Presented on Unigraphics User Group 1999 Spring Conference: Managing design evolution, March 14-18, Newport Beach, California, USA

OBJECT TECHNOLOGY DEVELOPMENT AND UNIGRAPHICS

Graeme Britton
Associate Professor
School of Mechanical and Production Engineering
Nanyang Technological University
Nanyang Avenue
SINGAPORE 639798

Ma Yong Sheng,
Project Leader,
QuickMould Development Team,
Gintic Institute of Manufacturing Technology,
71 Nanyang Drive,
SINGAPORE 638075.

Tor Shu Beng
Associate Professor
School of Mechanical and Production Engineering
Nanyang Technological University
Nanyang Avenue
SINGAPORE 639798

INTRODUCTION

Gintic Institute of Manufacturing (Gintic), Nanyang Technological University and the National Computer Board (Singapore) have been involved in developing a commercial Object Oriented (OO) software product for the last three years. The product is CAD software for designing plastic injection moulds; it is called QuickMould. One of the objectives of this project was to develop a capability in Singapore to undertake commercial, OO, CAD/CAPP software development projects. We chose Unigraphics (UG) as our platform because it has an extensive development toolkit and Unigraphics Solutions provide very good support. A major disadvantage in building on top of UG is it is not Object Oriented, although the development toolkit does support C and C++ compilers.

In this paper, we share our experience in building up an OO development team and environment using the UG CAD platform. It should be noted that we write our code in C++ and initially we were developing solely on HPUX. We now have the capability to develop on either HPUX or Windows NT.

SOFTWARE PROCESS METHODOLOGY

Object Oriented Pte. Ltd. (OOPL), an Australian company, has developed a full lifecycle, object-oriented, software engineering process suitable for commercial software development. It is called Mentor^{TM}. Mentor was chosen for our project because it is a full lifecycle methodology and allows incremental and iterative software development.

A software engineering process is a time-sequenced set of process units that is used to transform a user's requirements into a software system. Mentor's software engineering process covers the development of object-oriented software systems, from initiation to deployment.

It encompasses all aspects of the software engineering process, including requirements gathering, system design, implementation, testing, project management and quality. Mentor also provides the key elements required for ISO 9000 quality accreditation and has been used by a number of organisations as part of their overall quality system.

Figure 1 shows the underlying principles and constructs used within Mentor, arranged as a set of building blocks.

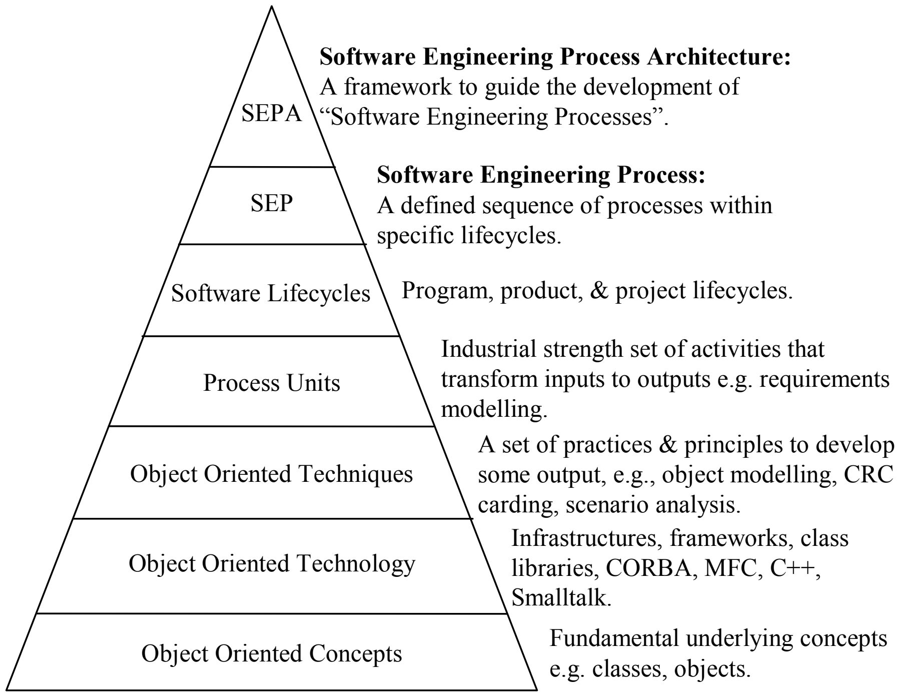
SEPA
Software Engineering Process Architecture: A framework to guide the development of "Software Engineering Processes".
SEP
Software Engineering Process: A defined sequence of processes within specific lifecycles.
Software Lifecycles
Program, product, & project lifecycles.
Process Units
Industrial strength set of activities that transform inputs to outputs e.g. requirements modelling.
Object Oriented Techniques
A set of practices & principles to develop some output, e.g., object modelling, CRC carding, scenario analysis.
Object Oriented Technology
Infrastructures, frameworks, class libraries, CORBA, MFC, C++, Smalltalk.
Object Oriented Concepts
Fundamental underlying concepts e.g. classes, objects.

