Implementation

In this Lecture you will Learn:

- About tools used in software implementation
- How to draw component diagrams
- How to draw deployment diagrams
- The tasks involved in testing a system
- How to plan for data conversion from an old system
- Ways of introducing a new system into an organization
- Tasks in review and maintenance

Software Implementation Tools

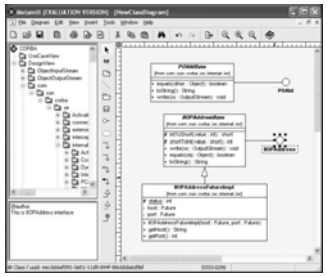
- In an iterative project, planning for implementation will begin in the inception phase
- The implementation workflow includes tasks to set up the environment for implementation
- Some tools, particularly CASE tools and configuration management systems will carry through from analysis and design activities
- A wide range of types of software tools will be used

Software Implementation Tools

- Different categories of software that may be used in developing:
  - CASE Tools
  - Compilers, Interpreters and Run-times
  - Visual Editors
  - IDE (Integrated Development Environment)
  - Configuration Management Tools
  - Class Browsers
  - Component Managers
  - DBMS (Database Management Systems)
  - CORBA
  - Testing Tools
  - Installation Tools
  - Conversion Tools
  - Documentation Generators

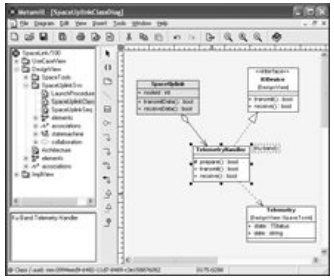
Software Implementation Tools: CASE Tools

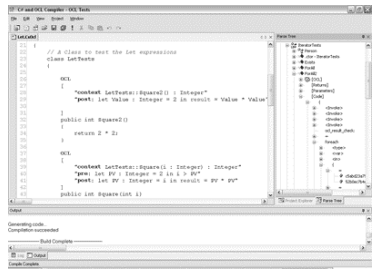
- Many tools now support UML
- Make it possible to generate code from the models
- May make reverse engineering of code possible, to provide round-trip engineering
- May map classes to a relational database
- Link to configuration management tools



Software Implementation Tools: Compilers, Interpreters and Run-times

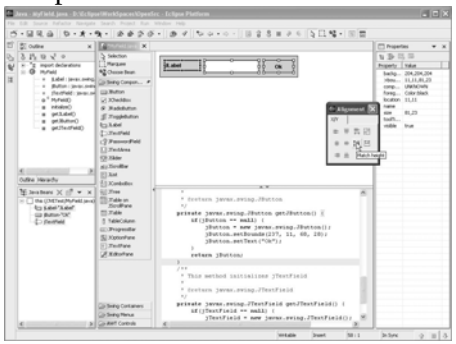
- Different languages require different tools
- C++ requires a compiler and a linker to build executables
- Java requires a compiler and a run-time program and libraries to run the byte-code produced by the compiler
- C# is like Java and is compiled into MSIL (Microsoft Intermediate Language)





Software Implementation Tools: Visual Editors

- Provide a way of designing GUI interfaces by dragging and dropping buttons, text fields etc. onto a window
- Can often also handle controls or objects that represent non-visual components such as links to a database or communications processes

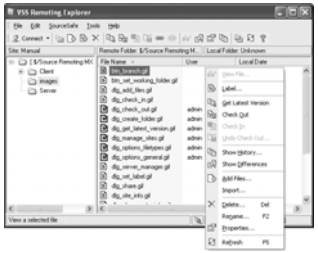


Software Implementation Tools: IDE (Integrated Development Environment)

- Manage the many files in a project and the dependencies among them
- Link to configuration management tools
- Use compilers to build the project, only recompiling what has changed
- Provide debugging facilities
- May include a visual editor
- Can be configured to link in third party tools

