

IST-2001-32133

GridLab - A Grid Application Toolkit and Testbed

D14.1: Quality Assurance Plan

Author(s):	Jarek Nabrzyski, Juliusz Pukacki
Document Filename:	GridLab-14-QAP-0004-QualityAP
Work package:	WP14 Project Management
Partner(s):	Poznan Supercomputing and Networking Center
Lead Partner:	Poznan Supercomputing and Networking Center
Config ID:	GridLab-14-QAP-0004-1.0
Document classification:	INTERNAL

Abstract: In this short document we describe the general rules for designing the GridLab project. As discussed in Frascati and finally decided in Poznan we will be using UML for the project modeling and design. We describe here the process of problem solving and development process lifetime. These are the general rules to be followed by all the GridLab partners.





Contents

1	Modeling	2
2	Problem solving.	2
3	Development lifecycle model.	4
4	Development lifecycle phases.	5
5	Incremental builds	6
6	Glossary	7

1 Modeling

Modeling is a way of thinking about problems organized around real-world ideas. Models are useful for: - understanding problems - communicating with everyone involved with the project - modeling enterprise - preparing documentation - designing implementation

Model is based on an abstraction that extends human capability to deal with complexity of the problem. The abstraction hides system's details, making it possible focus on overall structure. The tool to describe model of the system is the UML - Unified Modeling Language. UML is not a method of designing system, but the (mainly graphical) notation that methods use to express designs. The UML includes set of consistent diagrams that may be used to describe and communicate a software system's requirements, design, and code. It can be used to provide views of the system design and requirements at different levels of abstraction, and that common view is helpful for collaborative design. The UML artifacts that are used to capture the requirements and design of a system, and that should be used during development process are:

- **Use-case diagrams:** document the system's dynamic requirements; there are two levels of use-case diagrams: user level, which describes how the users interact with the system, and developer level, which describes how the system components interact; the user level use-cases place requirements on the system; the developer level provide requirements to subsystems;
- **Class and package diagrams:** provide a view of the class design of the system; the diagrams contain a representation of the classes, and packages (related sets of classes) and how they are associated;
- **Sequence diagrams:** tie together the use-case diagrams and the class diagrams; they show how the classes collaborate to carry out a use case;
- **Component diagrams:** show how the classes relate to the actual code; in particular they define the libraries, and subsystems that are integrated to make up the system;

That is only general view of UML, but it is out of scope of this document to give complete description of each UML diagram type.

2 Problem solving.

A software development process is a method to organize the activities related to creation, delivery, and maintenance of software system. Four main activities in software development can be distinguish: - scoping: ensuring that the problem is fully understandable, - designing: developing an approach to solve the problem (usually using some sort of diagrams) - implementing (building): carrying out the design - verifying: confirming that the solution actually solves the original problem Problem solving does not have to progress in a linear manner from one activity to the next. It is not needed to wait until problem is fully scoped before design started. But on the other hand however, to start design some work have to be done in scoping.

Development process can be divided into phases - the periods of task-related work during which the problem-solving activities occur. Phases are not strictly tied to problem-solving activities since the activities often span the phases.

Although the phases are tied to the activities, the mapping is not perfect. The phases are not marked by completion of activities, but by achieving a level of understanding of the problem and the solution. Figure 1 illustrates the relationship of the phases to the activities.

