WHITE PAPER EXTRACT

OOCASE User Manual

Version 4.0.4 2018-12-11

Abstract

Welcome to the OOCASE User Manual!
The purpose of this manual is to deliver a core conceptual understanding of some of the fundamentals in efficient practical information management of technical information whose life times span decades.
These core concepts are high-level and hard to acquire without concrete examples. Thus OOCASE which is a fully commercially competetive product within its scope of applicability, is used as a teaching tool to transfer this higher-level of understanding.
Intended readers are programming experienced life-time-students of any age that have severe challenges to master, and need the best education possible to implement what is neccessary to gain the long-lasting quality-of-life producing effects into foundational physical infrastructure that we all need.

Table of Contents

Preface

To Software Engineers who want to get something done without complicating it more than neccessary

OOCASE is an Objection Oriented Computer Aided Software Engineering application that significantly enhances the productivity of a software engineer or collaborating software development team.

Every tool has it's *scope of applicability*. Outside this scope there is an exponentially rising "cost/benefit" barrier that requires a different architectural and methodological approach to break through. This barrier has to do with what we humans are and the limitations of how much knowledge and information a person or small team of efficiently collaborating engineers can be masters of.

OOCASE works well for model driven software development for applications whose information models contain up to 250 classes and a similar amount of relationships. Such models may produce generated source code sizes of perhaps ½ a million lines of source code, for ONE implementation that can be maintained by a small team.

Even if OOCASE works well with object oriented information models up to 5000 classes and has been used for integration planning between software systems with more than 10000 classes and hundreds of thousands of attributes, such kinds of developments require larger teams of people that are organized in a healthy knowledge ecology by a company with healthy corporate governance and well maintained long-term win-win relationships with companies in its supply chain and its customers.

The purpose of this User Manual is to provide the theoretical framework for becoming proficient in using OOCASE, and with that conceptual understanding have acquired the knowledge potential to develop and deliver the next generation of tools we software engineers and our collaborating environment peers need, to with implementations that work in practice, cost-efficiently master the known challenges that decade spanning information systems management impose. Including providing users with high-performance.

Chapter 1 The Purpose of OOCASE

When the purpose of something is MUCH bigger than itself, it receives a divine force that can make it overcome
[The Poet]

The purpose of OOCASE is efficient use and reuse of Information Models[1].

An Information Model is a declarative design specification, or can be seen as a contract of what information a computer system software should be able to capture, store, process and present.

The information model is expressed in a language that after a short introductory training can be understood by non computer specialists, and serves as a tool for communication between 3 different groups of people. a) domain specialists that provide the requirements of information representation needed by their knowledge domain, b) computer specialists that implement a software solution, and c) end users of the computer system software who use the information model as documentation.

OOCASE is an object oriented computer aided software engineering tool with a number of unique features.

1) Calculation of information quantity in a model

2) Support for automated checks and documented quality assurance.

3) Globally unique 128 bit object identifiers for distributed development

4) Open Export / Import in many neutral formats

5) Explicit well documented quality assured and release managed meta model.

6) Automated source code generation for a number of well established software platforms.

Now it's perfectly allright and recommended to skip to Chapter 2 on page 17, since the following is only interesting for large scale project managers who build IT-systems that need to be operational for decades in whatever shape new technology allows safe information storage and exchange resources to be implemented upon.

1.1 Calculation of information quantity in a model.

Information quantity is measured in a unit called EC for elementary constellation. An elementary constellation or e-constellation is the smallest possible unit that still carries meaningful information that can be stored or transmitted as a message from a sender to a receiver.

[1] An information model defines Structure AND from the Structure directly inferable Behaviour. Inferable behavior that implements from the structure inferable application programming interfaces (API's) that follow easy to remember and use naming and parameter setting conventions and can be implemented by automated source code generators specialized for a highly optimized implementation in a particular target source code language. Other conformant languages such as UML define both structure and behavior. Large amounts of basic kinds of behavior can be derived from the structure and implemented automatically with model driven source code generators or model driven interpreters. For new application development it is cost efficient not to waste too much time on hand written software behavior before the core information structure (DomainModel) is well understood with examples that are entered into generated prototype applications populated with realistic production environment information.