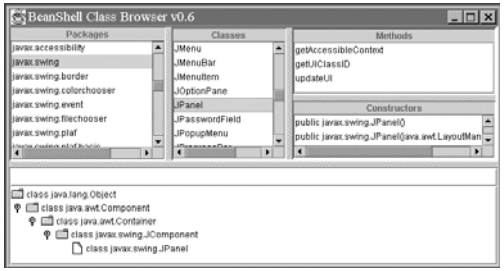
Software Implementation Tools: Configuration Management Tools

- Also called **Version Control Tools**, although configuration management is more than just version control
- Maintain a record of file versions and the changes from one version to the next
- Record all the versions of software and tools that are required to produce a repeatable software build



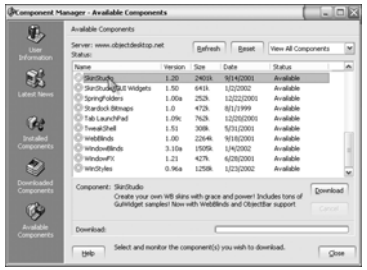
Software Implementation Tools: Class Browsers

- May be part of IDE or visual editors
- Originally provided as the way of browsing through available classes in Smalltalk
- Java API documentation is provided in a browseable hypertext format generated by Javadoc



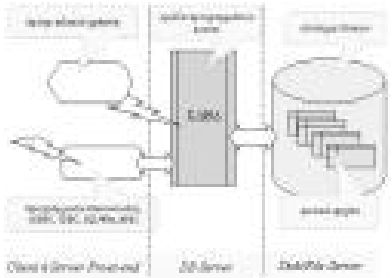
Software Implementation Tools: Component Managers

- New kind of tool to manage components
- Provide mechanisms to
  - Add components
  - Search for components
  - Browse for components
  - Maintain versions of components



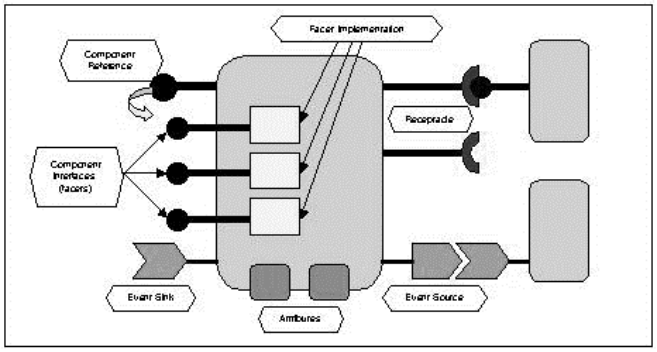
Software Implementation Tools: DBMS (Database Management Systems)

- Server system
- Client software (administration interfaces, ODBC and JDBC drivers)
- Tools to manage the database and carry out performance tuning
- Large DBMS, such as Oracle, come with many tools, even their own application server



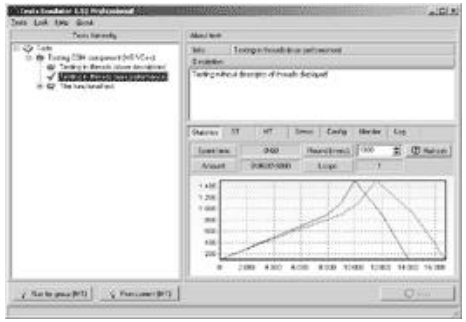
Software Implementation Tools: CORBA

- CORBA ORB to handle the marshalling and un-marshalling of requests and objects
- IDL compiler
- Registry service



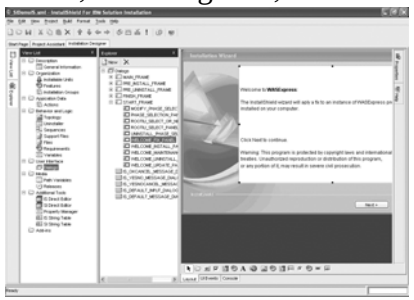
Software Implementation Tools: Testing Tools

- Tools written by developers as test harnesses
- Automated test tools to run repeated or multiple simultaneous tests
- May allow user to run through test once manually, then generate a script that can be edited to provide variations