Figure 1: The fundamental principles underlying *Mentor*.

The foundation of Mentor is the set of fundamental Object Oriented concepts; for example, class, object, inheritance, etc. The next level is the infrastructure and software development languages that enable implementation of the concepts.

The third level, also built on the fundamental concepts, is a set of techniques that detail the tasks and guidelines for constructing models. Mentor incorporates a number of techniques derived from other OO methods including Booch, the Object Modelling Technique, Object Oriented Software Engineering, Responsibility Driven Design and MOSES. Mentor's techniques are notation independent. This pragmatic approach makes it possible to use a number of different graphical notations within Mentor, as long as the underlying concepts supported by the notation are defined by Mentor.

The techniques need to be packaged as process units before they are useful for commercial development projects. Process units make the techniques robust and usable in industrial settings by providing such things as deliverables templates, resources requirements and review guidelines. They are the reusable building blocks of industrial strength approaches. In short, process units provide the activities, guidelines and tasks that are applied within a software engineering process that is used to execute a software project.

The version of Mentor we used provided definitions for 17 process units: software lifecycle model, concept exploration, progamme development, requirements modelling, alternative evaluation, user interface modelling, system modelling, sub-system modelling, component modelling, repository modelling, prototyping, acceptance testing, installation, post deployment review, retirement, project management and quality assurance.

The process units are executed within the framework of a lifecycle (next level in Figure 1) and the lifecycle is defined by a software engineering process, which is defined within an architecture (top level).

In 1996, one software engineering process, known as M-SEP1, was defined within Mentor. It consists of three related software lifecycles:

- Programme Lifecycle
- Product Lifecycle
- Project Lifecycle

The relationship between these software lifecycles is shown in Figure 2. Essentially, the Programme Lifecycle consists of a set of projects, each of which undergoes a Project Lifecycle. The software lifecycles are briefly described below.

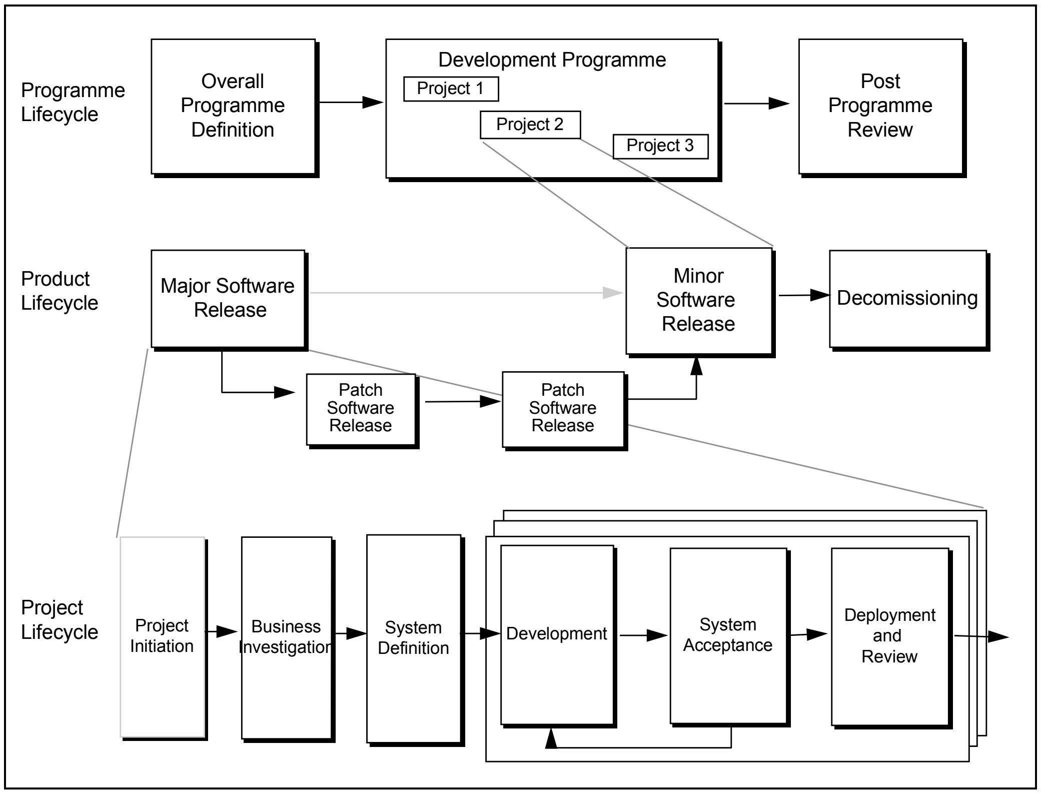

Figure 2: Software lifecycles of Mentor's software engineering process (M-SEP1).

