of the figure on the right side), but the effort that would succeed was lead by 3 prominent methodologies who became known as the 3 amigos.

UML Designed by The 3 Amigos

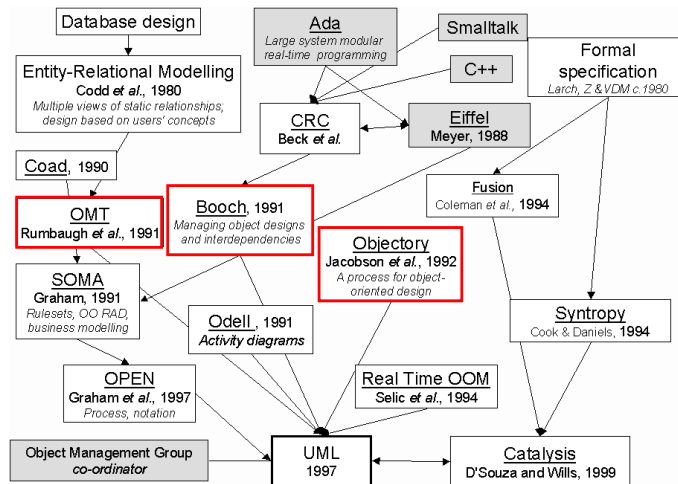


Figure 1. UML – a tutorial, ©1999 Trireme International

The 3 Amigos and their methodologies are shown in red.

The 1st amigo, on the right, is Jim Rumbaugh (pronounced ‘Rumbow’) at GE who invented the Object Modeling Technique (OMT).

The 2nd amigo is Grady Booch (pronounced ‘Bootch’) at Rational whose methodology was simply known as by his name.

If you had a chance to read my statement about my approach to software development, this is the methodology I referred to.

It’s the one I’m familiar with from my previous position at OSU.

The 3rd amigo is Ivar Jacobson who created the methodology originally called Object-Oriented Software Engineering (OOSE) while working at Ericsson, the Swedish telecom giant.

Jacobson started his own company called Objectory in 1987, and continued to refine his methodology which became known by his company’s name.

The unification effort really took off when these 3 ended up under one roof at Rational.

Booch had been with Rational since it was founded in 1981.

In 1994, Rambaugh was hired by Rational.

And a year later, in 1995, Jacobson joined them when Rational bought his company, Objectory.

Their initial efforts focused on unifying their notation.

The next year, 1996, they issued a draft for public comment of their merged notation which they called UML.

It was approved as an industry-standard by the Object Management Group the next year, in 1997.

OMG is a member-based association that promotes the standardization with object software to improve interoperability.

OMG uses a public process in developing its standards or specifications.

As the figure shows, a lot of groups & people influenced the UML specification.

But the 3 amigos are considered the original designers of UML.

Types of Models

- Software Models
 - Models of the software system which developers use to build the system
- Ecological Model
 - Model of forest landscape and ecological processes, which the software must implement

Why is the notation called Unified Modeling Language?

It's used to represent the models produced by the methodologies.

I should take a moment to point out that, in the context of the LANDIS-II project, we're using the word "model" to refer two types of models.

There are software models, which are models of the software system that we're building.

These models are used by developers to build the system.

Then there are ecological models, in particular, LANDIS, which is a model of the forest landscape and associated ecological processes.

The software system we're building must implement this ecological model so researchers can use the model in their research.

Unified Process

1997 – UML specification

1998 – Rational Unified Process 5.0 (RUP)

1999 – *The Unified Software Development Process*

Unified Process (UP)

– OMG's RFP for a Metamodel of Software Engineering Process

2000 – Unified Process Model (UPM) proposal

2002 – Software Process Engineering Metamodel (SPEM) specification

So, in 1997, UML is approved as a standard.

Meanwhile, the 3 amigos continue their unification efforts, and merge their methodologies.

The combined methodology becomes the basis for the next major version of software that Rational produces to help developers.

The new version is released a year later, in 1998, and the software has a new name: the Rational Unified Process.

The 3 amigos also want to advance the field of software engineering by putting the basic concepts of their process out in the public.

So one year later, in 1999, they published a book called *The Unified Software Development Process* which describes the generic process embodied in the commercial product RUP.

They call this generic, non-commercialized, version of their process, the Unified Process, which they refer to with the acronym (UP).

That same year, the industry standards association, OMG, issues an RFP for a specification of a metamodel of the processes used to engineer software.

The next year, 2000, Rational & several other companies work together to submit a proposal in response to the RFP.

Their proposed metamodel specification is called the Unified Process Model (UPM).