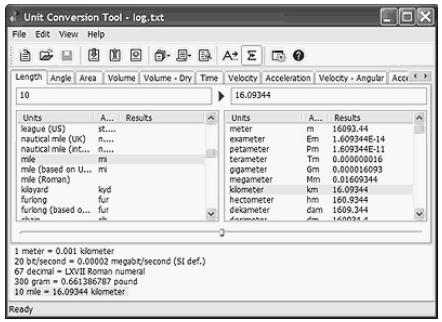
Software Implementation Tools: Installation Tools

- Automate the extraction of files from an archive and the setting up of configuration files and registry entries
- Some maintain information about dependencies on other pieces of software and will install all necessary packages (e.g. Redhat RPM)
- Uninstall software, removing files, directories and registry entries



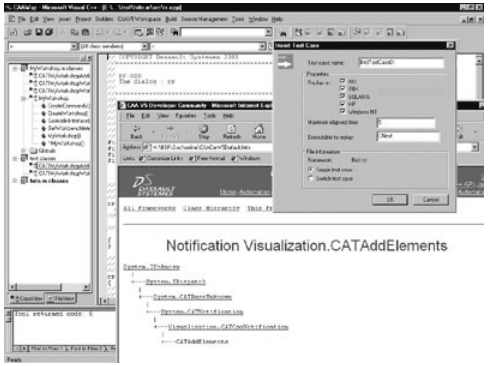
Software Implementation Tools: Conversion Tools

- Extract data from existing systems
- Reformat the data for the new system
- Insert it into the database for the new system
- May require manual intervention to 'clean up' the data – removing duplication or invalid values in fields



Software Implementation Tools: Documentation Generators

- Document models and code
- Extract standard information or user-defined information into document templates
- Produce HTML to document the API of classes in the application



Coding and Documentation Standards: Object-oriented Standard

- Naming standards are agreed early in a project
- A typical object-oriented standard:
  - Classes with capital letters: Campaign
  - Attributes and operations with initial lower case letters: title, recordPayment()
  - Words are concatenated together with capital letters to show where they are joined: InternationalCampaign, campaignFinishDate, getNotes()

Coding and Documentation Standards: Names Prefix

- Hungarian Notation
- Used in C and C++
- Names prefixed by an abbreviation to show the type of the member variable
  - **b** for boolean: bOrderClosed
  - **i** for integer: iOrderLineNumber
  - **btn** for button: btnCloseOrder

Coding and Documentation Standards: Underscores

- Using underscores to separate parts of a name instead of capital letters
  - Order_Closed
- Often used for column names in databases, as it is easier to replace the underscores with spaces to produce meaningful column headings in reports than trying to find the word breaks in concatenated names

Coding and Documentation Standards: Document code

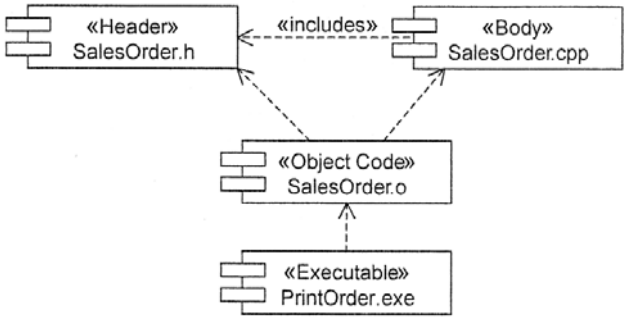
- Think of the people who will maintain your code
- Others may be able to use your code to learn good practice, but only if it is clearly documented
- No language is self-documenting; conventions and standards help
- Comply with Java documentation standards, if coding in Java (Javadoc)
- You can take advantage of tools that automate the production of documentation from comments

Implementation Diagrams

- Component Diagrams
  - Used to document dependencies between components, typically files, either compilation dependencies or run-time dependencies
- Deployment Diagrams
  - Used to show the configuration of run-time processing elements and the software components and processes that are located on them