In large projects there are often many individual projects that are in some way centrally co-ordinated or managed. A coherent and related set of projects is known as a programme of work. A Programme Lifecycle is a software lifecycle that defines a series of phases for such a programme of work. A Programme Lifecycle therefore encompasses a set of projects and may last for many years. It consists of the following phases:

- Overall Programme Definition
- Development Programme
- Post Programme Review

A software system usually has a life-span longer than the individual project that created it. Some software systems may have a very long life-span during which they are modified, enhanced and patched. The complete life-span of a software system is modelled within M-SEP1 as the Product Lifecycle. The Product Lifecycle describes the entire life-span of a software system from initial development through to decommissioning. It consists of:

- Major Software Releases
- Minor Software Releases
- Patch Software Releases
- Decommissioning

Each project undertaken on a software system follows a software lifecycle called the Project Lifecycle. The Project Lifecycle defines the phases and software processes for developing a software system from initiation to deployment. It consists of the following phases:

- Project Initiation
- Business Investigation
- System Definition
- Development
- System Acceptance
- Deployment and Review

We used the Project Lifecycle for our project.

SOFTWARE DEVELOPMENT ENVIRONMENT

All software development requires a development environment. You can develop team-based, OO applications using the Microsoft Integrated Development Environment (IDE). In 1996 when we looked at IDE, we found that it supported version control but was not really efficient for structured builds, although this could be done. To guarantee code quality you also need additional tools for memory checking and code coverage.

For several reasons (not relevant to this paper) we chose to develop on HPUX. As Mentor is based on component modelling, we needed to set up a development environment to support this. The key issues we addressed were:

- Version Control and Automatic Builds: For version control and automatic builds we used CVS and makeit (both freeware) which were customised for us by OOPL. Initially we developed on HPUX and then ported across to the other operating platforms. We can now do automatic, structured builds on HPUX, Sun Unix and Windows NT. It should be noted our development environment is capable of handling digital documents (e.g. requirement specifications) as well as code.

- Support Tools: The UG development environment has some unusual features. We had no success in getting commercial support tools to work with UG on the Unix operating system. We finally managed to get NuMega BoundsChecker (Visual C++ edition) to work in the Windows NT environment. So now we develop on Windows NT and then port from there. Currently we are in the process of implementing NuMega TrueCoverage.

- Code Standards: The HPUX compiler is very forgiving, but Windows NT is not. So if you are programming for multiple platforms then it is advisable to base your C++ programming standards on Windows NT.

LESSONS LEARNT

We started our project with a team of 5 people who had no OO experience at all. Thus we had to train the team in OO concepts, C++ programming, Mentor and the UG toolkit. This was a major challenge. OOPL were contracted to provide the following: Mentor manuals, training in Mentor, training in C++ programming, Mentors to assist us in implementing Mentor and sub-contract programming. Some lessons we learnt are:

- Architecture: As UG is not Object Oriented it is necessary to provide an interface between the UG database and the main OO program. This can be done two ways depending on the circumstances. The first and easiest method is simply to provide interface objects to write to and read from the UG database. The second method is to build an OO infrastructure on top of UG. This is what we did because we wanted a platform technology that could be used to develop applications over different application domains.

- Incremental and Iterative Development: Normally incremental and iterative development is used as a strategy for delivering the software to a client. As our team was new to OO and Mentor, we decided to use this approach for training. We took a small part of the project and used it as a mini-project in order to take the team through one project lifecycle (excluding testing and deployment). This was very successful because it helped the team relate the requirement specifications and system models to the C++ code. We recommend this approach if you are new to OO.

- Learning Takes Time: One of the very painful lessons we learnt was that the learning curve was much longer than we anticipated. It has taken our team about 2 years to develop expertise in requirements modelling, system modelling, component modelling, C++ and UG API's.

- Discipline is Vital for Team-based Projects: All team-based projects require discipline to ensure project deadlines are met and software quality. We could not have achieved the results we did without the discipline provided by Mentor.

- Team Organisation: A common practice for OO development projects is to have a GUI specialist to do all the user interface design and coding. We tried this approach during our min-project, but found that it was not suitable because of overloading of the specialist. CAD application development is very interactive and it is better for each developer to develop the interface as part of his or her component. It is necessary to enforce GUI standards if this is done, otherwise components will not have the same look and feel.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the contributions of the other QuickMould team members, especially, Dr. Lee Eng Wah, Group Manager, MIT Division, Gintic, who is also the project manager for the project; and Jason Cheng and Lee Han Boon.

The authors also gratefully acknowledge the National Computer Board (Singapore), which funded the project, and Object Oriented Pte. Ltd., which provided the software process support.