Lecture 8

Computer-Aided Software Engineering

Dr Neil Maiden
Dr Stephen Morris
Dr Wolfgang Emmerich

School of Informatics
City University

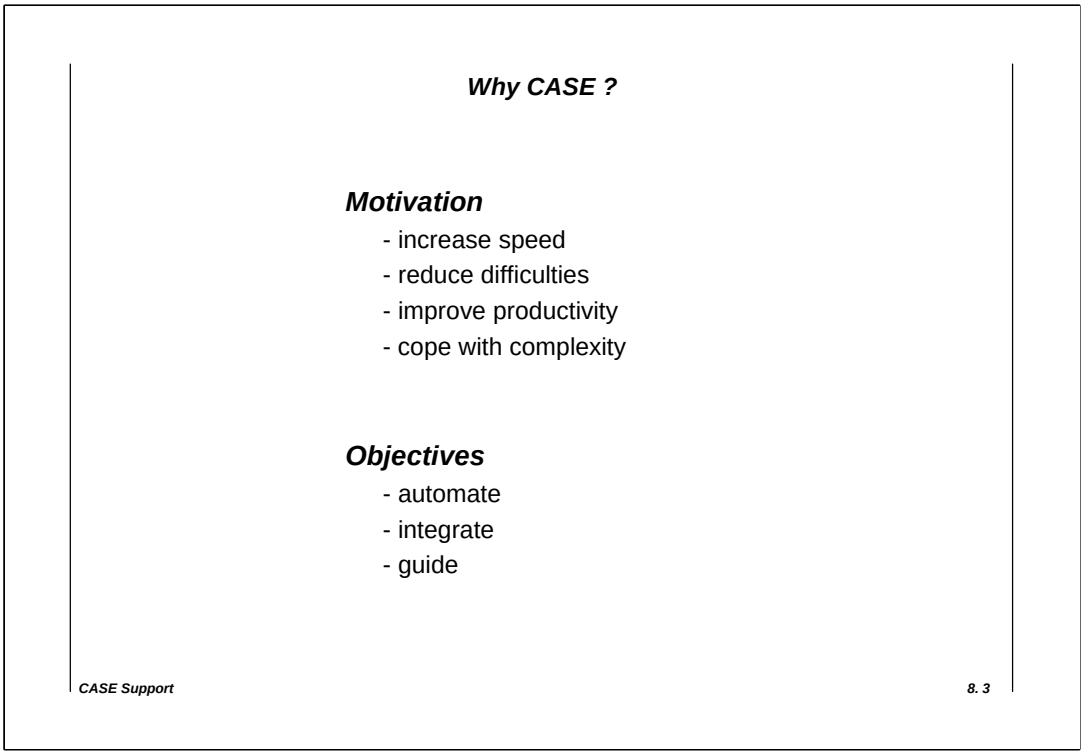
- Object-oriented techniques, as earlier lectures should have shown, are based on a simple principles. However application of these techniques and procedures demands the production, manipulation and storage of large numbers of complex textual and graphical representations. Manual manipulation and management of these representations becomes a daunting task even for small scale projects. The general theme of this lecture is the computer-based support that is available for execution of analysis and design tasks. This requires some understanding of the field generally known as CASE (computer-aided software engineering) and an introduction to at least one of the latest tools. Fortunately the company whose members have drafted the UML also produces a tool for which a demonstration version is freely available, Rational Rose 4.0 (downloadable from http://www.rational.com/demos/rose4.html). Showing this tool also provides an opportunity to say something more about the link between design and code.

> First an overview of the lectures contents

Lecture Overview

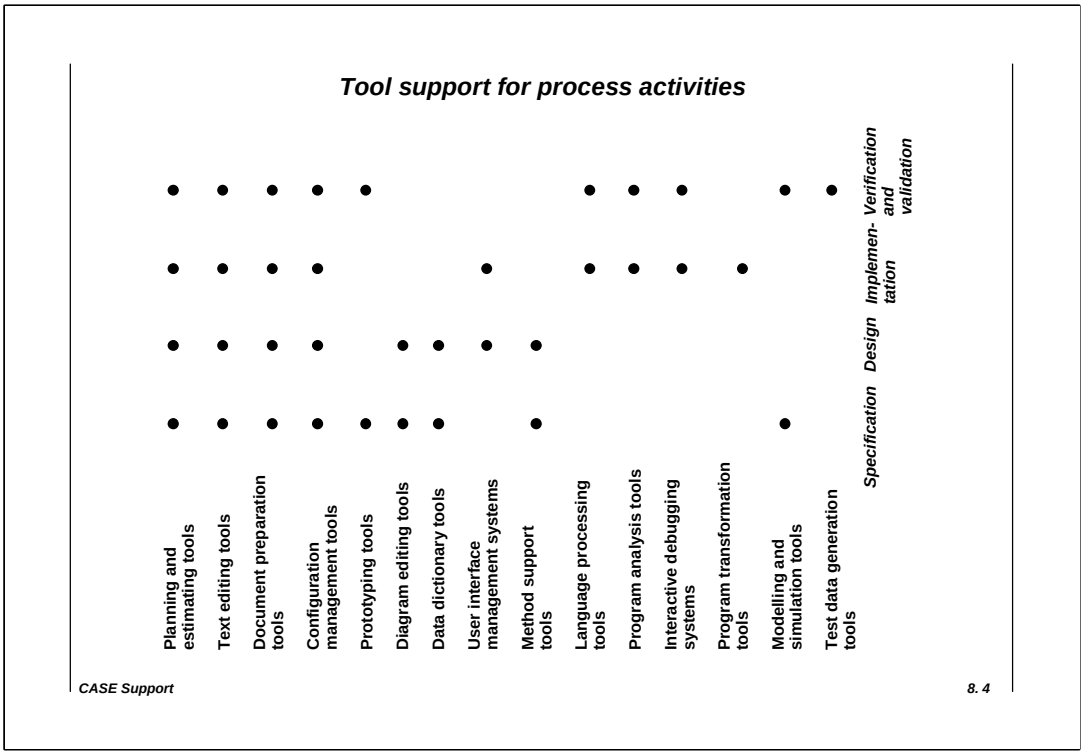
- Motivation and objectives for CASE
- Achievements to date
- Classification of CASE technology
- Sample o-o tools
- A UML-based tool
- Link to code
- *Caveat emptor*

- The general motivation to increase productivity has been behind the development and use of CASE as much as it has been behind the development of all forms of tool. The specific objectives for the use of tools for the production of software relate to their particular field of operation.
- A great number of tools, workbenches, toolsets and environments have come to market, given a variety of names that often make it difficult to identify real capabilities and features.
- In order to make sense of this proliferation Fuggetta has provided a classification of CASE technology that offers a very useful framework for viewing the field.
- In this context we can look at a the capabilities of a couple of tools intended specifically to assist object oriented analysis, and then at the specific functinality and interface of the Rational Rose tool which uses UML , including its code generating facility.
- The final section will point out some of the drawbacks of these tools, in particular those involved in there introduction into the workplace.