But the acronym UPM is short-lived, because by the time their proposed specification is adopted formally two years later in 2002, the name of the metamodel has changed drastically, and is now called the Software Process Engineering Metamodel (SPEM).

Levels of Viewing Software Development Processes

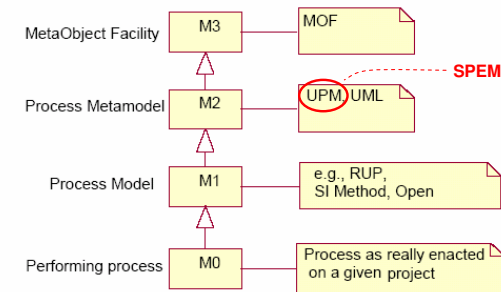


Figure 1-1 Levels of modeling

OMG Specification *Software Process Engineering Metamodel v1*

This figure is from the metamodel specification.

It shows the different levels of how to view (model) the processes used to develop software.

It has one addition show in red on the right.

Note that it still uses the acronym UPM.

This same figure appears in the draft proposal submitted in 2000 that was titled *Unified Process Model*.

Apparently, although the final specification was renamed and had replaced each occurrence of the UPM acronym in the text with SPEM, the figure was not changed.

So I've marked it to indicate that UPM is not in use, and the metamodel goes by SPEM.

The metamodel, as the name implies, is a higher-order abstraction for the common aspects of various models of soft-development processes.

It might be easier to think of process models (level M1 in the figure) as "process frameworks", e.g., types of software process.

Examples of these frameworks are RUP/UP, IBM's System Integration Method (SI), Open Process Framework (OPF).

A process framework provides the basis for customizing a process for a specific project.

Obviously, software projects come in different sizes and scales, ranging from a small utility program like the calculator accessory on Windows to a large, complex system like the air traffic control system for Canada.

So certain aspects of the process framework are selected & tailored based on the needs of the specific project at hand.

What I want to do next is describe an overview of the framework known as the Unified Process.

Unified Process

- Use-Case Driven
- Architecture-Centric
- Iterative and Incremental

These are the fundamental tenets of the Unified Process.

First, it is use-case driven.

You are already familiar with use cases, and so you know that a use case is a sequence of actions, performed by **actors** (people or other systems) and the system itself that produces a valuable result for the actors.

Understanding how these actors will interact and use a software system is important to designing and building that system.

Second, the Unified Process is architecture-centric.

Architecture is the fundamental organization of the system as a whole.

A well-designed architecture helps maintain a unifying coherent structure for the system.

Without such an architecture, it's easy to add parts onto a system throughout its lifetime without maintaining the system's integrity.

This eventually leads to a system that is difficult to maintain and extend.

A well-designed architecture facilitates reuse.

It identifies common functionality shared among components, avoiding redundancy.

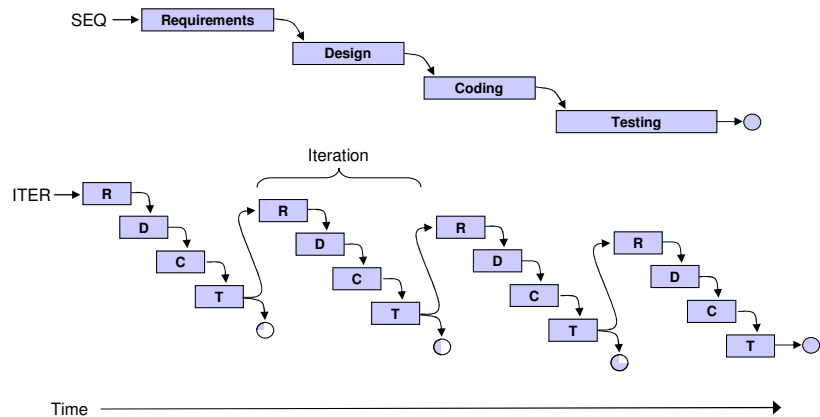
It also facilitates external reuse because a well-designed architecture will use architectural patterns that have proved successful in other projects, and therefore, allow reuse of existing components.

A well-designed architecture helps organize the work done on the project.

For example, with LANDIS, the architecture has disturbance components, and developers organize their work by focusing on an individual disturbance component.

And third, the Unified Process is an iterative process which is best illustrated with the next figure.

Sequential vs Iterative Development



Based on Figure 4-2, Kruchten 2004

This slide shows how the classic sequential approach to software development compares with an iterative approach like the Unified Process.

In the sequential approach, all the requirements are documented before any design is done. Once all the design is completed, then start writing code, and once all the code is written, the testing is done.

The result is the complete software system.

Research has shown that projects that are successful don't use this approach, but rather follow an iterative approach.

In this approach, each iteration is a mini-project that builds a piece of the system.

