Chapter 1 : Acta Informatica

DIGITAL's VAX Ada is a validated, production-quality implementation of the full Ada language that is well-integrated into the VMS TM operating system environment on VAX systems. The programming support environment consists of an Ada compiler, an Ada program library manager, and a multi-language.

Further, developers may customize and extend the rapid application development features to suit their development needs in a particular case. This related patent application is herein incorporated by reference in its entirety. Field of the Invention Embodiments of the invention are related to data processing and more particularly to tools used to develop application software. Description of the Related Art Developing software applications is a complex task, and IDE tools are available to assist computer programmers with the development process. Currently, IDE tools are available to assist programmers developing applications in a variety of programming languages e. These tools are typically configured with features such as auto-indenting, syntax highlighting, type checking, and a variety of other features that assist the development process. Typically, IDE tools are optimized for different programming languages. One common feature provided by IDE tools is the capability to generate application source code or other project artifacts for a development project e. More specifically, currently available IDE tools may be configured to auto-generate pre-defined types of application source code or artifacts based on a pre-defined set of inputs. Similarly, an IDE tool may be configured to create a framework for a web-service or some other application type. In such a case, however, the IDE tool may only be configured to build a generic web-service framework or in some cases a web-service framework localized for a particular application server. One problem with this approach is that developers wish to create an application package that may be used on a variety of commercially available application servers. Thus, even when the IDE tool generates a comprehensive web-service for a given application server, it locks the developer into that application server. To use others, the developer has to either develop multiple versions of their web-service by hand or attempt to modify the one generated by the IDE tool, an ad-hoc and error-prone process at best. In other words, it is left to the developer to build most of the application, and at best the available IDE tools may provide various options in the form of user preferences that allow developers to customize what is created by the IDE tool. And even this still requires the developer to have knowledge of the type and extent of customization that is offered by the IDE tool. Importantly, for any new user customizations, IDE tool builders have to release new versions of the IDE tool, developers have to wait and usually pay for changes in what code-generation features are included in a given IDE tool. Accordingly, there remains a need in the art for an IDE tool that provides for extensible rapid application development for disparate data sources. The method generally includes receiving, from a user interacting with an integrated development environment IDE tool, a selection of an input resource, parsing the input resource to identify a resource type of the input resource, and generating a collection of template-input data from the input resource based on the resource type and content of the input resource. The method also includes identifying a template corresponding to the resource type, wherein the template specifies an output resource type that may be generated from the template, generating content of the output resource, based on the template and the template-input data, and storing the content of the output resource within a project file associated with the development project. Another embodiment of the invention includes a computer program product comprising a computer useable medium having a computer readable program, wherein the computer readable program, when executed on a computer causes the computer to perform an operation for generating application source code or development project artifacts for a software development project. The operation generally includes receiving a selection of an input resource, parsing the input resource to identify a resource type of the input resource, and generating a collection of template-input data from the input resource, based on the resource type and content of the input resource, The operation also includes identifying a template corresponding to the resource type, wherein the template specifies an output resource type that may be generated from the template, generating content of the output resource, based on the template and the template-input data, and storing the content of the output resource within a project file associated with the development project. Still another embodiment of the invention includes a system having a processor and a memory containing an integrated development environment IDE tool, which when executed on the processor, is configured to generate application source code or development project artifacts for a software development project by performing an operation. The operation may generally include, receiving, from a user interacting with the IDE tool, a selection of an input resource, parsing the input resource to identify a resource type of the input resource, and generating a collection of template-input data from the input resource, based on the resource type and content of the input resource. The operation may generally further include identifying a template corresponding to the resource type, wherein the template specifies an output resource type that may be generated from the template, generating content of the output resource, based on the template and the template-input data, and storing the content of the output resource within a project file associated with the development project. Still another embodiment of the invention provides a method for generating a generic service configuration template used to create a collection of localized configuration files for a distributed application. The method generally includes receiving a set of service parameters associated with the distributed application, generating, based on the set of service parameters, a generic service configuration template, and storing the generic service configuration template. The generic service configuration template may include one or more extensible style language transforms XSLTs used to generate a collection of localized configuration files for a given application server. It is to be noted, however, that the appended drawings illustrate only typical embodiments of this invention and are therefore not to be considered limiting of its scope, for the invention may admit to other equally effective embodiments. Developers may customize and extend the rapid application development templates to suit their development needs in a particular case. In one embodiment, the developer may use a variety of input resources as metadata to generate application source code or other artifacts for an application development project. What content actually gets generated depends on the input resources as well as a rapid application development template used to process the input metadata. By customizing the template to process different input resources and customizing what application source code or other artifacts may be generated for a given set of input resources the resulting application source code may be built from and for heterogeneous and disparate data sources in a seamless fashion. Further, developers do not need to rely on upgrades to the IDE tool in order to change what application source code or other development artifacts are generated from a given input resource. Thus, embodiments of the invention provides an efficient and flexible framework for generating application source code that is fully customizable, greatly reducing cost and giving power to developers using the IDE tool. In the following, reference is made to embodiments of the invention. However, it should be understood that the invention is not limited to specific described embodiments. Instead, any combination of the following features and elements, whether related to different embodiments or not, is