The purpose of this measure is to provide decision support for which model to choose if there are several alternaive ways of modeling a certain physical product or artifact, perhaps using different software systems.

A model can be anything that has been formalized with the language constructs in the DomainModel language to a level of detail that allows software to be implemented and the models information stored in a database representation. If two models provide equivalent functionality but differ severely in complexity or computational performance on present technological infrastructure available for the purpose of the model's end user application, someone has to decide which model to choose.

Fact based measurements are descisive (or important decision support in political environments) when the accumulated cost of several decades of IT-system maintenance for serving a fleet of complex engineered products that delivers a fundamental service for the sustainment or protection of a customer's society, requires IT-support for cost-efficient management and maintenance.

In a modern larger engineering company that promotes its staff by merit and leaves plenty of opportunities for choices of a future carrer open and inviting, the time at the IT department while learning the information structures of the core business is a knowledge development platform for the staff supply that can take on challenges of the more advanced jobs[2]. Jobs that require an understanding of how to implement production capacity for new business opportunities. This in modern engineering companies generous staff meritation requires that the supply can be held up by efficient education of new staff to replace the vaccum for skill enabled career advancers that exposure to this core business knowledge in a concentrated format produces. The choise of model can actually impact the choise of paradigm for career advancement in a company, and with that the whole future for it's business[3].

The output of a modern larger engineering company in the form of life quality sustaining societal infrastructure products are frequently taken for granted by consumers. It is only the experts of these products who have the REAL power to make them deliver their output at the cost possible given the current state of the art knowledge in all those fields of of expertise who combined make such products producible at an affordable cost for the end users of the product or it's services.

Information quantity based decision support is especially important when two separate communities fight over a standard and are unexperienced[4] in each others technological domains or software implementation support for these technological domains. Unneccesary complex models are harder to teach and maintain, and divert resources from other important areas of development. Having some measureable facts may resolve disputes and get the "fighting communities" focused again on delivering added value, standing on a fact based ground, and if corporate governance is excellent, get amplified by mutual education.

1.2 Support for automated checks and documented quality assurance.

The Quality Assurance techniques applied here were adopted from the mechanical engineering industry where the high cost of failures due to errors in design specifications (drawings etc) drove this industry to develop survival skills that add some spending in the earlier stages of development. Where this added spending serves as insurance against unpleasant expensive surprises later. OOCASE supports a number of levels of automated quality checks, that aid various "check-points" in a cost efficient iterative development cycle where many people (frequently with too little time) are involved.

---

[2] Where the best examples of Engineering Companies are plain clever self sustaining career production machines for talanted people whose skills are neccessary to fix our real problems.

[3] There is always potential competetive advantage layers above a current well-known established business. Those above layers requires creative people who know the core business AND something else that none of the established market players have thought of before or delivered the investments to make it become available for a solvent enought customer basis who appreciates its new products and buys them for a GOOD reason. If the core business is obscured by an unneccessary complex model, the supply of people who understand it well enough to develop the potentially business income generating layers above it will be throttled. This is stuff that matters over decades, when "great asset" people move for reasons outside the control of the company.

[4] This is a real fact due to the enourmous size of various industrial information models, and the amount of studying time and practice it takes to become familiar enough with their details to make fact based decisions.

1.3 Globally unique 128 bit object identifiers

Ensuring global unique identifiers for objects is a re-occurring problem throughout the whole globalizing IT industry. It has with performance in applications to do, scalability and ability to produce large amounts of uniquely identified objects in parallel by people and teams that are unaware of each others existence. Where the unique object identifier issuing mechanism must provide the ability to combine results of independent uncoordinated work in a database that requires unique object identification AND traceability for quality assurance reasons, and efficient on-demand-access to linked external resources outside the local database.