In an iteration, we do requirements, design, coding and then testing, but the result is a piece of the whole system.

What we learn during this iteration is used to plan subsequent iterations.

For example, we may learn that there are issues about implementing a particular piece of the system which we hadn't thought of.

Each iteration yields a bit more functionality, so that in the end, the whole system has been built.

The advantage of this approach is that it allows the project to deal with ongoing changes in the requirements.

Requirements naturally evolve, and as I mentioned in my statement about software development, this is particularly true with research software.

Also, this approach minimizes the consequences of missing deadlines.

If a project runs out of resources, and must stop prior to finishing the whole system, at least, there is working software that has some partial functionality.

It's not all or nothing, as it is with the sequential approach.

Unified Process – Phases

- Inception – *Establish a common vision*
- Elaboration – *Build and test the risky core*
- Construction – *Build and test the rest*
- Transition – *Deploy the system*

The iterations in the Unified Process are grouped into 4 phases.

In the first phase, Inception, a common vision for the project is developed, for example, the LANDIS-II project charter.

It identifies some of the key features of the software.

The Unified Process involves actively managing risks to the project, and in the Inception phase, the major risks are identified.

In the Elaboration phase, the top risks are addressed first.

In other words, the parts of the project that are the most risky, uncertain that they will work, are tackled first.

Possible solutions for these components of the system are designed, coded, and tested in this phase.

The result of this phase is that the system's core architecture and its key elements have been built.

So in the 3rd phase, the Construction phase, the rest of the system is built and tested.

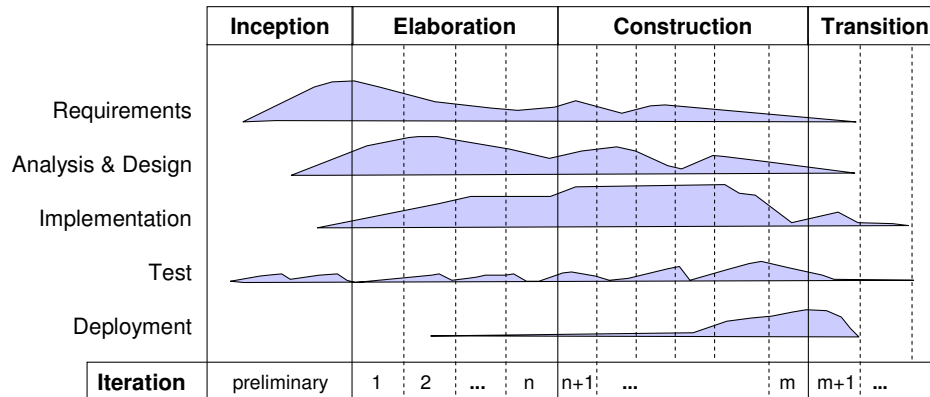
Typically this is the longest phase and has the most iterations.

At the end of this phase, the system is believed to be ready to deployed.

So in the final phase, Transition, the work focuses on transitioning from releasing the software for internal use by the project team, to releasing it externally for others.

The phase begins with the first beta release, and ends when the system is ready for users to install (for example, it's available for download from our web site).

Phases & Iterations



Based on Figure 9.5, Larman 2004 & EUP Lifecycle

This figure shows how the phases are composed of multiple iterations.

On the left are listed the work activities that used to be considered phases in the sequential approach to software development.

In that approach, they occurred separately, one after the other.

But as you can see, in the Unified Process, these activities (called “disciplines”) occur simultaneously, with each one receiving different emphasis depending upon the particular phase.

For example, requirements receives a lot of emphasis in Inception and early in Elaboration, while implementation (coding) occurs across all phases, but is really emphasized in the Construction phase.

The iterations are numbered starting in the Elaboration phase, because recall that an iteration produces a piece of working software, and typically the Inception phase produces no software.

Therefore, its iteration is called “preliminary” or sometimes “Iteration 0”.

The figure shows that coding can occur at the end of the Inception phase.