contemplated to implement and practice the invention. Furthermore, in various embodiments the invention provides numerous advantages over the prior art. Thus, the following aspects, features, embodiments and advantages are merely illustrative and are not considered elements or limitations of the appended claims except where explicitly recited in a claim s. One embodiment of the invention is implemented as a program product for use with a computer system. The program s of the program product defines functions of the embodiments including the methods described herein and can be contained on a variety of computer-readable storage media. Illustrative computer-readable storage media include, but are not limited to: Such computer-readable storage media, when carrying computer-readable instructions that direct the functions of the present invention, are embodiments of the present invention. Other media include communications media through which information is conveyed to a computer, such as through a computer or telephone network, including wireless communications networks. Such communications media, when carrying computer-readable instructions that direct the functions of the present invention, are embodiments of the present invention. Broadly, computer-readable storage media and communications media may be referred to herein as computer-readable media. In general, the routines executed to implement the embodiments of the invention, may be part of an operating system or a specific application, component, program, module, object, or sequence of instructions. The computer program of the present invention typically is comprised of a multitude of instructions that will be translated by the native computer into a machine-readable format and hence executable instructions. Also, programs are comprised of variables and data structures that either reside locally to the program or are found in memory or on storage devices. In addition, various programs described hereinafter may be identified based upon the application for which they are implemented in a specific embodiment of the invention. One of ordinary skill in the art will readily recognize, however, that embodiments of the invention may be adapted for use with a wide variety of programming languages used to develop software applications. Similarly, embodiments of the invention may be adapted for use with other database query languages. Also similarly, embodiments of the invention are described herein adapted for use with the widely used XML markup language. However, the invention is not limited to the XML markup language; rather, embodiments of the invention may be adapted to other markup languages or other data object formats or data representations, whether now known or later developed. As shown, computing environment includes computer system Computer system is included to be representative of existing computer systems, e. However, embodiments of the invention are not limited to any particular computing system, application, device, or network architecture and instead, may be adapted to take advantage of new computing systems and platforms as they become available. Additionally, those skilled in the art will recognize that the illustration of computer system is simplified to highlight aspects of the present invention and that computing systems and data communication networks typically include a variety of additional elements not shown in FIG. As shown, computer system includes a processor or processors , a storage device , a networking device , and a memory , connected by a bus CPU is a programmable logic device that executes user applications e. Computer system may be connected to a display device and to user input devices Typically, user input devices include a mouse pointing device and a keyboard, and display device is a CRT monitor or LCD display. Additionally, the processing activity and hardware resources on computer system may be managed by an operating system not shown. Linux is a trademark of Linus Torvalds in the US, other countries, or both. Network device may connect computer system to any kind of data communications network, including both wired and wireless networks. Storage device stores application programs and data for use by computer system Typical storage devices include hard-disk drives, flash memory devices, optical media, network and virtual storage devices, and the like. As shown, storage device contains a database and a development project Database may store a collection of data records organized according to a data model For example, data model may provide a relational schema of tables columns, and keys for organizing data records stored in database and accessed using SQL database statements. Development project represents a collection of information used to build a software application. For example, development project may include source code files, resource scripts, etc. As shown, memory stores a number of software applications, including an IDE tool , a template-based generation tool , and a query tool Although shown as separate components for clarity, one of skill in the art will recognize that these components may be combined into a single application package. Also as shown, memory includes a project file Generally, IDE tool provides a software application used to develop other software applications. IDE tool may include an editor, a compiler and other useful software tools. Project file represents a file included in development project that is being edited by a developer using IDE tool , e. IDE tool may display the content of project file on display device and provide an interface that allows the user to edit that content. In one embodiment, IDE tool is configured to interact with template-based generation tool to generate application source code or other artifacts for a software application represented by development project For example, in one embodiment, template-based generation tool may be configured to generate application source code or other artifacts based on an existing database schema, existing source code e. The particular queries, objects, methods, or other source code generated by the template-based generation tool may be specified by a rapid application development template Further, IDE tool may provide a plurality of development templates , where a developer may extend, customize, and control what input data is recognized by given template , as well as what application source code or other artifacts are generated for that template and a given set of input data. Further, IDE tool may generate all of the necessary source code needed by the developer to invoke these operations from within application source code e. The generated application source code may be stored in an existing projecting file or written to a new project file As another example, template-based generation tool may be configured to generate application code source using an existing database query and a corresponding rapid application development template Building an application for a known, often highly-customized and complex database query, is a common task performed by a software developer. In such a case, the query itself may be a combination of various languages e. Additional scenarios for both input data sources, rapid application development templates , and generated application source code or other artifacts are described below. As shown, IDE tool may access a variety of different types of project files Project file 4 represents a database schema e. Project file 5 represents a structured data file e. Finally, project file 6 represents an existing database query e. In different embodiments, any of the data sources represented by project files may be accepted as metadata input to template-based generation tool to generate application source code or other artifacts. Of course, other input resources may be used to suit the needs of a particular case. In one embodiment, IDE tool may externalize data from any of project files via an information layer API For example, information layer API may parse one of project files and generate a normalized set of information passed to template-based generation tool Thus, information layer API may provide a unified facade to access information from one of project files in a consistent manner, regardless of the form of input data or which project file is used.