The method chosen for OOCASE and plenty of production systems produced, is providing each creator of new object identifiers with a unique 64 bit identifier (HighId). Each such creator has a self managed 64 bit incrementing counter (LowId) for lifespan unique identifiers from that source.

There are plenty of other ways to solve this problem, but this approach is simple, efficient and it works in practice.

1.4 Open Export / Import in many neutral formats

OOCASE provides many ways of exporting and importing information models. Thus your models are never locked-in within this tool. If a better tool comes along (creative destruction) you can proceed with that.

1.5 Explicit well documented quality assured and release managed meta model.

The meta-model of OOCASE is licensed to all paying customers for their own implementation needs in the most empowering format. OOCASE is modeled in OOCASE. Thus if you develop source code generators for a new software platform that completely outperforms the one OOCASE is using, you are free to implement your own OOCASE tool on that new platform and migrate to that platform with all your information model assets intact[5].

1.6 Automated source code generation for a number of well established software platforms.

Source code generators are available for a number of SQL92 compliant relational databases, Smalltalk, C++ and a few other programming languages.

The rest of this introductory chapter is there for readers who wan't to understand the benefit of a more efficient standardized "asphalt laying machine" for putting "tarmac on the emerging INFORMATION gravel roads of all diverse shapes and sizes, so they without numerous severely errorprone and costly reloading can carry the truckloads of global information we need to ship to the computational centers that can convert it to reliable decision support for our industrial leaders and national goverments.

[5] This may seem stupid with regards to the self-sustainment principle of the company making a living on OOCASE. However in the perspective of global warming whose solution is more important than the self-sustainment of a particular company whose employees can find a living somewhere else, it's non-productive with regards to over history gathered experience to prevent "creative destruction" to happen if the new alternative over time and by facts and evidence delivers a much better output performance with regards to achievement of the goals setup in the UN 2030 Agenda.

1.7 Purpose of the Infological Approach

To understand the purpose of any kind of software one has to analyze its role in the larger whole.

The following is a quote from the preface of [Sundgren 73] which describes the foundational infological theory on which OOCASE builds.

> "An infological approach to data bases" reports parts of the data base research and development work which has been carried out over a number of years at the National Central Bureau of Statistics, Sweden. Professor Börje Langefors, University of Stockholm, Department of Administrative Information Processing, has been the scientific supervisor of the reported project. Very briefly the objective of the project has been to develop an integrated theoretical framework for design of large-scale data bases. The framework should
>
> (a) enable people who are not data processing professionals to co-operate actively and constructively in data base design projects
>
> (b) make it possible to transform systematically the problems, desires, and requirements of those who are affected by the projected data base into problems which can be tackled by data processing specialists
>
> (c) enable data processing specialists to analyze the computer-oriented data base problems systematically and with sufficient precision
>
> (d) make it possible to design data bases with which decision-makers, planners, and researchers within different specialized fields could interact constructively, even if the information needs of the interactors are complex, and even if they lack knowledge about computers and computing
>
> There are definitely different opinions among authorities in the computing world as to whether it is feasible to cover all the aspects (a)-(d) within one an the same framework. This report supplies evidence in support of the hypothesis that an integrated approach is both feasible and necessary for the success of large-scale data base undertakings."

The above quote from year 1973 is still valid, however the situation has improved. (a) has been improved with graphical representations of information models that are used in interactive development seminars where a mix of domain specific expertise participate and all understand what they are talking about so efficient communication can take place. (b) and (c) have for the purpose of implementation of basic information handling software functionality for delivering fully functional prototype software implementations been fully automated for certain target platforms. (d) has been significantly improved with automated model driven declarative implementation of software prototypes from information models, that enable domain experts to express their expertise with large scale examples, that reveal the "problems" in the details, where the *Information Model* does not adequately represent reality. Some stuff in seminar or prototype evaluation situations are "gutt feelings" of participating experts, and it requires certain "emotional language literacy" and social skills by a software engineer/seminar leader to get that information out. According to a non-disclosed source there are 39 different emotional expressions that experienced people use while communicating interactively. Human skills in understanding and acting efficiently on the cues of non-spoken emotional language AND knowing the domain of the expert to a level where the facts can be brought out by asking the right questions requires a special brand of people, of which YOU are a candidate.