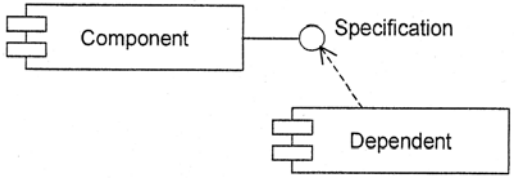
Notation of Component Diagrams

- Rectangles with two small rectangles superimposed at one end
- May implement interfaces, shown as circles connected by a line
- Can be stereotyped, for example to represent files

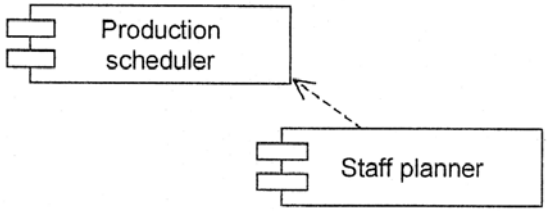


Example of Component Diagrams

- Dependency of a component on the interface of another component

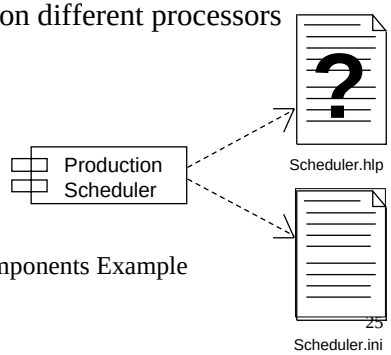


- Dependency between high-level components



Components

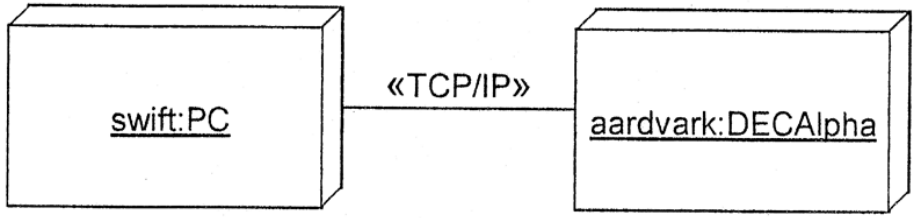
- Components should be physical components of a system
- Packages can be used to manage the grouping of physical components into sub-systems
- Components can be shown on deployment diagrams to document their deployment on different processors



Stereotyped Components Example

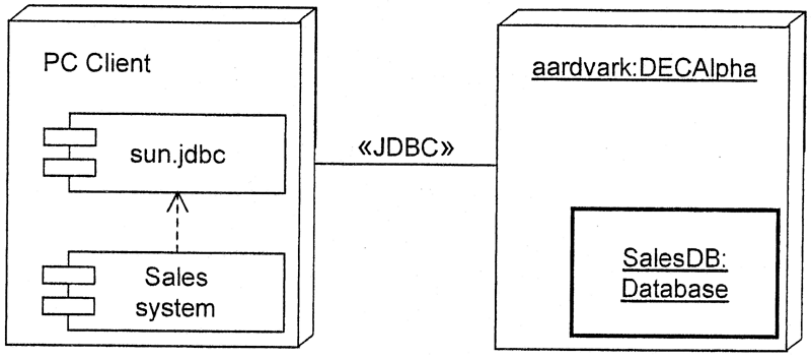
Notation of Deployment Diagrams

- Nodes
  - Rectangular prisms
  - Represent processors, devices or other resources
- Communication Associations
  - Lines between nodes
  - Represent communication between nodes
  - Can be stereotyped



Notation of Deployment Diagrams

- Can be shown with active objects or components located on the nodes
- In a component diagram, components are component types, in a deployment diagram, they are component instances



Package Diagram vs. Component Diagram

- Package diagrams show the *logical grouping of classes* in a system, whereas component diagrams show the *physical components* of a system.
- During implementation, package diagrams can be used to *show the grouping of physical components into sub-systems*; component diagrams can be combined with deployment diagrams to *show the physical location of components of the system*.