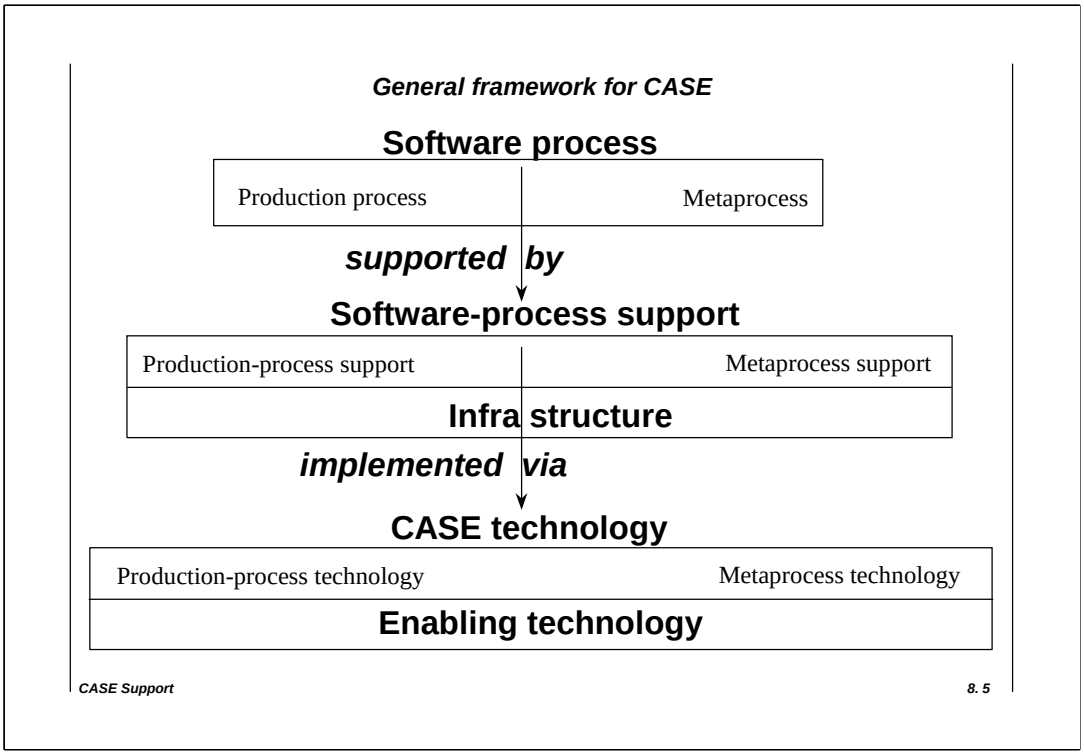
- In any industry there are obvious commercial advantages in being able to offer an actual or potential customer aproduct at the earliest date. In software production reducing the difficulties associated with large or complex products, which were discussed in the first lecture, should increase speed and improve quality. In the context of skilled labour shortages, both factirs should contribute to improved productivity, thereby reducing costs. Another factor often mentioned is flexibility, although this appears to be a desirable bye-product of other factors
- The specific objectives for CASE are often taken as given, but need to be clear. Automation of atomic or composite tasks both increases speed and reduces difficulty. For example, graphical modelling tools can produce composite graphical and textual forms as single elements and also validate their relationships with other instances or types of element. Integration needs to take place between data objects and can be achieved with shared data models, between system components and versions via configuration management mechanisms and between multiple users involved in any single project. It is also necessary, and should be possible, to provide some form of guidance (and control) for the process of production, via the total set of activites, methods, structures and tools that may be used, their order and relationships.

> Increased availabilty of CASE technology has not, however been uniform across activities.



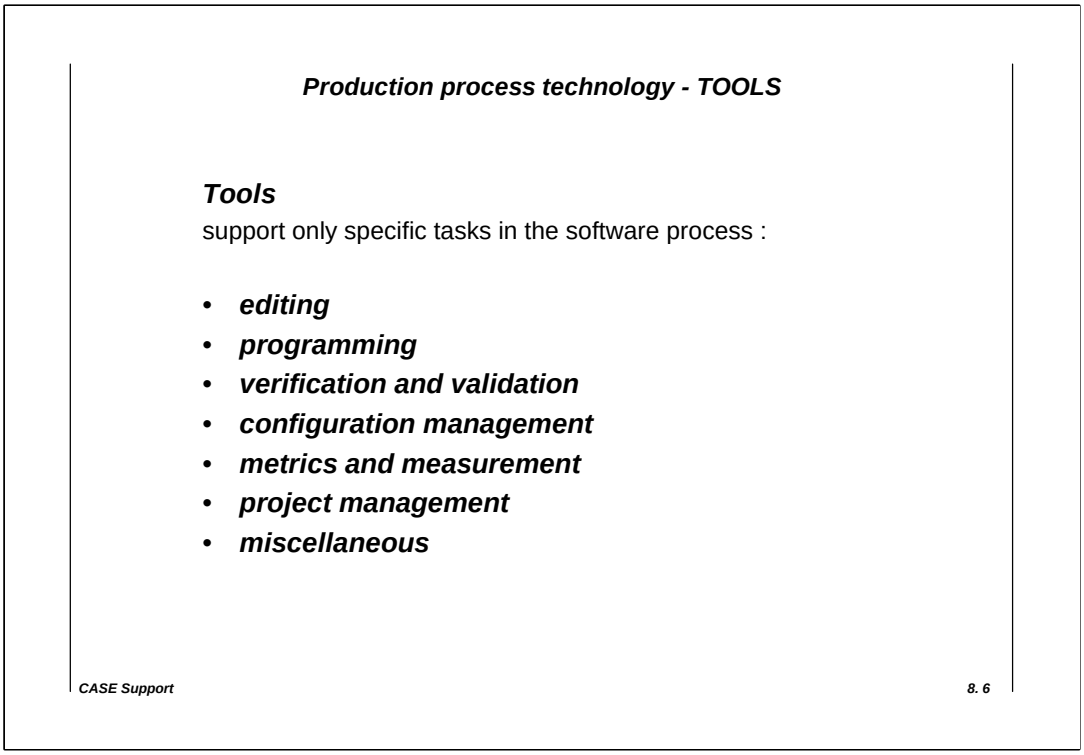
- This figure shows the process activities supported by different classes of CASE tools. It would appear to suggest that there is good coverage of th whole process. However as its originator (Sommerville ‘92) suggests, when you consider this coverage in relation to alternative methods and approaches, it appears much less strong in relation to the more recently expanding fields of object-oriented design and programming.
- In addition this coverage indicates success more in achieving the first objective of automation than the second and third (integration and guidance).
- In order to get a clearer view of what is involved in achieving these objectives, it is worth looking at a structured view of CASE technology.