Or you can focus on the technical implementation parts of translating declaratively specified *Information Models* to efficient software implementations on the latest superior hardware and software platforms.

1.8 Towards an Efficient Shared Information Highway Network for Implementing the UN 2030 Agenda for Sustainable Development

Some problems can not be solved in traditional ways since there is no-one who owns the problem. Or there exists no single entity with enough resources or authority to solve the problem in practice. Or the entity producing the problem is not within the authority of the entities subjected to its effects.

In order to solve a problem it must first be understood. The UN 2030 agenda for sustainable development has set out a goal. To reach that goal, we need a plan for how to get there.

A typical approach could look like:
1) Build a reasonably adequate map of the current situation
2) Identify spots where investments would provide largest return with regards to goal achievement
3) Allocate resources to fix those problem spots
4) Implement the fixes and restart at 1)

Besides ignoring the natural law of self-sustainment[6], the above approach hit's it's exponential boarder of applicability rather quickly, due to a problem we encountered in Software Industry a long time ago:

The Language Problem

We still have not solved it satisfactorily however software industry rolls on, in its complex supplier value chains, each actor in it's own little language islands, at the "speed and load capacity of a horse/8-bit CPU", where we could use a "modern truck/64-bit multi-core" instead.
But that is unfortunately not possible, since there are to few "roads/standards" that can "carry such a truck/make use of available information exchange eco-system" (even if there are instances of such "roads" and "trucks" in certain nisches that have an enormous turn-around).
Even if it is possible to implement a "truck" for, for instance "Environmental Data", there would not be a large enough market for it. The infrastructure ranging from CPU, OS, DataBase, Network Communication Protocols, User Interfaces, Local User Language Adaptations, needs to be implemented with instances for which there is an educated work force who can install and maintain them.
And finally the end users, which collectively are the most expensive and valuable actors in this chain, needs to be educated in order to know what decisions to take based on the delivered "Environmental Data".

But there is a solution to this. A distributed one. One that will require an agreement or arrangement of peace and non-hampering interference by external actors that have their own agendas (or internal problems) and don't care about "the truck's" success since it will not be under their control.

---

[6] Law of self sustainment) An active entity e.g. @) biological cell, a) plant, b) animal, c) person, d) organization, e) company or f) nation;
that can not find a way to sustain itself with; @) nutrition and energy (provided by its organism), a) nutrition and sunlight, b) food, c) food and housing, d) work force, e) work force and income to pay taxes and work force, f) tax income (to pay for education, defence and law enforcement), educated workforce, peace and stability, law enforcement (that allows its educated citizens to set up companies that are not robbed of their resources (material, money, working time), and employ an educated workforce that can produce the added value necessary to generate tax and the companie's self sustainment);
will starve and eventually perish. The basis for action is energy, without energy action is not possible.

The solution is based on some lessons from the evolution within software industry and other domains, and delivers shortcuts that can shorten the time table from decades to years.

To explain this the "generalize from examples" approach is taken

1.8.1 The Software Expansion

The personal computer was founded on the invention of the CPU on a chip. Success stories of chips like the Intel 8080 lead to low cost competitors like the Zilog Z80, Motorola 6800 and MOS 6502. The possibility to mass produce personal computers at a low cost lead to an extraordinary expansion during the late 1970's. The availability of PC's in all kinds of different industrial, academic and personal environments lead to the creation of software industry that in turn had an extraordinary expansion in the 1980's.
Due to the large distributed presence of personal computers and software tools for software development, the same scale-up problems when software grew larger were detected in wast amounts of different places. Plenty found their own solutions and put them into their own software and software development methods.
In the 1980's a very diversified ecosystem of programming languages, software development methods, tools and software products had emerged.
In academic and other meeting arenas for software developers, people realized that they had common problems with translating data from one program to a suitable format that could be used in another. The same thing appeared in larger software development projects where models and methods from different interacting sub suppliers had to be interfaced or integrated. There was a growing need for a common language.