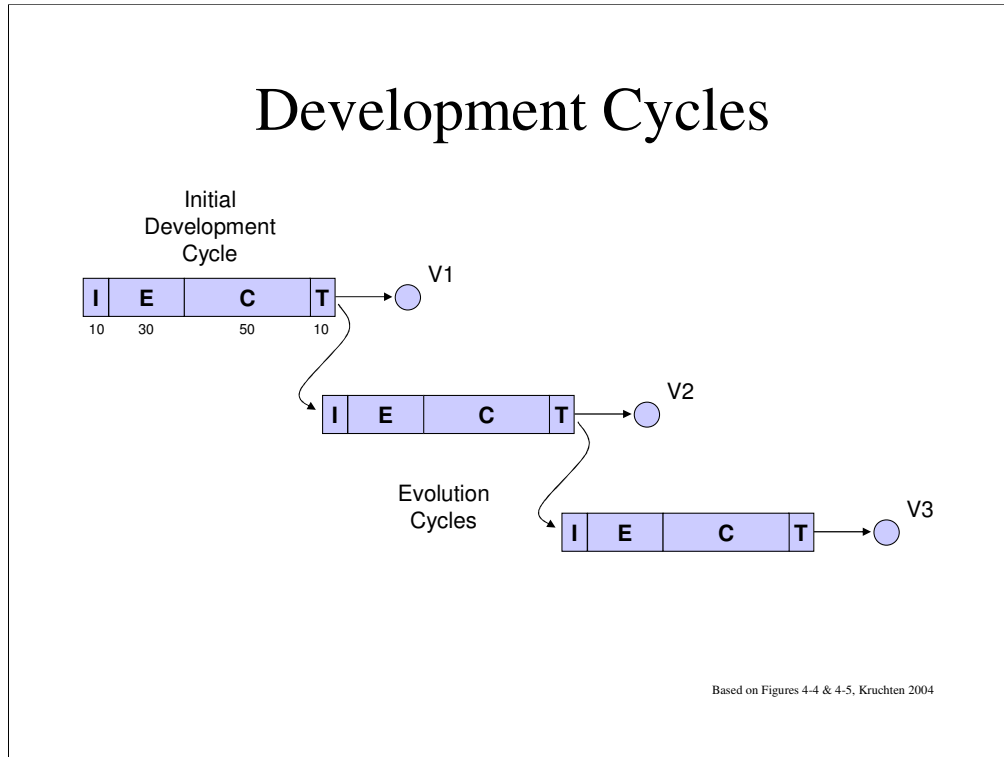
This would have to do if a prototype of a user-interface is created to help users refine their definitions of the software’s requirements.

Please note that the length in terms of time (width on the figure) of iterations varies among phases.

In Elaboration, iterations are longer while the risky uncertain components are worked on.

Once, those are done, work shifts focus onto the more typical components, so that the iterations in the Construction phase are shorter, say 2 or 3 weeks typically.

Development Cycles



The 4 phases in the Unified Process constitute a “development cycle”.

This figure shows a series of development cycles that happen during a software system’s lifetime.

It starts with the first one which is called the initial development cycle, which results in the first version of the system.

Then subsequent cycles known as evolution cycles occur as the system evolves throughout its lifecycle, resulting in updated versions of the system.

The numbers under the initial development cycle in the figure are percentages of time that each phase takes in a typical initial cycle.

For example, Inception takes 10% of the time, Elaboration 30%, Construction half of the time, and Transition the final 10%.

The percentages apply to typical initial cycles and not evolution cycles.

Evolution cycles can widely vary because such a cycle may be producing the next major version (e.g., version 2.0), or it may be producing a minor version update (e.g., version 1.1).

That concludes my brief overview of the Unified Process framework, so I’d like to finish by talking about how it relates to our project.

Process Goals for LANDIS-II

- Need to use a process
- Should facilitate primary objective:
Build working software
- Use a proven process
- Must be adaptable to our situation

I believe there are several objectives related to the process for this project.

First, we need a process.

The prior work on LANDIS has already shown that having no process leads to ad-hoc development that uses resources inefficiently, and is often ineffective in that it may not produce exactly what the users need.

On the flip side, we don't want too much process.

So our second goal is to have a process that helps us with our primary objective which is to build working software.

The process should be a tool to achieve this objective, and we don't want the use of this tool to become a burden.

Third, we don't want to have to re-invent the wheel, or use the latest-fad in terms of process.

We want to use a process that's proven, and has been used successfully on lots of projects.

And fourth, we need a process that is customizable, that we can adapt to our particular situation, given our specific needs and circumstances.

I believe the Unified Process can meet all these objectives.

UP & LANDIS-II

- Reassess commitment to UP
- Strategy for adopting UP

So, there was prior commitment to use the Unified Process.

I suspect some of this information I presented today about the Unified Process is new.

In light of this, we need to reassess if the team is still committed to using this process on this project.

And if the commitment is still there, then we need a strategy to adopt this process, and integrate it into our project.

I believe that strategy should be, like the Unified Process itself, an iterative and incremental approach.

We can gradually adopt elements of the process framework as we go through the iterations.

During the review step at the end of each iteration, we not only assess what we learned about the software we're building, but also what we learned about using the process.

Then as we develop the plan for the next iteration, we can include activities related to the process (for example, more education and training) as needed.