Use of Implementation Diagrams

- Can be used architecturally to show how elements of system will work together
- Typically used for simple diagrams
- Full documentation of dependencies and location of all components may be better handled by configuration management software or in a spreadsheet or database

Software Testing

- Who tests?
  - Ideally specialist test teams with access to software to build and execute test scripts
  - Often the analysts who have carried out the initial requirements gathering and analysis
  - In **eXtreme Programming** (XP) programmers are expected to write test harnesses for classes before they write the code
  - Users of the system, who will test against requirements and do user acceptance testing

Software Testing

- What kinds of tests?
  - Black box testing
    - Does it do what it's meant to do?
    - Does it do it as fast as it should?
  - White box testing
    - Is it not just a solution to the problem, but a good solution?

Purpose of Testing

- The purpose of testing is to try find errors, not to prove the software is correct
  - Test data should test the software at its limits and test business rules
    - Extreme values (very large numbers, long strings)
    - Borderline values (0, -1, 0.999)
    - Invalid combinations of values (age = 3, marital status = married)
    - Nonsensical values (negative order line quantities)
    - Heavy loads (are performance requirements met?)

Levels of Testing

- **Unit Testing** (Individual classes)
- **Integration Testing** (Classes work correctly together)
- **Sub-system Testing** (Sub-system works correctly and delivers required functionality)
- **System Testing** (Whole system works together with no unwanted interaction between sub-systems)
- **Acceptance Testing** (System works as required by the users and according to specification)

Outside-in or Inside-out?

- Test Harnesses
  - Write programs that create instances of classes and send them messages to test operations execute correctly
- Mock objects (Endo-testing)
  - Write mock objects that implement the interface of real objects and check the signature of messages sent to them and the state of the objects

Levels of Testing

- Level 1
  - Test modules (classes), then programs (use cases) then suites (application)
- Level 2 (Alpha Testing or Verification)
  - Execute programs in a simulated environment and test inputs and outputs
- Level 3 (Beta Testing or Validation)
  - Test in a live user environment and test for response times, performance under load and recovery from failure

Test Documentation: Test Plans

- Written before the tests are carried out!
- Written, in fact, before the code is written
- Contains Test Cases
  - Description of test
  - Test environment and configuration
  - Test data
  - Expected outcomes

Test Documentation: Test Results

- Ideally in a spreadsheet or database
- Record when tests are failed or passed
- Allow reporting of percentage passed
- Ideally should be linked to requirements
- Error results should be recorded in a fault reporting package with enough details for developers to reproduce them with a view to fixing the bugs

Data Conversion for Manual System

- Data from manual systems needs collating and putting into a standard format
- Incomplete data may need to be chased up
- There is a cost associated with keying data into a new system
- There may be a requirement for data entry screens that will only be used to get data into the system to start it up

Data Conversion for Existing System

- Existing data must be checked for correctness
- Specially written programs may be required to check and convert the data
- Data may be loaded into a staging area to be ‘cleaned up’
- Data is imported into the new system
- Data must be verified after being imported

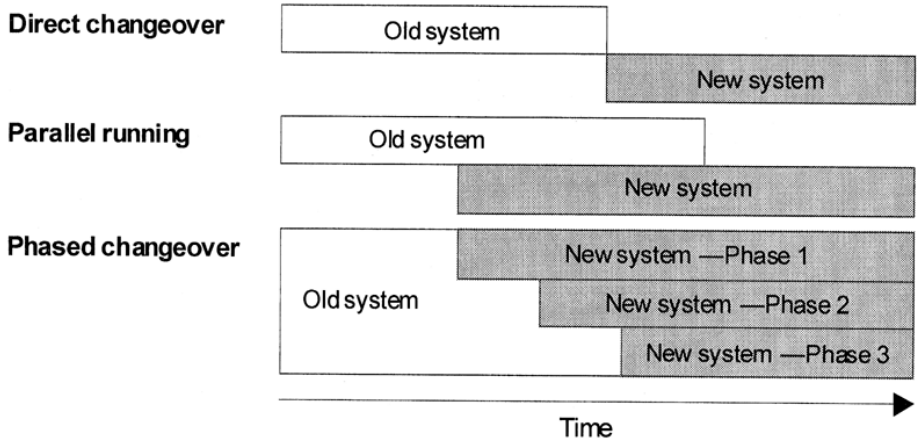
User Documentation

- Training manuals organized around the tasks the users carry out
- On-line Computer-based Training (CBT) that can be delivered when the users need it
- Reference manuals to provide complete description of the system in terms the users can understand
- On-line help replicating the manuals