Chapter 2 : Digital Technical Journal - PDF Free Download

Engineering VAX Ada for a Multi-Language Programming Environment Charles Z. Mitchell Digital Equipment Corporation Abstract. DIGITAL's VAX TM Ada r is a validated, production-quality implemen-.

Embodiments of the invention support multiple scenarios for database-aware application development, including beginning from a database table and automatically creating application code to access the table, beginning from an existing database query, beginning from existing application code that accesses a database, and hybrids or variations of these approaches. Field of the Invention Embodiments of the invention are related to tools used to develop application software. More specifically, embodiments of the invention provide an intelligent integrated development environment IDE tool used for rapid application development of database-aware applications. Description of the Related Art Developing software applications is a complex task, and IDE tools are available to assist computer programmers with the development process. Currently, IDE tools are available to assist programmers developing applications in a variety of programming languages e. These tools are typically configured with features such as auto-indenting, syntax highlighting, type checking, and a variety of other features that assist the development process. An IDE tool usually includes a text editor that visually displays errors as source code is typed, allowing a developer to correct errors before proceeding with the next line to code. Typically, IDE tools are optimized for different programming languages and errors are identified based on the programming language being used by the developer, often determined a suffix of a project file e. Although very useful, these IDE tools have a variety of limitations. For example, IDE tools provide little support for database statements embedded within the source code of an application. In the program source code, database statements are usually specified as text strings in a database query language, such as SQL. Database-aware software applications are typically built using an application programming interface API that provides a means to use native query languages e. Although effective, this approach may require substantial development time, as the developer is required to write all of the necessary source code to create a connection to a particular database, to pass the text of a database query to the database, to receive query results, and to store the results in an object of the application program; none of which is likely to be part of the core functions of an application. That is, the application program typically retrieves information from the database to perform some other logic or processing. The rest is simply overhead the developer must incur for the database-aware application to access an external database. Similarly, the developer must write all of the necessary source code to test each element of the application that accesses an external data source i. Thus, the developer spends significant time performing tasks that, while necessary, are unrelated to writing the source code that performs the intended functions of an application being developed. Moreover, the total development experience for a quality application grows exponentially difficult as complexity of the database accesses increase. Further, as the IDE environment does not provide database connectivity or programming assistance for the database statements embedded within the source code of an application program, developers are forced to juggle between various tools, spending substantial time to even get a database-aware application up and running before writing the core of the application, resulting in added cost to the application development process. Accordingly, there remains a need for an IDE tool that provides rapid application development support for database-aware applications. For example, embodiments of the invention may be used to generate a software component configured to access a specified database using an appropriate API. The generated component may include a collection of database queries and the appropriate API calls to create, retrieve, update, and delete records from a given database. Once the component is generated, the developer may invoke methods provided by the component to perform database operations without having to write the component from scratch. By doing so, the developer may focus on the key task of application development, instead of writing the incidental but necessary code used to access the database. Additionally, the IDE tool may be configured to generate unit tests used to evaluate the functioning of the generated component. One embodiment of the invention includes a method for generating source code for a database-aware software application. The method generally includes receiving, from a user interacting with an IDE tool, a selection of a database element and generating at least one database statements to access the database element. The method also includes encapsulating the at least one database statements within source code of a programming language in which the database-aware software application is being written, storing the source code within a project file associated with the database-aware software application, and displaying the source code in an editing pane of the IDE tool. Another embodiment of the invention includes a computer program product comprising a computer useable storage medium having a computer readable program, where the computer readable program when executed on a computer causes the computer to perform an operation. The operation may generally include receiving, from a user interacting with an integrated development environment IDE tool, a selection of a database element and generating at least one database statements to access the database element. The operation may also include encapsulating the at least one database statements within source code of a programming language in which the database-aware software application is being written. The operation may also include storing the source code within a project file associated with the database-aware software application and displaying the source code in an editing pane of the IDE tool. Another embodiment of the invention includes a system having a processor and a memory containing an IDE tool configured to generate source code for a database-aware software application. The operation may also include encapsulating the at least one database statements within source code of a programming language, wherein the user is writing the database-aware software application in the programming language, storing the source code within a project file associated with the database-aware software application, and displaying the source code in an editing pane of the IDE tool. It is to be noted, however, that the appended drawings illustrate only typical embodiments of this invention and are therefore not to be considered limiting of its scope, for the invention may admit to other equally effective embodiments. Advantageously, embodiments of the invention provide automatic code generation, query generation for multiple languages, and unit test automation, leaving the developer to spend more time focused on designing and writing code to perform the intended functions of a database-aware application. Further, the developer may make frequent use of unit-test programs executed during application development, providing seamless integration of database connectivity with the regular development process. Thus, embodiments of the invention may significantly reduce application development and testing overhead related cost. In the following, reference is made to embodiments of the invention. However, it should be understood that the invention is not limited to specific described embodiments. Instead, any combination of the following features and elements, whether related to different embodiments or not, is contemplated to implement and practice the invention. Furthermore, in various embodiments the invention provides numerous advantages over the prior art. Thus, the following aspects, features, embodiments and advantages are merely illustrative and are not considered elements or limitations of the appended claims except where explicitly recited in a claim s. One embodiment of the invention is implemented as a program product for use with a computer system. The program s of the program product defines functions of the embodiments including the methods described herein and can be contained on a variety of computer-readable storage media. Illustrative computer-readable storage media include, but are not limited to: Such computer-readable storage media, when carrying computer-readable instructions that direct the functions of the present invention, are embodiments of the present invention. Other media include communications media through which information is conveyed to a computer, such as through a computer or telephone network, including wireless communications networks. Such communications media, when carrying computer-readable instructions that direct the functions of the present invention, are embodiments of the present invention. Broadly, computer-readable storage media and communications media may be referred to herein as computer-readable media. In general, the routines executed to implement the embodiments of the invention, may be part of an operating system or a specific application, component, program, module, object, or sequence of instructions. The computer program of the present invention typically is comprised of a multitude of instructions that will be translated by the native computer into a machine-readable format and hence executable instructions. Also, programs are comprised of variables and data structures that either reside locally to the program or are found in memory or on storage devices. In addition, various programs described hereinafter may be identified based upon the application for which they are implemented in a specific embodiment of the invention. One of ordinary skill in the art will readily recognize, however, that embodiments of the invention may be adapted for use with a wide variety of programming languages used to develop database-aware applications. Similarly, embodiments of the invention may be adapted for use with other database query languages. As shown, computing environment includes computer system Computer system is included to be representative of existing computer systems, e. However, embodiments of the invention are not limited to any particular computing system, application, device, or network architecture and instead, may be adapted to take advantage of new computing systems and platforms as they become available. Additionally, those skilled in the art will recognize that the illustration of computer system is simplified to highlight aspects of the present invention and that computing systems and data communication networks typically include a variety of additional elements not shown in FIG. As shown, computer system includes a processor or processors , a storage device , a networking device , and a memory , all connected by a bus CPU is a programmable logic device that executes user applications e. Computer system may be connected to a display device and at least one input devices Typically, user input devices include a mouse pointing device and a keyboard, and display device is a CRT monitor or LCD display. The processing activity and hardware resources on computer system may be managed by an operating system not shown. Linux is a trademark of Linus Torvalds in the U. Network device may connect computer system to any kind of data communications network, including both wired and wireless networks. Storage device stores application programs and data for use by computer system Typical storage devices include hard-disk drives, flash memory devices, optical media, network and virtual storage devices, and the like. As shown, storage device contains a database and a development project Database may store a collection of data records organized according to a data model For example, data model may provide a relational schema of tables, columns, and keys for organizing data records stored in database and accessed using SQL database statements. Development project represents a collection of information used to build a software application. For example, development project may include source code files, scripts, etc. As shown, memory stores a number of software applications, including an IDE tool , a query parser , and a query tool Also, memory includes a project file Query parser may be a software application configured to evaluate a database statement for both syntactic and semantic correctness. And query tool may be a software application configured to execute a valid database statement e. In one embodiment, IDE tool may be configured to pass database statements to query tool for execution. IDE tool is a software application used to develop other software applications. Typically, IDE tool combines an editor, a compiler and other useful tools in the same software package. In one embodiment, IDE tool is configured to generate elements of source code for a database-aware software application. For example, in one embodiment, IDE tool may generate the appropriate source code to connect to a database associated with development project along with objects that encapsulate database statements used to create new records and to retrieve, update, and delete existing records from database Further, IDE tool may generate all of the necessary source code allowing the developer to invoke these operations from within application source code e. Additionally, IDE tool may be configured to generate and execute unit tests to test the functionality of the database-aware application as it is being developed. Thus, unlike conventional development methodologies where the database statements are only tested when the application is built and executed, embodiments of the present invention allow database testing to become an integrated part of the database software development process. In another embodiment, IDE tool may also be configured to generate database-aware application code around an existing query. Building an application for a known, often highly-customized and complex database query is a common requirement. The query itself can be a combination of various languages e. IDE tool hides this complexity from the developer by generating an application around such an existing query. Project file represents a file included in development project that is being edited by a developer using IDE tool , e. IDE tool may display the text of the source code to the developer on display device and provide an interface that allows the user to edit project file In one embodiment, database-aware application source code generated by IDE tool may be stored in project file As shown, the method begins at step where IDE tool receives parameters related to an existing database to associate with a development project e. In the example of FIG. Dialog box includes a list of available connections and a table a set of properties associated with a currently-highlighted database connection. Once the developer specifies the properties of a desired database connection, then at step , IDE tool may establish a connection with the database specified at step Additionally, IDE tool may associate the parameters of the database connection with development project Thus, the developer may save the development project and resume it at a later time without also having to recreate the database connection for a given project.