Sommerville, I. Software Engineering, Fourth Edition, Wokingham: Addison-Wesley, 1992 (in particular Chapters 17 and 18)



- The diagram shows at the top abstract processes and at the bottom concrete technologies. In this view (Fuggetta '93) the software production process (top left) embraces all its relevant activities, techniques, etc. and it evolves via some kind of systematic metaprocess (top right). These processes require support in the form of procedures, rules and technologies, all based on a common infrastructure providing an integrated and homogeneous environmeent. The implementation of this support comes by means of CASE technologies.
- Much of the justification and explanation for CASE tools often comes from the two upper levels, but the lowest level provides the clearest view. In what follows the emphasis is on production process technologies (bottom left), the subject of metaprocess technologies being very much an area of advanced research and experimental development (bottom right).

Fuggetta, A. A Classification of CASE Technology. In IEE Computer, Vol 26 No 12, December 1993 (pp 25-38)



- Editing tools have two large subclasses: textual editors and graphical editors, each including specialised subcategories such as syntax-directed editors.

- Programming tools also include numerous variants in three main groups: coding and debugging, code generators that start from higher level descriptions and code restructurers that can analyse, reformat and sometimes improve source code.

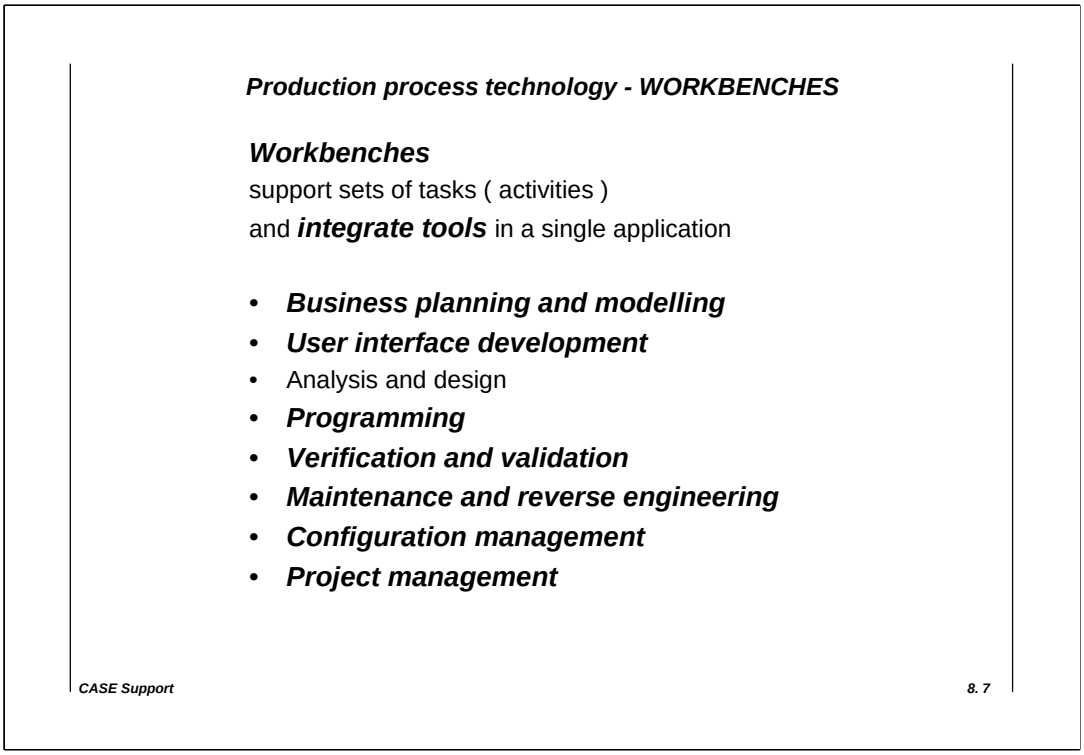
- Validation and verification tools assist getting ‘the right thing’ and ‘the thing right’ , e.g. test case generators.

- Configuration management help to coordinate and control the construction of a system composed of many parts, versions, and individual items.

- Metrics and measurement tools collect data about programs and their execution.

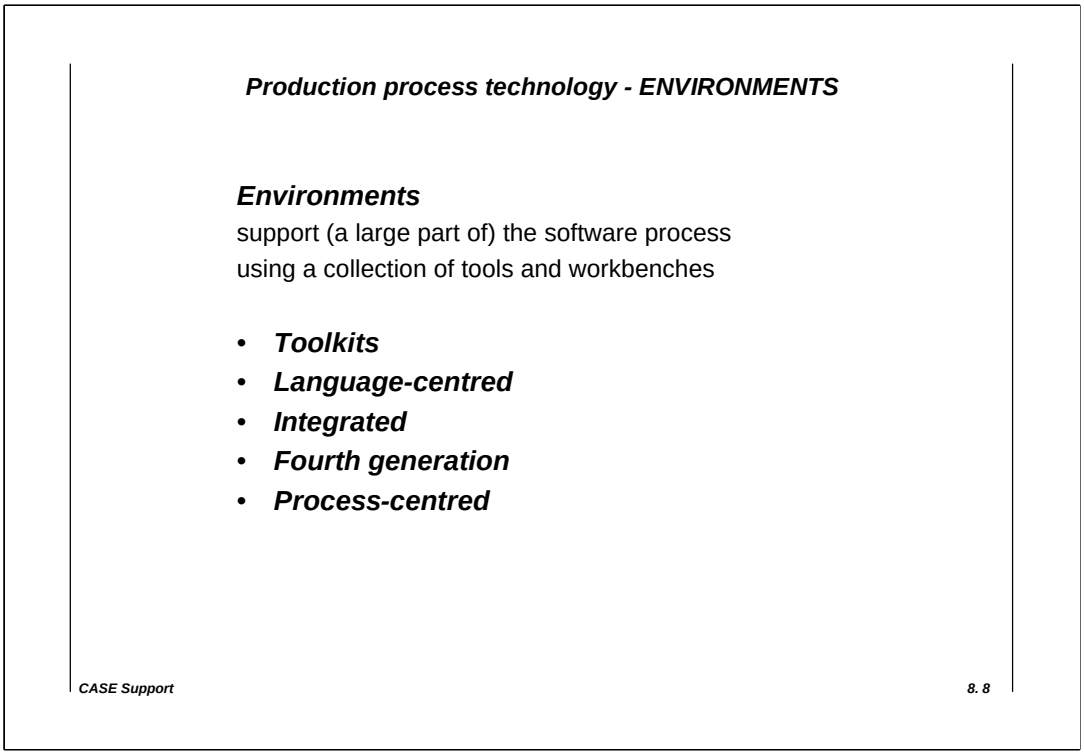
- Project management tools help to estimate costs, to support planning and to aid communication and coordination within project teams.