There seems to be three approaches for finding a common language. These have been tested in plenty of computer science historical peak events and delivered their output in the form of publications, standards, organizations and companies maintaining the standards. The three identified alternatives are:

1) Some clever people with indepth knowledge of the problem, design something that takes the best out of everything they know and design something new that solves the problem.

2) A large group of representatives with different backgrounds, requirements and visions come together and with fact based arguments and efficient negotiation techniques come up with a compromise that works and deliver an output that is useful to all participants.

3) Some actor goes ahead and markets its solution to a scale that it in practice becomes the de-facto standard that gets the most users and thus everyone, in one way or another, has to adapt to.
A lock-in that, if the technology is inferior compared to others, may put a suffocating blanket on development seen from a larger context and the potential available in its large user base.

OK, so what are we supposed to say about this?
A Swedish famouns quote is: "Ja, det är för jävligt" and that ends the story with a statement that everyone can agree with. A big sigh and no change. However that comment is a violation against since long gathered wisdom. The ones that know something better need to pursue their acts, knowing and learning more, and facts if proven usually makes a difference. You might know where the facts are, however you must find them and present them in a way that serves the purpose of change towards the better.

And remember, it usually does not matter what a GOOD leader who cares about her/his followers does, as long as it put's unity within the group which ends internal fighting and frees up working resources for working towards a goal that leads to an improvement.
So lets focus on combatting global warming, and the facts about what is needed for that will fix the rest.

1.8.1.1 1) Engineered Standard - Relational Database Language (SQL) Example

Proos:

+

Cons:

-

1.8.1.2 2) Negotiated Standard - Unified Modeling Language (UML) Example

Proos:
+
Cons:
-

1.8.1.3 3) DeFacto Standard - (somewhat sensitive to select the example) Example

UNIX, Windows, Linux, C, C++ etc.

Proos:
+
Cons:
-

1.8.2 The Railway Expansion

With the invention of the steam engine, and the landmark locomotive Rocket , it suddenly was possible to move heavy goods and passengers over larger distances that outperformed "horse and carriage". Railways were built in all kinds of places in Europe by entrepreneurs with a transport business idea by different contractors.
As the different tracks came to meeting ends, it eventually became clear that the whole transportation system would become much more efficient if there was a standardized track size. The benefit with a common standard was that the goods would not have to be moved between trains running on different track sizes at the meeting points, and that a manufacturers of locomotives and waggons would get a much larger market if all rail-tracks had the same dimensions, thus enabling mass production of the same designs, with a better profit margin as result, and competition which lowered the prices for "rolling transport infrastructure".

1.8.3 The Shipping Industry Expansion

A break through in the cost efficiency of shipping was the invention of the container. A standardized package for all kinds of goods that could be transferred between ships, railway carriges and trucks reduced time and cost at goods transit points. Just the lowered transport cost made it possible to trade larger volumes of goods at a lower profit margin, thus increasing trade.

The common lesson from all these examples is that the EXACT formulation of the contents of a standard is not that important, however it MUST work in practice, it MUST be efficient compared to the state of the art, and it MUST have a self-sustaining system structure where all participant roles in that structure have win-win relationships to each other. Who is taking a particular role in the system structure is not that important if that actor maintains healthy win-win relationships with its partners in a supply chain AND maintains healthy co-opetition with its competitors, where they collaborate on developing efficient standards for the higher layers that are not yet mature enough or large enough to provide the volume benefits of a by GOOD standard enabled mass market.
There is always a higher layer for the actors that are thrifty, follow the natural laws of healthy business ecosystem dynamics and build their value adding products on the best available standards.