Chapter 3 : Full text of "Bulletin of Agricultural and Technical College of North Carolina"

DIGITAL's VAX â„¢ Ada r is a validated, production-quality implementation of the full Ada language that is well-integrated into the VMS â„¢ operating system environment on VAX systems. The programming support environment consists of an Ada compiler, an Ada program library manager, and a multi-language programming environment including a variety.

Because all compi led languages use th is standard, modu les written in d i fferent languages can always cal l each other, provided both languages understand the data types of the parameters. SCA can provide cross-reference services and stat ic analysis for any language whose compi ler creates analysi s fi les. To support multiple languages, all tools use essent i a l ly the same i m plementation strategy. They define a s ingle canon ical representation for the data they need so that the same data from two d ifferent l a nguages is always represented the same way. LSE has only one template file format and one d iagnostics fi le format. PCA uses the same symbol i n formation as the debugger. If two languages pass a given p iece of i n formation to a given tool , they must always do it the same way. The debugger must su pport every data type that occurs in any language. SCA must understand every kind of cross-reference and every kind of caJl i ng 18 seq uence that may come u p i n a multi language progra m , even though no one language has them a l l. PCA and the debugger must both understand case-sensitivity, which occurs only in C. Multi language support thus complicates the design of most progra m m i ng roots considerably. The tools must be designed to cope with a wide variety of language constructs. They must a lso be very extensible since it is impossible to pred ict what l anguages they may have to support in the fut u re. As a result, the tools generally are very table-drive n , and they are very dependen t on having wel l-defined i nterfaces wi th the compi l ers and the other tools. However, there arc also substant ial savings in solving a given prob lem once for I 2 languages i nstead of solving it 1 2 t i mes. The i ncreasing use of workstations and their capabili ties is another trend that wi l l affect practica l ly a l l tools i n the VAXjVMS programming environment t o one degree or another. Knu t h , The TeXbook Read ing: Ad d ison Wesley, 1 9 Digital Technical journal No. Harris Software Productivity Measurements One objective of Digital Software Engineering is to build and maintain high-quality software products at reduced costs. User-docu mentation writers and editors. The Digital e ngineeri ng cultu re a l l ows each software project team substantial freedom to deter m i ne i ts own conventions, standards, and i nfrastructure. I n t he 1 9 7 0s and early 1 9 80s few supported rools were avai lable, and tool deve lopmen t was done at the project l evel , if at a l l. Some processes were a u tomated , most were not. Another aspect of this c u lture was the sense that each project team had ro write a l l the code needed for that project. This attitude meant that code to do common routi nes was dupl icated from project to project. Each team believed that i ts problem was u n i q u e , that it cou l d not share code with any other tea m. These trends were as fol lows: They needed software systems that woul d provide them with a competitive advantage in t heir marketplaces. They a lso wanted software that cou ld be used safely by people of varied a b i l ities and training. These businesses had l ittle tolerance nor should they have had for software systems with defects. And these objectives have to be accomplished within the constrai nts of our budgets and the avai labil ity of good software engineers. The graph s howed the actua l and projected rates of growth in l ines-of-code per programmer 4 from 1 98 0 through 1 9 9 0. This graph re-emphasized to u s the need to c larify how productivity shoul d be defined and measured. Productivity in software development is more complex than simply increasi ng the l ines-of-code produced by each programmer. The productivity of peopl e , regardless of how it is measured, is only one part of the software development process. In any case , that projection caused us to seek answers to several i mportant questions, such as the fol lowing: What do we mean by " better"? The remainder of this paper describes some answers to these questions and how the answers were derived. And our findi ngs justi fy the costs of coll ecting and analyzing the data so that we can know whether we are continuing to do better work. Software Productivity Software productivity encompasses more than just the programming of software prod ucts. A software system is completed only when the functional and performance requ irements have been met and when i t is useful for t he i n tended user. Two major d i mensions of software engineer ing prod uctivi ty are l the change i n quan t i ty of software produced for a given period of t i me at a given cost and 2 the qual i ty of the resu l tant software syste m. Then we can measure productivity and use the resul ts to help us focus on whether our process is better , and what needs to be changed in t he deve lopment process. The qual i ty attributes t hat are i mportant to the user of the software are i m portant also to the engi neer who supports and extends the software. For example, quality attr ibutes i ncl ude the usab i l i ty , usefu l ness, defect level and rate , and perfor mance of the software system. Also, we must i nclude the qua l i ty attributes that affect future costs; for example, the ease of mod ifying ro en hance or correct the software and the ease of porting the software to other hardware. The first factor wou ld be expected to decrease t he overall productivity of the project teams. The second two factors wou ld be expected to i ncrease productivi ty. Being easier, test ing is done more often. Many projects rout inely reb u i l d the project code using CMS and MMS and run either the entire test system or a part of it every nigh t. The next morn ing, the software engineers know i m mediately if the i r previous work caused a new problem or regression. Projects that h ave adopted th is process for b u i lds and tests have almost completely e l i m inated the many hours of integration at base leve l. The project members can open a separate topic dea l i ng with an issue. Reduced Redundancy O ur software engineers now search for code, designs, add i t ional tools, and doc u mentation that can be reused. Both managers and engineers consider reused code as an investment in design, programm ing, and testing that has already been paid for. Moreover, the su pport for that code has been planned and i s i n place. Reusable run-time components have been used and avai lable since the first version of the VAXjVMS operating system in 1 9 7 7. A software metric is a quantitative way to characterize an attribute of either the software system or the software development process. Then various software systems and development projects can be compared to themselves over t i me and to each other. That assu mes other variables remain constant; for example, s i m i lar types of organizations b u i lding s i milar types of software using s i m i lar methods and processes. O n ly when metrics h ave the same definition and therefore are measured in the same way shou ld? Size, usabi li ty, maintainab i l i ty , number of defects , and performance are a l l attribu tes of a 23 Software Productivity Measurements software syste m. For this study, we used t h ree software product metrics: Each l i ne is coun ted as one regardless of the number of operators, operands, and com ments that the l ine may i ncl ude. Incl ude fi les are cou nted once , and reused code shi pped w i t h t he product is counted. Blank l i nes are not coun ted. Project tools , tests , test data , and control fi les are a lso not cou nted.