User Training

- Set clear learning objectives for trainees
- Training should be practical and geared to the tasks the users will carry out
- Training should be delivered ‘just in time’ not weeks before the users need it
- CBT can deliver ‘just in time’ training
- Follow up after the introduction of the system to make sure users haven’t got into bad habits through lack of training or having forgotten what they had been told

Implementation Strategies



Implementation Strategies: Direct Changeover

- On a date the old system stops and the new system starts
  - + brings immediate benefits
  - + forces users to use the new system
  - + simple to plan
  - – no fallback if problems occur
  - – contingency plans required for the unexpected
  - – the plan must work without difficulties
- Suitable for small-scale, low-risk systems

Implementation Strategies: Parallel Running

- Old system runs alongside the new system for a period of time
  - + provides fallback if there are problems
  - + outputs of the two systems can be compared, so testing continues into the live environment
  - – high running cost including staff for dual data entry
  - – cost associated with comparing outputs of two systems
  - – users may not be committed to the new system
- Suitable for business-critical, high-risk systems

Implementation Strategies: Phased Changeover

- The new system is introduced in stages, department by department or geographically
  - + attention can be paid to each sub-system in turn
  - + sub-systems with a high return on investment can be introduced first
  - + thorough testing of each stage as it is introduced
  - – if there are problems rumours can spread ahead of the implementation
  - – there can be a long wait for benefits from later stages
- Suitable for large systems with independent sub-systems

Implementation Strategies: Pilot Project

- Complete system is tried out in one department or at one site
  - + can be used as a learning experience
  - + can feed back into design before system is launched organization-wide
  - + decision on whether to go ahead across the whole organization can depend on the pilot outcome
  - + reduces risk
  - – there is an initial cost without benefits across the whole organization
- Suitable for smaller systems and packaged software

Review

- Post-implementation review
- Review the system
  - whether it is delivering the benefits expected
  - whether it meets the requirements
- Review the development project
  - record lessons learned
  - use actual time spent on project to improve estimating process
- Plan actions for any maintenance or enhancements

Evaluation Report

- **Cost Benefit Analysis** – Has it delivered? Compare actual with projections
- **Functional Requirements** – Have they been met? Any further work needed?
- **Non-functional Requirements** – Assess whether measurable objectives have been met
- **User Satisfaction** – Quantitative and qualitative assessments of satisfaction with the product
- **Problems and Issues** – Problems during the project and solutions so lessons can be learned

Evaluation Report (cont')

- Positive Experiences – What went well? Who deserves credit?
- **Quantitative Data for Planning** – How close were time estimates to actual? How can we use this data?
- **Candidate Components for Reuse** – Are there components that could be reused in other projects in the future?
- **Future Developments** – Were requirements left out of the project due to time pressure? When should they be developed?
- **Actions** – Summary list of actions, responsibilities and deadlines

Maintenance Activities

- Systems need maintaining after they have gone live
- Bugs will appear and need fixing
- Enhancements to the system may be requested
- Maintenance needs to be controlled so that bugs are not introduced and unnecessary changes are not made
- Helpdesk, operations and support staff need training to take on these tasks
- A Change Control System is required to manage requests for bug fixes and enhancements
- Changes need to be evaluated for their cost and their impact on other parts of the system, and then planned

Maintenance Documentation

- Bug reporting database
- Requests for enhancements
- Feedback to users
- Implementation plans for changes
- Updated technical and user documentation
- Records of changes made