- Other miscellaneoustools e.g. spreadsheets.



- Whereas tools can be seen as the means of achieving the first identified objective of automation, workbenches serve as the principal means of achieving the second, integration. Integrating tools in a workbench can provide:
a homogeneous and consistent interface (presentation integration), easy invocation of tools and tool chains (control integration), and access to a common data set, managed in a centralised way (data integration). Some products can also enforce predefined procedures and policies (process integration).

> Before considering analysis and design workbenches in more detail, a digression to complete the overall picture with comments about environments.



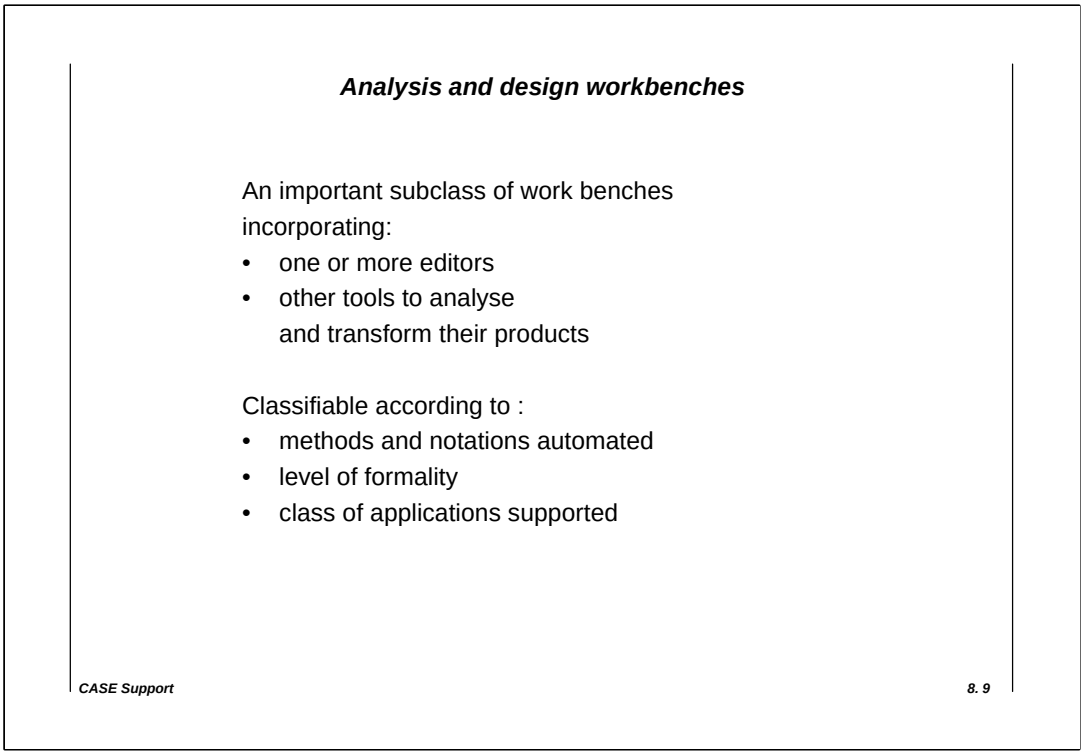
- Toolkits are loosely integrated collections of products easily extended by aggregating different tools and workbenches, e.g. Unix Programmer's Work Bench

- Language-centred environments e.g. Smalltalk (for Smalltalk) offer a good level of presentation and control integration but suffer from a lack of processs and data integration.

- 'Integrated environments' have this name because they operate using standard mechanisms, in particular specialised data repositories that manage all information produced and accessed in the environment, so that users can integrate tools and workbenches.

- 'Fourth generation environments' were precursors to and, in a sense are subclasses of, integrated environments.They often support specific classes of programs, e.g. business orientated applications. Four characteristics are : simple operations in the application but complex information structure, criticality of user interface, importance of prototype development because of imprecise requirements, an evolutionary development process. Macromedia Director is an example.

- Process-centred environments are based on a formal definition of the software process, a process model itself created by specialised tools. A process driver or engine uses this definition to guide development by automatically invoking tools and work benches.

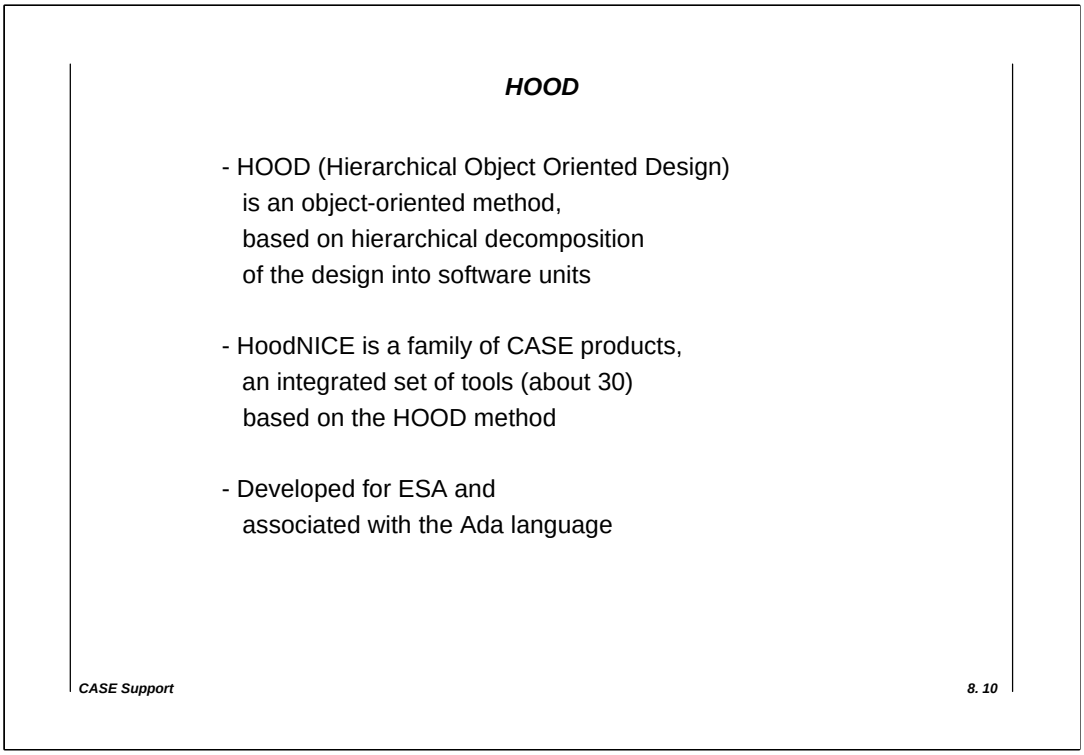


- Workbenches are an important subclass, for which term CASE is often used exclusively, automating most of the analysis and design methods developed that have becomne widely used (structure analysis and design, Jackson System Development, and now object-oriented analysis and design).