Chapter 4 : BibTeX bibliography blog.quintoapp.com

Engineering VAX Ada for a multi-language programming environment View colleagues of Charles Z. Mitchell Engineering VAX Ada for a multi-language programming.

Volume 15, Number 1, January, John C. Reynolds Reasoning about arrays. Kaye Interactive Pascal input. Laski A hierarchical approach to program testing. Sutton An algorithm for user interaction. Taylor Assertions in programming languages. Mayor A language for network analysis and definition. Kieburtz The external consistency of abstract data types. Bernstein Global register allocation for non-equivalent register sets. Klaeren An abstract software specification technique based on structural recursion K. Raiha Bibliography on attribute grammars. Wulf Toward relaxing assumptions in languages and their implementations. Eisenberg Confinement of a class of harmful effects of the goto statement. Ellis A Lisp shell. Jones Tasking and parameters: Leinbaugh Indenting for the compiler. Galkowski A critique of the DOD common language effort. Blatt On the great big substitution problem Franklyn T. Bradshaw and George W. Ernst and Raymond J. Hookway and William F. Ogden Procedure semantics and language definition. Foster Performance measurement of a Pascal compiler. Reid Functions for manipulating floating-point numbers. Misra A simple model of distributed programs based on implementation-hiding and process autonomy. Campbell and Robert B. Kolstad and Roy H. Campbell Path Pascal user manual. Chiarini On FP languages combining forms. Dodd Some comments on a recent article by Salvadori and Dumont regarding the evaluation of compound conditional expressions. Sharp Data oriented program design. Tanik Two experiments on a program complexity perception by programmers. Gobin File handling in programming languages J. Goodenough Ada July syntax cross reference listing. Ledgard A human engineered variant of BNF. Moffat A categorized Pascal bibliography June, Goodenough The Ada compiler validation capability David S. Notkin An experience with parallelism in Ada Richard E. Fairley Ada debugging and testing support environments. Hutchison Using Ada for industrial embedded microprocessor applications. Belmont Type resolution in Ada: Sherman and Martha S. Borkan A flexible semantic analyzer for Ada. Luckham and Wolfgang Polak A practical method of documenting and verifying Ada programs with packages. Young and Donald I. Good Generics and verification in Ada. Clarke and Jack C. Wileden and Alexander L. Price The rendezvous and monitor concepts: Stevenson Algorithms for translating Ada multitasking. Filipski and Donald R. Moore and John E. Newton Ada as a software transition tool. Albrecht and Philip E. Garrison and Susan L. Graham and Robert H. Ada to Pascal and Pascal to Ada. Dewar and Gerald A. Rogers The design of a virtual machine for Ada Judy M. Bishop Effective machine descriptors for Ada Anonymous Compilers. Anonymous Execution models and architecture. Volume 15, Number 12, December, L. Buxton An informal bibliography on programming support environments. Craig Cleaveland Mathematical Specifications. Pentzlin A syntax for character and string constants supplying user-defined character codes. Siero APL and Algol68, the correspondence and the differences, especially in applications of graph-analysis. Sumpter and Gerry E. Quick Concurrency specification in high level languages. Taylor Protection of proprietary software. Rowe Data abstraction from a programming language viewpoint. Balzer Dynamic system specification. Borgida and Sol Greenspan Data and activities: Exploiting hierarchies of classes. Brodie Data abstraction for designing database-intensive applications. Carbonell Default reasoning and inheritance mechanisms on type hierarchies. Codd Data models in database management. Feather Some contrasts and considerations of an approach to modelling. Terry Hardgrave and Donald R. Deutsch Processing data model abstractions. Hayes and Gary G. Hendrix A logical view of types. Hendrix Mediating the views of databases and database users. Katz Heterogeneous databases and high level abstraction. King Modelling concepts for reasoning about access to knowledge. Leavenworth A data abstraction approach to database modelling. Levesque Incompleteness in knowledge bases. Mayr Make more of data types. Rowe Issues in the design of database programming languages. Schmidt Data abstraction tools: Design, specification and application. Sibley Database management systems past and present. Sowa A conceptual schema for Knowledge-based systems. Wasserman The extension of data abstraction to database management. Zilles Types, algebras and modeling. Bandyopadhyay A study on program level dependency of implemented algorithms on its potential operands. Bandyopadhyay Theoretical relationships between potential operands and basic measurable properties of algorithm structure. Cormack An algorithm for the selection of overloaded functions in ADA. Dewhurst An equivalence result for temporal logic Paul Rutter Using a high level language as a cross assembler. Hanson The Y programming language.