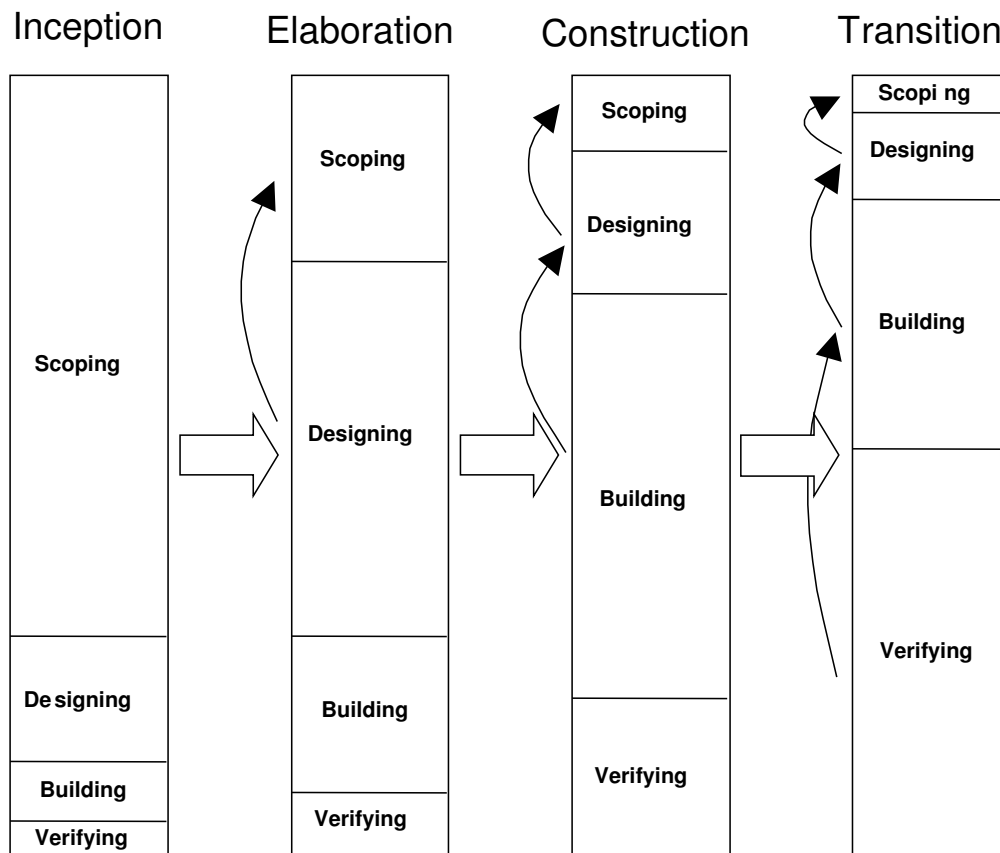


Figure 1: Problem solving - phases and activities

The phases are:

- **Inception**

During this phase the scope of the project is decided, the goal is to understand what the project is and what is not. It is not only gathering requirements but also building initial mental model of the problem. For example if the problem contains a lot of details, it should be decided to cast a simpler, less-detailed problem. Once it is understandable how to approach the simpler problem, it is possible to deal with details later. The most of activities in this phase is scoping, but other activities may also take place. As understanding of a problem is gained, it is possible to engage in design, implementation, and verification to confirm understanding.

- **Elaboration**

This phase is entered when there is sufficient understanding of the problem, and mental model designed. The goal of this phase is to design a solution, and also to discover gaps in understanding - designing activity helps to refine mental model.

- **Construction**

This phase begins when the design is in place. The most important activity in it is implementation, but some design work is also possible.

- **Transition**

When some partial solution is created it is possible to enter transition phase. It mainly

consists in verifying that solution. During that process some flaws may be discovered, which may be in implementation or may ripple all the way back to understanding of the problem.

3 Development lifecycle model.

The software development literature contains a variety of lifecycle models: waterfall, spiral, rapid application development (time boxes) and controlled iteration. All lifecycles have scoping, design, implementation, and verification activities, but they differ in how they schedule and organize the practice of these activities.

One of the most commonly used lifecycle model is controlled iteration model. It takes advantage of the flexibility and modularity of object development to provide a lifecycle that both matches how people work and allows for sufficient management control. The phases of the controlled iteration model are the same as those in problem solving described earlier:

- Inception: achieving initial understanding and agreement of the product definition (what will be delivered)
- Elaboration: achieving initial understanding and agreement of the product's detailed design (how it will be build)
- Construction: creating the initial fully functional product build
- Transition: delivery of the product that meets the initial goals

Note that the goals are not tied to the completion of the activities, only that activities are developed well enough so that the necessary level of agreement and understanding is reached to enable progression. It is understood that the product content and design may change throughout the development.