- The functionality provided depends heavy on the level of formality of the notation used. The greater the formality the more precise the definition of the syntax and semantics and the greater degree of automation that can be achieved. At the most informal level only text editing and document production are possible; with formal notations, e.g. finite state machines (FSMs) and Petri nets, it is possible to have consistency checks and some automatic generation of elements.

- There is also an important distinction between the uses of these notations and hence of the types of applications that can be supported. Entity relationship diagrams, for example, are most commonly used for data-intensive applications; FSMs are more likely in the analysis and design of control-intensive applications. some workmenches, e.g. Software through Pictures (StP), attempt to support analysis and design with a variety of notations at different levels of formality and for a variety of purposes.

> Now a more detailed look at some examples specifically intended for o-o development

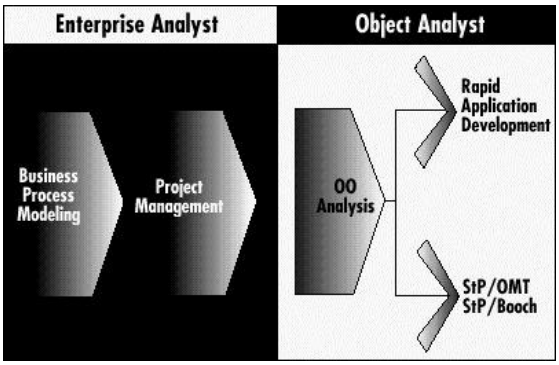


- HoodNICE is an example of a workbench developed explicitly for a particular design method.

The main tools are:

- The HOOD Diagram Editor (HDE) which provides graphical facilities for the architectural design of the application.
- The ODS Editor, which is a syntax directed editor that supports the textual notation of HOOD.
- The Informal Solution Strategy (ISS), which is an informal view of the project.
- Import/Export of the Standard Interchange Format (SIF) representation of the entire design trees and design fragments.
- The Document Generator which generates design documentation according to formats that can be customized by the user for the most popular publishing tools (Ptroff, FrameMaker, TPS, LaTEX,..). Document structure, contents and physical formatting can be customized by the users.
- Off-line checker, which supports time consuming HOOD checks.
- The Code Generator which generates Ada, C or C++ code.
- The Reverse Code Generator which reconciles the design with the code if it has been modified by the user.

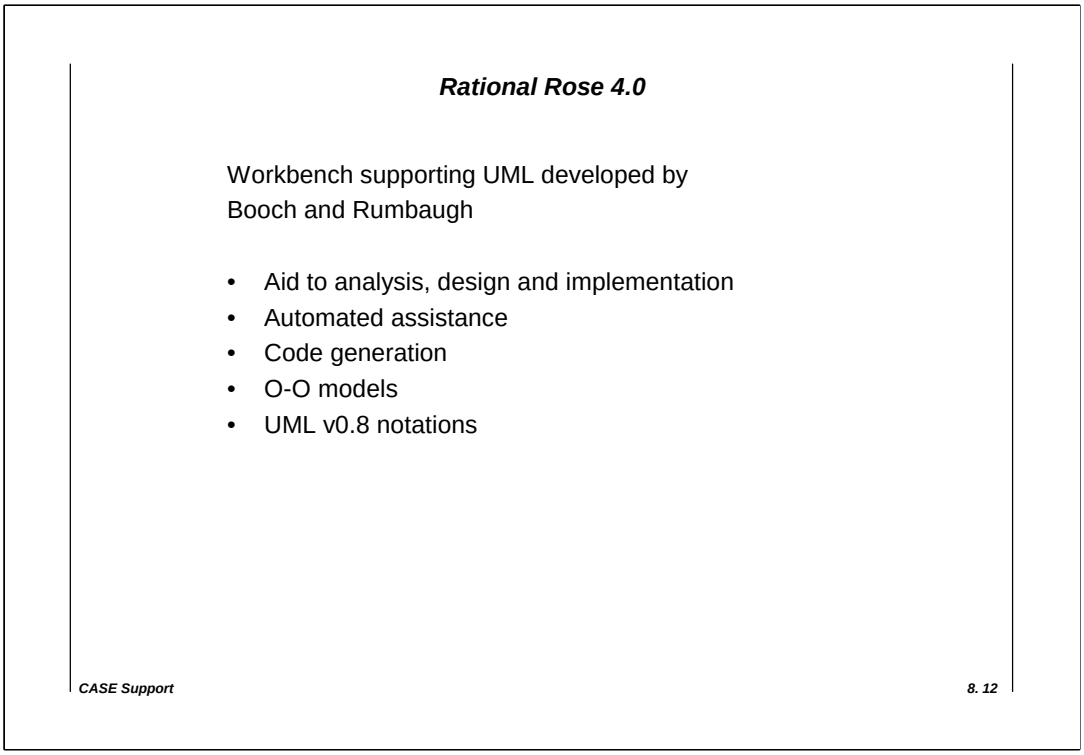
Software through Pictures (StP)



CASE Support *8. 11*

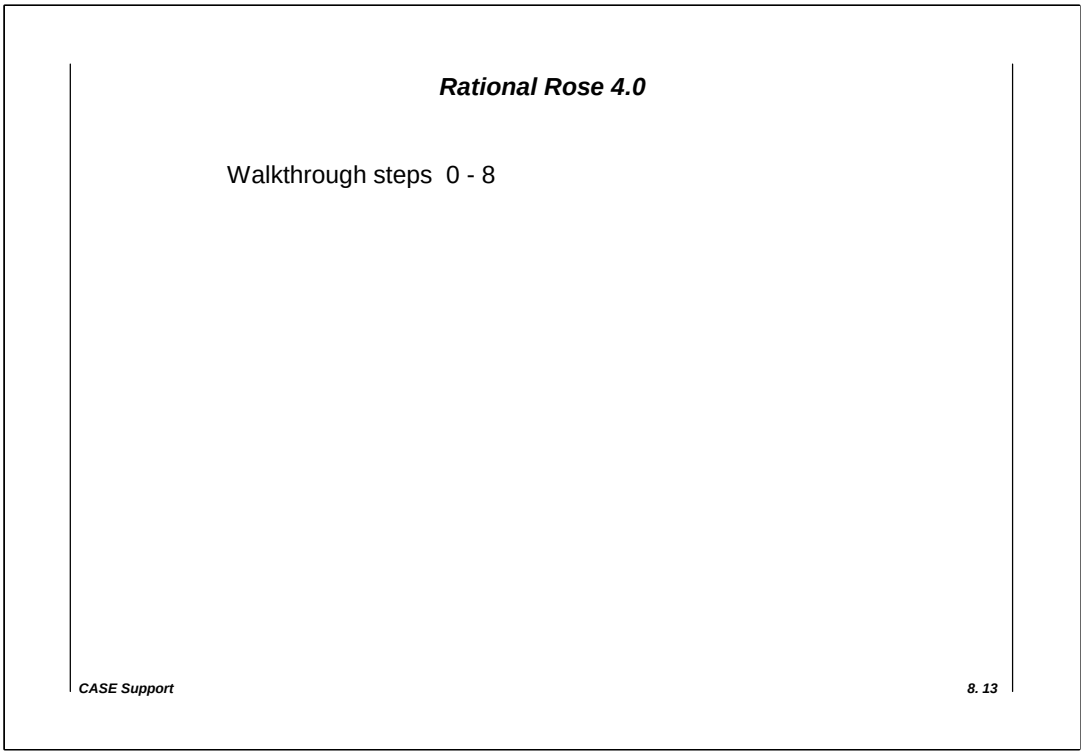
- As this diagram suggests Software through Pictures (StP) is a very ambitious family of tools aiming to help development teams working through all stages of the planning, design and implementation of business systems. Integrated graphical editors and tables support two popular object-oriented methodologies (Booch and Rumbaugh's Object Modeling Technique). Project model locking and access control, checking for model syntax, semantics and completeness, and code and documentation generation are basic features included with all the tools.
- StP/OMT integrates all OMT models into a multi-user environment via 7 editors and an Object Model Browser:

Requirements Table Editor - Captures high level business rules and requirements; Use Case Editor - Performs high level analysis with usage models to identify requirements; Object Model Editor - Expresses classes, their attributes and operations, and the relationship between classes; Dynamic Model Editor - Models class behavior, events and timing; Functional Model Editor - Describes the data transformations associated with a class operation; Object Interaction Editor - Models how objects will interact, what events will invoke the class operations and how interactions are sequenced; Class Table Editor - Provides a textual expression of important class attribute and operation detail not desirable in a graphical model; Object Model Browser - Browses the repository for detailed understanding of class information and system designs.

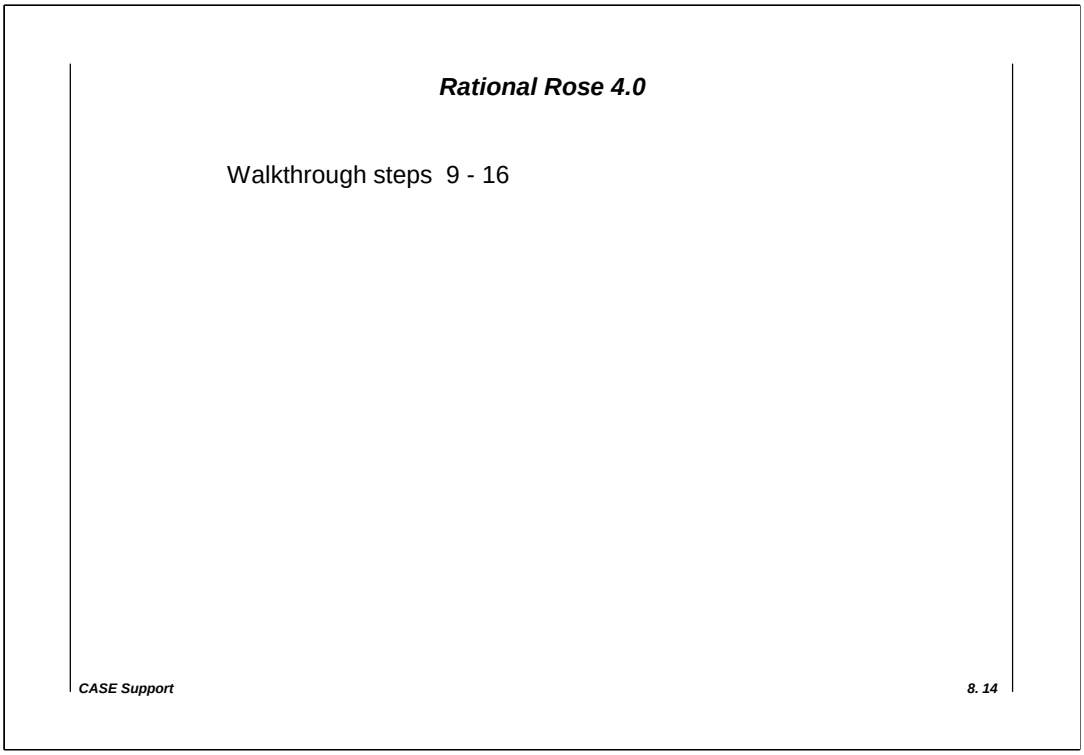


- Rational Rose is an set of tools (a workbench) developed by the company now promoting UML. Its purpose is to support the analysis, design and implementation of object-oriented systems in an integrated way.
- The assistance it provides takes the form of text and graphical tools, most importantly the latter, plus a common database containing the characteristics of all the elements created. There is also a code generation facility which translates these characteristics into a correctly structured and coded (C++) form.
- The workbench does not provide any specific process support, although particular 'views' of the elements created correspond in part to the requirements, design and implementation models in the OOSE method.
- The graphical tools automatically generate representations in UML (although not in the latest v1.0)
and can also be switched automatically to Booch and Rumbaugh modes.
- The 'Demo Walkthrough' provide by Rational gives a good general view of its features.

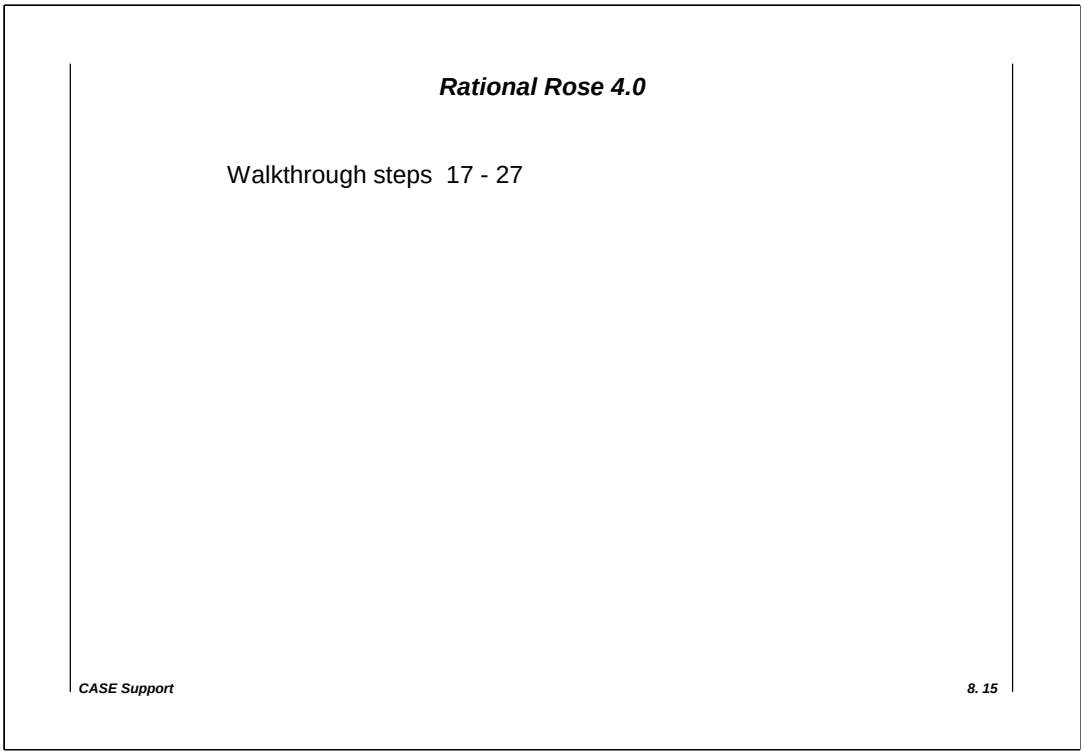
[A demo version of this workbench, which requires Windows 95 or NT to run, is downloadable from http://www.rational.com/demos/rose4.html]