Chapter 5 : Digital Tectlnical Journal Cumulative Index | blog.quintoapp.com

The programming support environment consists of an Ada compiler, an Ada program library manager, and a multi-language programming environment including a variety of tools which all work together.

What is claimed is: A method to programmatically generate source code for a software application to connect to a database, comprising: The method of claim 1, wherein the first database statement is selected from statements to create, retrieve, update, and delete records from the database, wherein the second source code is used in passing the first database statement to the database over the database connection; wherein the selection is received from a user interacting with an integrated development environment IDE tool to author application code; wherein the first source code is of a programming language in which the software application is being written; wherein the method further comprises outputting the first source code for display in an editing pane of the IDE tool. The method of claim 2, wherein the database element is a database statement embedded as a text string within the source code of an application being developed using the IDE tool. The method of claim 3, further comprising: The method of claim 4, further comprising: The method of claim 5, wherein the data model is a relational schema. The method of claim 6, further comprising, generating a unit-test application for the software application, wherein the unit test executes i a first test query to retrieve a set of records from the database, ii a second test query to retrieve a first record of the set of records, a iii a third test query to update the first record with sample test values, and iv a fourth test query to delete the first record, and v a fifth test query to insert the first record back into the database. The method of claim 7, wherein the IDE tool provides a programming environment used to develop the software application, and wherein the IDE tool includes at least a graphical text editor and a compiler; wherein the IDE tool is configured to: A computer program product comprising a computer useable storage medium having a computer readable program, wherein the computer readable program when executed on a computer causes the computer to perform an operation comprising: The computer useable storage medium of claim 9, wherein the first database statement is selected from statements to create, retrieve, update, and delete records from the database, wherein the second source code is used in passing the first database statement to the database over the database connection; wherein the selection is received from a user interacting with an integrated development environment IDE tool to author application code; wherein the first source code is of a programming language in which the software application is being written; wherein the method further comprises outputting the first source code for display in an editing pane of the IDE tool. The computer useable storage medium of claim 10, wherein the database element is a database statement embedded as a text string within the source code of an application being developed using the IDE tool. The computer useable storage medium of claim 11, wherein the operation further comprises: The computer useable storage medium of claim 12, wherein the operation further comprises: The computer useable storage medium of claim 13, wherein the data model is a relational schema. The computer useable storage medium of claim 14, wherein the operation further comprises: The computer useable storage medium of claim 15, wherein the IDE tool provides a programming environment used to develop the software application, and wherein the IDE tool includes at least a graphical text editor and a compiler; wherein the IDE tool is configured to: The system of claim 17, wherein the first database statement is selected from statements to create, retrieve, update, and delete records from the database, wherein the second source code is used in passing the first database statement to the database over the database connection; wherein the selection is received from a user interacting with an integrated development environment IDE tool to author application code; wherein the first source code is of a programming language in which the software application is being written; wherein the method further comprises outputting the first source code for display in an editing pane of the IDE tool. The system of claim 18, wherein the database element is a database statement embedded as a text string within the source code of an application being developed using the IDE tool. The system of claim 19, wherein the operation further comprises: The system of claim 20, wherein the operation further comprises:

The system of claim 21, wherein the data model is a relational schema. The system of claim 22, wherein the operation further comprises: The system of claim 23, wherein the IDE tool provides a programming environment used to develop the software application, and wherein the IDE tool includes at least a graphical text editor and a compiler; wherein the IDE tool is configured to: Field of the Invention Embodiments of the invention are related to tools used to develop application software. More specifically, embodiments of the invention provide an intelligent integrated development environment IDE tool used for rapid application development of database-aware applications. Description of the Related Art Developing software applications is a complex task, and IDE tools are available to assist computer programmers with the development process. Currently, IDE tools are available to assist programmers developing applications in a variety of programming languages e. These tools are typically configured with features such as auto-indenting, syntax highlighting, type checking, and a variety of other features that assist the development process. An IDE tool usually includes a text editor that visually displays errors as source code is typed, allowing a developer to correct errors before proceeding with the next line to code. Typically, IDE tools are optimized for different programming languages and errors are identified based on the programming language being used by the developer, often determined a suffix of a project file e. Although very useful, these IDE tools have a variety of limitations. For example, IDE tools provide little support for database statements embedded within the source code of an application. In the program source code, database statements are usually specified as text strings in a database query language, such as SQL. Database-aware software applications are typically built using an application programming interface API that provides a means to use native query languages e. Although effective, this approach may require substantial development time, as the developer is required to write all of the necessary source code to create a connection to a particular database, to pass the text of a database query to the database, to receive query results, and to store the results in an object of the application program; none of which is likely to be part of the core functions of an application. That is, the application program typically retrieves information from the database to perform some other logic or processing. The rest is simply overhead the developer must incur for the database-aware application to access an external database. Similarly, the developer must write all of the necessary source code to test each element of the application that accesses an external data source i. Thus, the developer spends significant time performing tasks that, while necessary, are unrelated to writing the source code that performs the intended functions of an application being developed. Moreover, the total development experience for a quality application grows exponentially difficult as complexity of the database accesses increase. Further, as the IDE environment does not provide database connectivity or programming assistance for the database statements embedded within the source code of an application program, developers are forced to juggle between various tools, spending substantial time to even get a database-aware application up and running before writing the core of the application, resulting in added cost to the application development process. Accordingly, there remains a need for an IDE tool that provides rapid application development support for database-aware applications. For example, embodiments of the invention may be used to generate a software component configured to access a specified database using an appropriate API. The generated component may include a collection of database queries and the appropriate API calls to create, retrieve, update, and delete records from a given database. Once the component is generated, the developer may invoke methods provided by the component to perform database operations without having to write the component from scratch. By doing so, the developer may focus on the key task of application development, instead of writing the incidental but necessary code used to access the database. Additionally, the IDE tool may be configured to generate unit tests used to evaluate the functioning of the generated component. One embodiment of the invention includes a method for generating source code for a database-aware software application. The method generally includes receiving, from a user interacting with an IDE tool, a selection of a database element and generating at least one database statements to access the database element. The method also includes encapsulating the at least one database statements within source code of a programming language in which the database-aware software application is being written, storing the source code within a project file associated with the database-aware software application, and displaying the source code in an editing pane of the IDE tool. Another embodiment of the invention includes a computer program product comprising a computer useable storage medium having a computer readable program, where the computer readable program when executed on a computer causes the computer to perform an operation. The operation may generally include receiving, from a user interacting with an integrated development environment IDE tool, a selection of a database element and generating at least one database statements to access the database element. The operation may also include encapsulating the at least one database statements within source code of a programming language in which the database-aware software application is being written. The operation may also include storing the source code within a project file associated with the database-aware software application and displaying the source code in an editing pane of the IDE tool. Another embodiment of the invention includes a system having a processor and a memory containing an IDE tool configured to generate source code for a database-aware software application. The operation may also include encapsulating the at least one database statements within source code of a programming language, wherein the user is writing the database-aware software application in the programming language, storing the source code within a project file associated with the database-aware software application, and displaying the source code in an editing pane of the IDE tool. It is to be noted, however, that the appended drawings illustrate only typical embodiments of this invention and are therefore not to be considered limiting of its scope, for the invention may admit to other equally effective embodiments. Embodiments of the invention support multiple scenarios for database-aware application development, including beginning from a database table and automatically creating application code to access the table, beginning from an existing database query, beginning from existing application code that accesses a database, and hybrids or variations of these approaches. Advantageously, embodiments of the invention provide automatic code generation, query generation for multiple languages, and unit test automation, leaving the developer to spend more time focused on designing and writing code to perform the intended functions of a database-aware application. Further, the developer may make frequent use of unit-test programs executed during application development, providing seamless integration of database connectivity with the regular development process. Thus, embodiments of the invention may significantly reduce application development and testing overhead related cost. In the following, reference is made to embodiments of the invention. However, it should be understood that the invention is not limited to specific described embodiments. Instead, any combination of the following features and elements, whether related to different embodiments or not, is contemplated to implement and practice the invention. Furthermore, in various embodiments the invention provides numerous advantages over the prior art. Thus, the following aspects, features, embodiments and advantages are merely illustrative and are not considered elements or limitations of the appended claims except where explicitly recited in a claim s. One embodiment of the invention is implemented as a program product for use with a computer system. The program s of the program product defines functions of the embodiments including the methods described herein and can be contained on a variety of computer-readable storage media. Illustrative computer-readable storage media include, but are not limited to: Such computer-readable storage media, when carrying computer-readable instructions that direct the functions of the present invention, are embodiments of the present invention. Other media include communications media through which information is conveyed to a computer, such as through a computer or telephone network, including wireless communications networks. Such communications media, when carrying computer-readable instructions that direct the functions of the present invention, are embodiments of the present invention. Broadly, computer-readable storage media and communications media may be referred to herein as computer-readable media. In general, the routines executed to implement the embodiments of the invention, may be part of an operating system or a specific application, component, program, module, object, or sequence of instructions. The computer program of the present invention typically is comprised of a multitude of instructions that will be translated by the native computer into a machine-readable format and hence executable instructions. Also, programs are comprised of variables and data structures that either reside locally to the program or are found in memory or on storage devices. In addition, various programs described hereinafter may be identified based upon the application for which they are implemented in a specific embodiment of the invention. One of ordinary skill in the art will readily recognize, however, that embodiments of the invention may be adapted for use with a wide variety of programming languages used to develop database-aware applications. Similarly, embodiments of the invention may be adapted for use with other database query languages. As shown, computing environment includes computer system Computer system is included to be representative of existing computer systems, e. However, embodiments of the invention are not limited to any particular computing system, application, device, or network architecture and instead, may be adapted to take advantage of new computing systems and platforms as they become available. Additionally, those skilled in the art will recognize that the illustration of computer system is simplified to highlight aspects of the present invention and that computing systems and data communication networks typically include a variety of additional elements not shown in FIG. As shown, computer system includes a processor or processors , a storage device , a networking device , and a memory , all connected by a bus CPU is a programmable logic device that executes user applications e. Computer system may be connected to a display device and at least one input devices Typically, user input devices include a mouse pointing device and a keyboard, and display device is a CRT monitor or LCD display. The processing activity and hardware resources on computer system may be managed by an operating system not shown. Linux is a trademark of Linus Torvalds in the U. Network device may connect computer system to any kind of data communications network, including both wired and wireless networks. Storage device stores application programs and data for use by computer system Typical storage devices include hard-disk drives, flash memory devices, optical media, network and virtual storage devices, and the like. As shown, storage device contains a database and a development project Database may store a collection of data records organized according to a data model For example, data model may provide a relational schema of tables, columns, and keys for organizing data records stored in database and accessed using SQL database statements. Development project represents a collection of information used to build a software application. For example, development project may include source code files, scripts, etc. As shown, memory stores a number of software applications, including an IDE tool , a query parser , and a query tool

Chapter 6 : dblp: Charles Z. Mitchell

Engineering VAX Ada for a multi-language programming environment Engineering VAX Ada for a multi-language programming environment Mitchell, Charles Z DIGITAL's VAX â„¢ Ada r is a validated, production-quality implementation of the full Ada language that is well-integrated into the VMS â„¢ operating system environment on VAX systems.