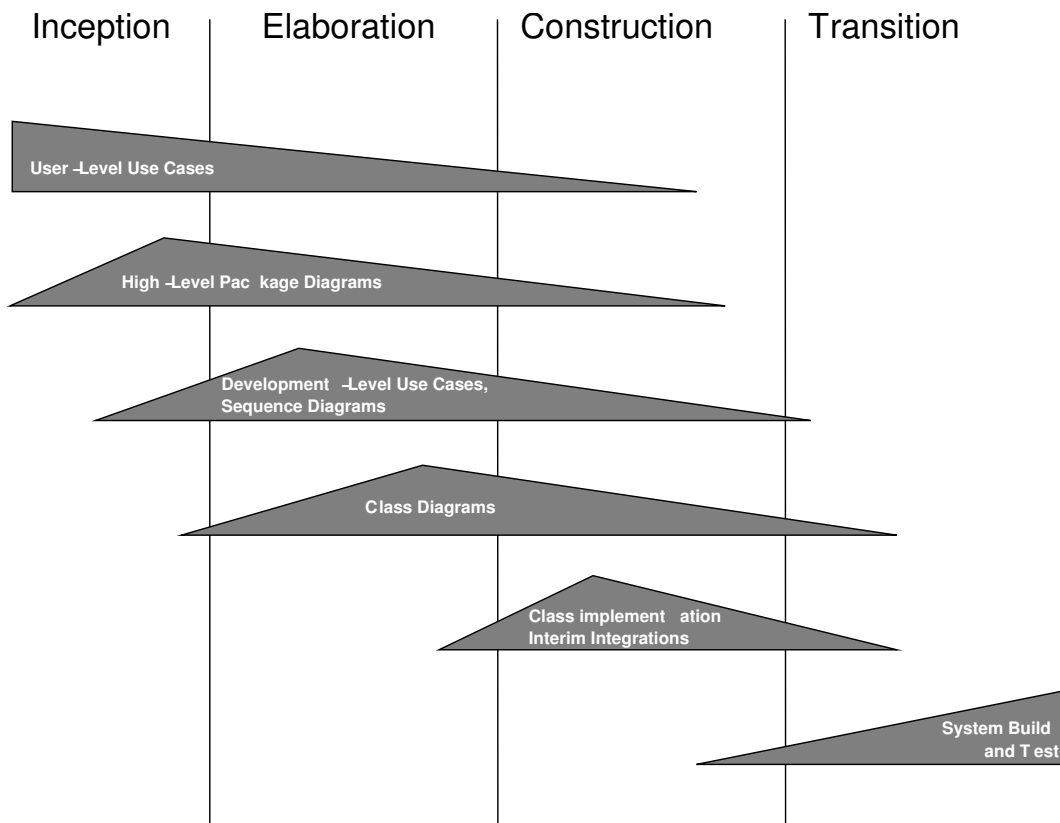


Figure 2: The controlled iteration model.

4 Development lifecycle phases.

Inception

In inception phase, the goal is to gain a common understanding of the product’s definition. By including customer (end user) in this process it is possible to complete this phase by agreeing on the product requirements. The agreement is reached by developing set of use cases, and other artifacts that specify the behavior of the desired system. To end this phase it is not required to have complete set of requirements.

Elaboration

The goal of elaboration phase is to complete enough of the detailed design to start building the code. In practice it means that the object class and package diagrams are to be created. Often this phase ends when there is no way to proceed without going ahead with the building. The most important activity in this phase is design, but other activities are also possible: in particular, the requirements are refined as new issues arise during design activity. In addition some amount of implementation may be useful as a way to gain confidence in the feasibility of the design.

Construction

In the construction phase, the goal is to build functionally complete, operational code that is ready for system test. In this phase frequent interim system and subsystem integration take place. This integration of code occurs on various levels as it is build, so there is no need to wait until the end of the phase to see if the code comes together. The construction phase ends when all of the planned use cases can be executed.