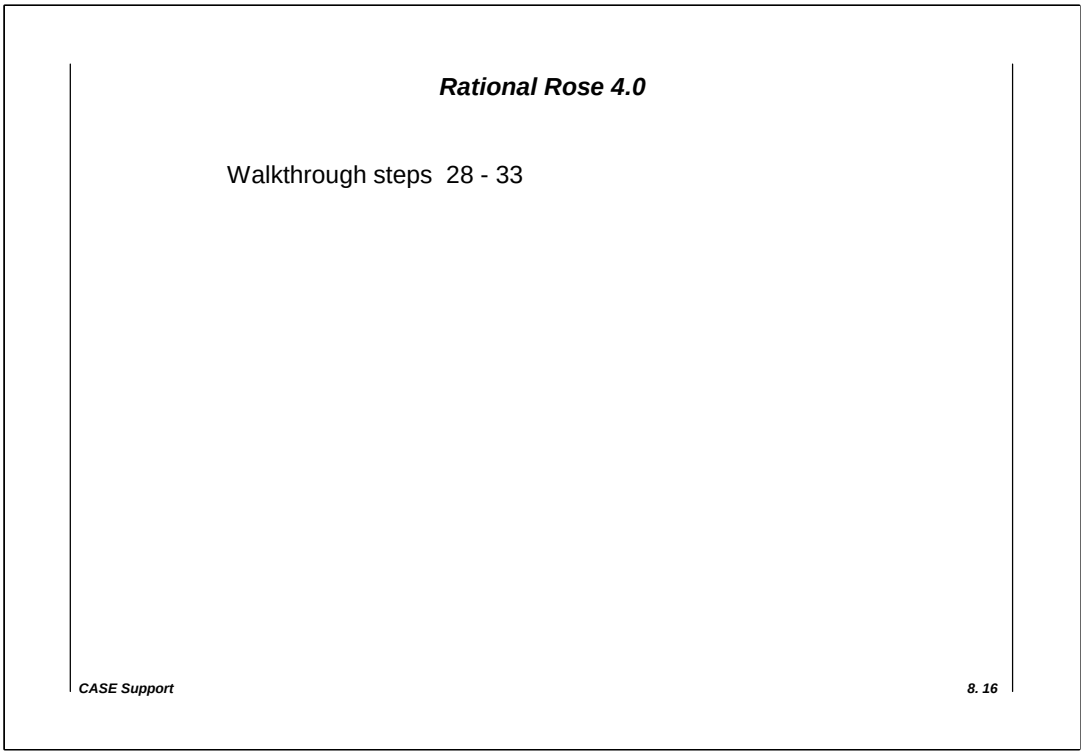
- There are three resizable windows, but only the ‘Browser’ (lower left) can be used for creating new elements in the database, which can then be combined and amended in different diagrams in the window on the right. It is possible to ‘sketch’ on the right, but this does not create new elements in the scope of the Browser.
- Although the documentation window can contain text associated with any element, there is no suggestion that scenarios should precede the generalised actors and use cases.



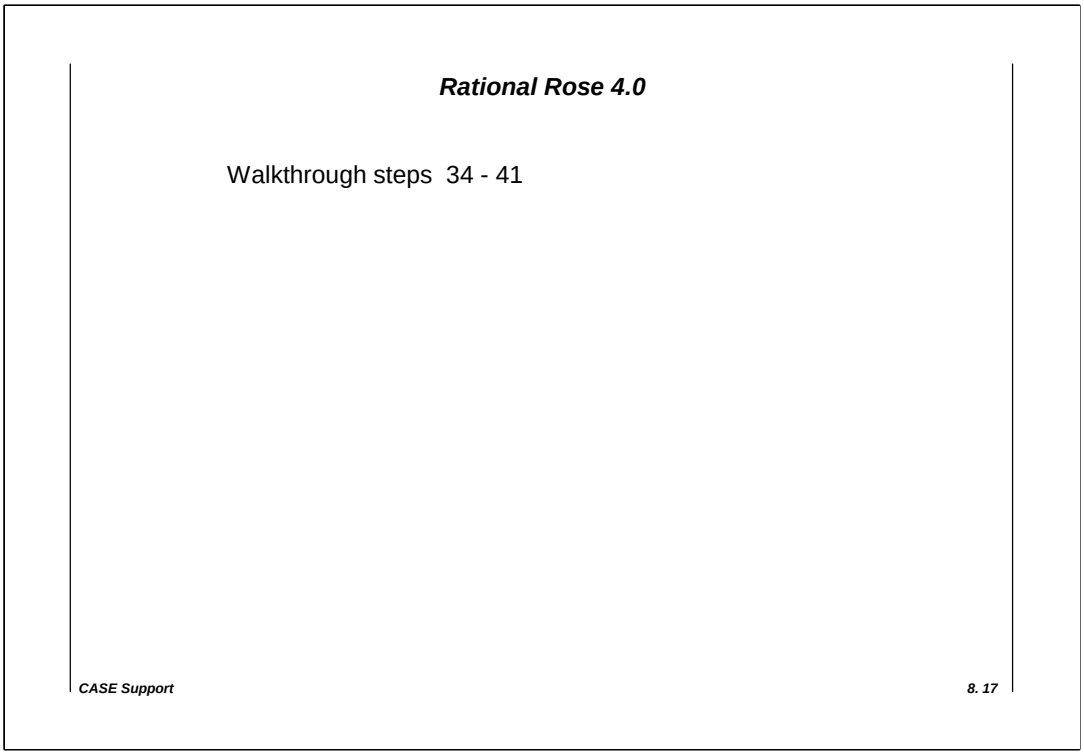
- These steps show the creation of a standard UML-type sequence diagram, but the objects are created independent of any analysis of the problem domain objects suggested by a particular case.
- Note also that sequence diagrams form part of the design model in OOSE and no special support is provided for the analysis phase.