Chapter 19 Summary and Conclusions

The design of OOCASE has its roots in the enourmously enthusiastic software industry and academic computer science research advances in object-oriented programming, object-oriented databases, expert systems, programming environments, model driven software engineering and complete model driven application compilers that exploded in a period around 1988-1992. The core highly efficient design patterns evolving during that intense period have now been production battle tested for 30 years and plenty of practical experience has been gained from information life in an environment evolving under Mores Law.

The core technical information science has not changed, and the DataDictionary and DomainModel language has remained the same except for the adaptation to IEC 61360 in 1997 where Property was renamed to DataElementType, and augmented with standard attributes to be able to import large industrial standard libraries into the DataDictionary.

The core theory and the concepts have no competitor as simple and generic as OOCASE with the same platform neutral capabilities amongs widely available standard programming languages.

The addition in 2015 with improved functionality for Quality Assurance with Edition, Version and Release (QAEVR) management following SemVer 2.0, with full traceability through the releaseBasedOn(Highid, Lowid, Version, Release) attributes and renumbering of object identifiers when issuing a new release of a model, enable full distributed in parallell version tracking by independent organizations that know nothing about each other, while still being able to trace the version history in distributed independently maintained repositories in any SQL92 compatible relational database or simple TAB-separated table text files for the classes of the information model (DataDictionary and DomainModel).

This manual is a summary of what someone who really wants to make a long-term meaningful difference needs to know with regards to technical information that needs to be production live and maintained over decades while hardware and software platforms and programming languages change.

A. References

[CORBA 1991] Object Management Group, "The Common Object Request Broker: Architecture and Specification", OMG Publications, http://www.omg.org

[CORBA 2012] Object Management Group, "Common Object Request Broker Architecture (CORBA) Specification, Version 3.3", OMG Publications, http://www.omg.org

[Fowler 2003] Martin Fowler, "UML Distilled, 3rd Edition: A Brief Guide to the Standard Object Modeling Language", https://www.martinfowler.com/books/uml.html

[Gamma et.al. 1995] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, "Design Patterns - Elements of Reusable Object-Oriented Software", Addison-Wesley, ISBN ISBN 0-201-63361-2, , 1995, pp. 395

[Goldfarb 1990] C.F. Goldfarb, "The SGML handbook", Oxford University Press, 1990, ISBN 0-19-853737-9

[IBM 1989] IBM, "System Application Architecture - Common User Access - Advanced Interface Design Guide", International Business Machines Corp., 1989, Document Number: SY328-300-R00-1089

[Langefors 66] B. Langefors, “Theoretical Analysis of Information Systems.”, Lund: Studentlitteratur, 1966.

[Langefors 93] B. Langefors, “Essays on Infology, Summing up and Planning for the Future”, Gothenburg Studies in Information Systems, Department of Information Systems, University of Gothenburg, Report 5, Augusti 1993.

[Johansson 1996] O. Johansson, "Development Environments for Complex Product Models", 1996, ISBN 91-7871-855-4

[SemVer 2.0.0] Tom Preston-Werner, "Semantic Versioning 2.0.0", Semantic Versioning, http://semver.org/spec/v2.0.0.html

[Sundgren 1973] B. Sundgren, "An Infological Approach to Data Bases", National Central Bureau of Statistics, Sweden, and University of Stockholm, Dept. of Administrative Information Processing, Beckmans Tryckerier AB, Stockholm 1973.

[Sundgren 1989] B. Sundgren, "Conceptual Modeling as an Instrument for Formal Specification of Statistical Information Systems", National Central Bureau of Statistics Sweden, 1989:18.

[UML] OMG, "Unified Modeling Language (UML) Resource Page", http://www.omg.org/uml

[UML 2.5.1] OMG, "OMG Unified Modeling Language (OMG UML) Version 2.5.1", Object Management Group, OMG Document Number: formal/2017-12-05, December 2017,