Transition

This is a phase of system tests, removing bugs, alpha and beta releases. The focus is on the transition of the code from the developers to the customer.

Traversing through the phases, the product becomes increasingly well defined and concrete. The specification is not assumed to be complete and unchangeable from the beginning, but hardens throughout the development cycle. Change in requirements is allowed and facilitated early in the cycle and becomes more expensive, but not impossible, toward the end of the cycle.

5 Incremental builds

One pass through each of the lifecycle phase produces a system build. However it is a good idea to develop a product to release with incremental builds - a sequence of builds where each has greater functionality than the last (Fig. 3).

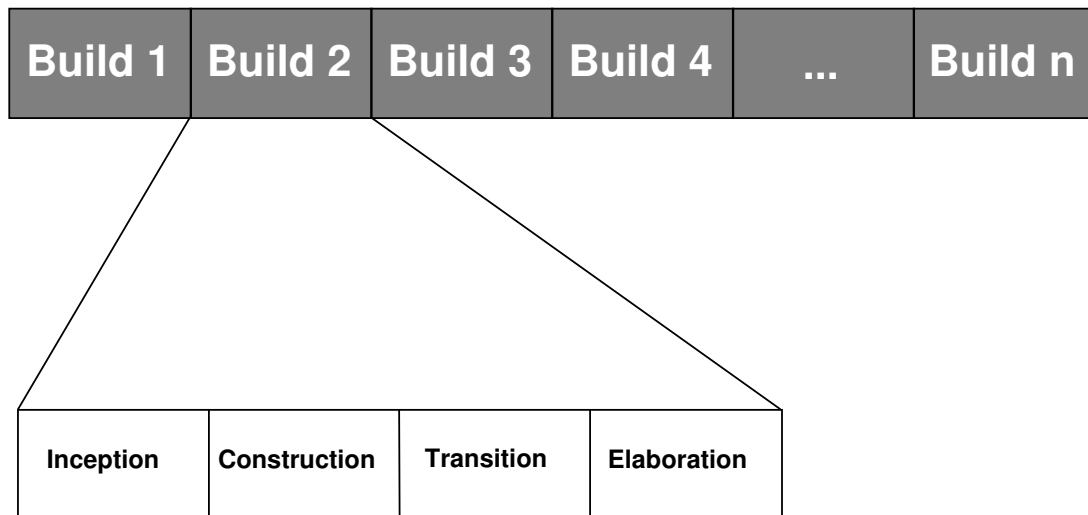


Figure 3: Incremental controlled iteration

6 Glossary

Artifact - a documented output of an activity or a task in the life cycle. Artifact includes request for proposal, statement of work, requirements, plans, specification, design, code, test document, user's manuals, etc., but not managers' and engineers' work notes/files, progress/evaluation reports, correspondence, etc. It may also include interim artifacts which are used in producing final artifacts (deliverables). (e.g. UML comprises a collection of artifacts that are used to capture requirements and design software system.)

Software development process - method to organize the activities related to creation, delivery, and maintenance of software systems.

UML - is a standard notation for the modeling of real-world objects as a first step in developing an object-oriented design methodology.

Use case - narrative operational description of how the system is used. It captures the system's dynamic functional requirements in clear familiar terms, so that both developer and consumer understand them.

Release - a complete version of the product, which has been polished, tested, hardened, and prepared for release to the customer

Build - a coherent version of a system that meets predetermined functionality and has completed a development cycle. Builds are often demonstrable versions of the code

Phase - the components of a development cycle from inception through transition

References

- [1] Grady Booch: "Object-Oriented Analysis and Design with Applications".
- [2] Murray R. Cantor: "Object-Oriented Project Management with UML".
- [3] Martin Flower: "UML Distilled".
- [4] Ivar Jacobson, Grady Booch, James Rumbaugh: "The Unified Software Development Process".
- [5] Craig Larman: "Applying UML And Pattern".
- [6] Terry Quatrani: "Visual Modeling With Rational Rose 2000 And UML".