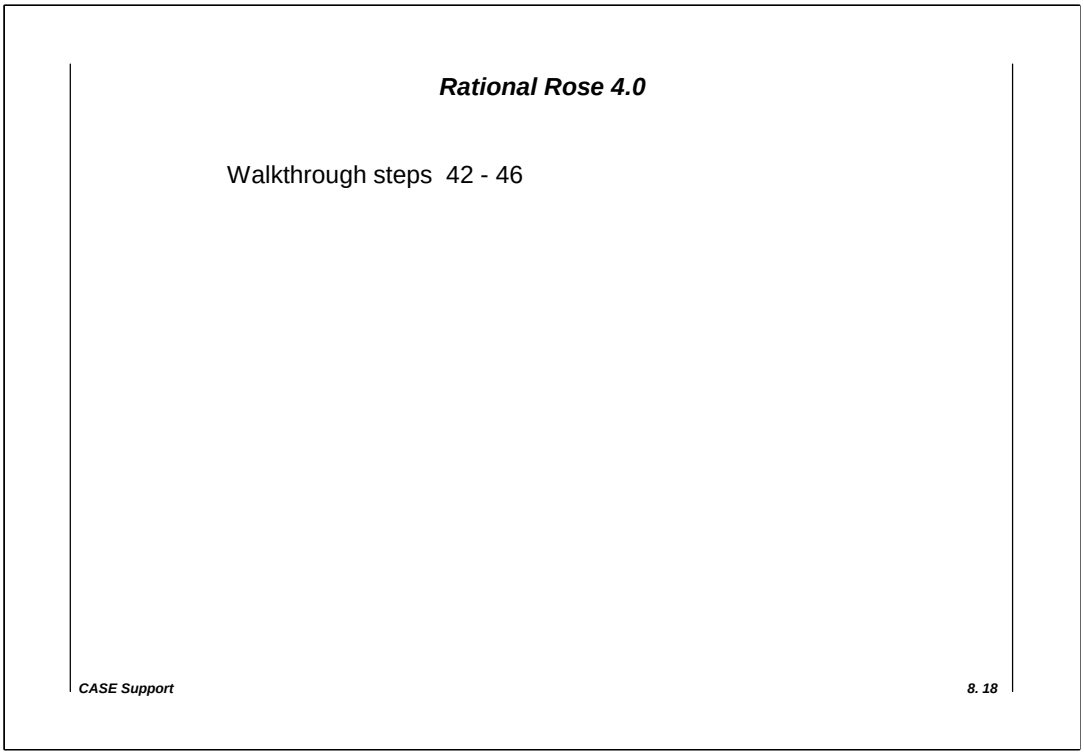
- The ‘logical view of the model’ corresponds to the design model in OOSE.
- Note that in 17 you create classes in the Browser which in 23 are ‘dragged onto’ the appropriate objects in the sequence diagram, thus giving them a class identity automatically.



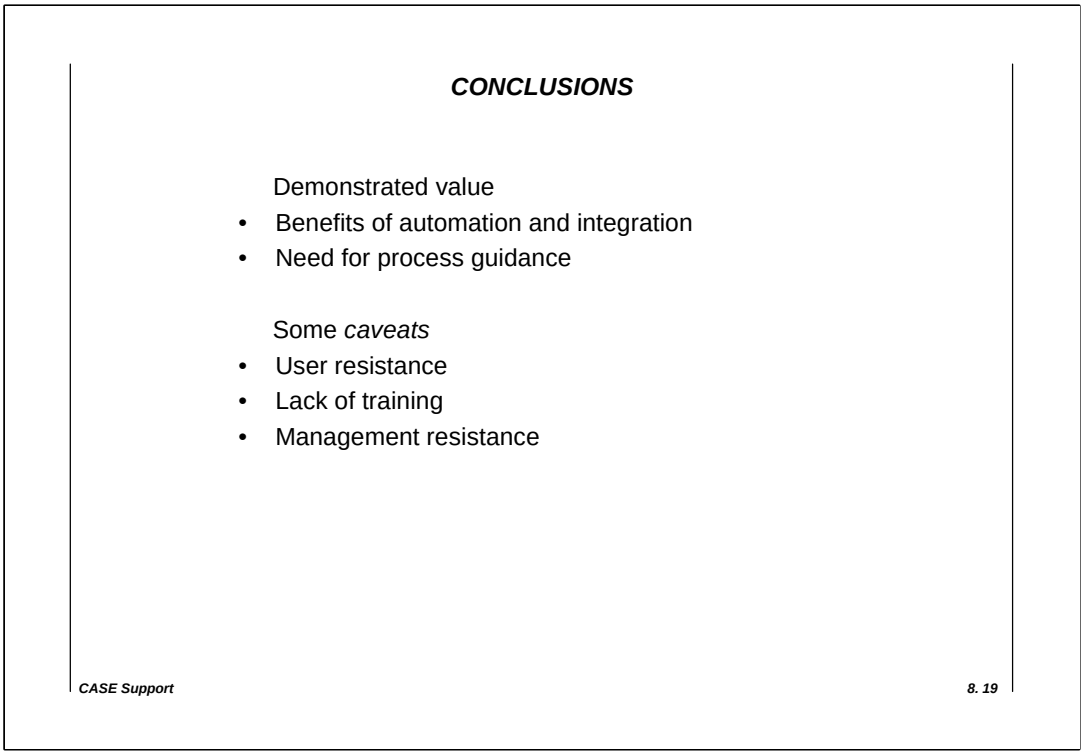
- This section deals principally with associations and at the end with inheritance.
- At step 33 the demo version reaches its maximum number of saveable elements.



- This section is concerned with the creation of logical packages and the mapping to sub-systems (physical packages) created in the Component view.
- Note in 36 that scenarios/sequence diagrams can be updated, which would be forbidden in OOSE given the fixed nature of the analysis model and of the use case descriptions contained in it, and the likely prejudice to the contents of the requirements model.



- This section demonstrates the code generating capability, translating the defined structure and its characteristics into compilable C++ code.



- Rational Rose illustrates the benefits of automation and integration that have been achieved using CASE technology. It also shows the problems that might arise if it were to be used independent of any coherent method.
- There are also a number of general *caveats* regarding the introduction of CASE, and particularly comprehensive environments, sometimes called SDE or SEE. These provide considerably increased management control over the process and conversely may reduce individual creativity and satisfaction. There may also be a need for a considerable investment in training which, if it is not made, will reduce the value of both staff and investment. The cost advantages of what may be a large investment are unlikely to be quantifiable, leading to management reluctance to introduce the latest technology and techniques.