Embodiments of the invention relate to statement generation using statement patterns. RDBMS uses relational techniques for storing and retrieving data in a relational database. Relational databases are computerized information storage and retrieval systems. Relational databases are organized into tables that consist of rows and columns of data. The rows may be called tuples or records or rows. A database typically has many tables, and each table typically has multiple records and multiple columns. SQL is commonly used by applications in order to interact with relational databases. A majority of SQL queries are similar in their structure and format, varying only in predicates or number of parameters. These variations occur due to a variety of factors, including optimistic concurrency control, paging e. Under an optimistic concurrency control scheme, locks are obtained immediately before a read operation and released immediately afterwards and update locks are obtained immediately before an update operation and held until the end of the transaction. By itself, SQL lacks portability i. One reason for the lack of portability in SQL is that some relational database vendors either do not follow the SQL standard or do not incorporate the entire standard. Another reason for lack of portability in SQL is that relational database vendors may choose to implement certain features of SQL differently because the SQL standard is not explicit in certain areas. These issues force an application programmer to write slightly different versions of an original SQL query in order to satisfy the syntactic requirements of each relational database the SQL query is supposed to run against. Some cross-platform applications utilize string concatenation to build different flavors of the same original SQL query at runtime. The main drawback of this technique is that it leaves no room for static execution. Static execution requires that a statement is available prior to the runtime so that the statement can be prepared and bound to the database. If String concatenation is used, the actual String is only available at runtime, thereby leaving no room for static execution. Preparing and binding the SQL statement may be described as processing the SQL statement to get an access plan and registering the statement and access plan with the database. Thus, application programs have the daunting task of writing one variation of the original SQL query for each relational database that the SQL query needs to run against. That is, application programmers are forced to learn various flavors and quirks of several flavors of SQL. The task of learning several SQL flavors not only presents a steep learning curve for the application programmers, but also makes it hard for the application programs to test these SQL queries. Moreover, application programmers must learn enough about the various options and flags for several flavors of SQL in order to write a correctly functioning and optimal SQL query for that particular relational database. This approach of manually modifying the original SQL query to fit the requirements imposed by each relational database is also not optimal due to the reason that each and every customized SQL query must change if the original SQL query changes. Thus, there is a need in the art for improved generation of SQL queries. A statement that includes at least one statement pattern is received, wherein each statement pattern is a template that indicates how a statement is to be modified for execution against a data store and wherein each statement pattern modifies program logic and control flow. Each statement pattern is expanded to generate one or more new statements, wherein each statement pattern is capable of being expanded to zero or more statement patterns. The one or more new statements are executed against a data store to obtain a result set. It is determined whether to modify the result set based on the at least one statement pattern. In response to determining that the result set is to be modified, modifying the result set. It is understood that other embodiments may be utilized and structural and operational changes may be made without departing from the scope of the invention. A statement pattern may be described as a template that can be used to generate new artifacts or to modify existing artifacts. An artifact may be described as a statement e. The template may be described as indicating how a statement is to modified before execution against a data store a. Embodiments enable application programmers to define generic statement patterns that allow application programs to customize, modify, and enhance a statement e. These pre-defined statement patterns may be applied to the original statement during development or at runtime to enable the statement to be modified so that the statement can be executed against different relational databases. A statement pattern may modify both program logic i. Also, the statement pattern may modify the program logic and one or more statements simultaneously. In certain embodiments, client and Web server are on a single computing machine. In alternative embodiments, client and Web server are on different computing machines. A client is coupled to a Web server The client includes code including at least one statement specifying one or more statement patterns and, optionally, one or more parameters Each statement may be an SQL statement, such as a query i. The parameters may be part of the statement pattern , part of the statement or passed in as arguments to a method on which the statement is defined. The client also includes one or more statement patterns The statement patterns may include pre-defined statement patterns from a developer and newly added statement patterns from a user. The newly added statement patterns may be versions of the pre-defined statement patterns that have been extended e. The Web server includes a service A service may be described as providing some capability. A service may also be described as a piece of application code that can be remotely invoked over a network by a client application or by another service. For example, the service may be a web service e. A REST service may be described as one in which information content is retrieved from a Web site by invoking a service that returns an XML file that describes and includes the information content. In certain embodiments, the service is a stateless data access service i. The service includes server application code , a data access framework , and a statement generator The server application code executes to respond to a client request. A data access framework may be described as a set of pre-built classes and their instances that collaborate with the application code to retrieve and store data from and to the data stores a. A data access framework may also be described as providing an abstraction that allows executing statements e. The statement generator is invoked by the data access framework to rewrite a received statement that includes at least one statement pattern. The statement generator returns one or more statements to the data access framework for execution. The computer and Web server may each comprise any computing device known in the art, such as a server, mainframe, workstation, personal computer, hand held computer, laptop telephony device, network appliance, etc. In certain embodiments, the client and Web server are coupled via a network. The Web server is coupled to data stores a. In certain embodiments, each data store a. In certain embodiments, each of the data stores a. Each data store a. Embodiments enable the definition of statement patterns that allow users to execute statements and perform result set augmentation. Result set augmentation may be described as modifying the result sets e. Use of statement patterns is flexible, extensible, and provides users with the option to extend existing statement patterns or to add new statement patterns Statement patterns allow users to map a result set returned from a data store a. In this manner, users can provide metadata that is used for mapping via the use of statement patterns Statement patterns not only modify the existing statement but also allow the result set of executing the statement to be modified in a way that the user wants. Embodiments allow users to specify statements and result set modification with one statement pattern Multiple scenarios can benefit from the use of these statement patterns Example scenarios are discussed herein merely to enhance understanding of the invention. During the application of statement patterns , parameters may be added to or removed from the original statement. As an example, parameters for page number and page size may be added due to the application of a paging pattern A paging pattern allows users to fetch one page of data at a time from the data store a. Similarly, the application of an optimistic concurrency pattern may add a combination of previous values, timestamps, row modification counts, or row identifiers as parameters to the original statement. Other statement patterns include name alias patterns that enable users to specify different names or column metadata. Metadata patterns avoid the use of statement e. Metadata that describes the statement is especially useful when different database systems return incomplete or inconsistent metadata for the same statement. A decomposition pattern allows users to convert a query on a view, or a complex query into a modification statement to the underlying tables. Statement patterns themselves may be parameterized in order to enable grouping of various functions into a single statement. As an example, a paging pattern may have two forms: When a statement pattern is parameterized, embodiments are able to identify which pattern should be executed based on parameter values. Although, examples herein may refer to SQL statements, however, embodiments may be used with various types of statements e. As an example, consider query 1 , which may be written by an applications developer using statement patterns: The query 1 also includes parameters: The pageByValue pattern includes a parameter startingA, which indicates the page number to start at, and a parameter numRows, which indicates a number of rows to be returned in a result set when the SQL statement is executed. In this example, the pageByValue pattern expands to the following logic 1 however, the pageByValue pattern may include different logic in different embodiments. As can be seen from logic 1 and logic 2 , both program logic and control flow i. Also, the statement pattern may modify the program logic and statements simultaneously e. Thus, statement patterns may be applied recursively to SQL queries. Additionally, statement patterns may be applied recursively to output result sets e. In the case of static SQL, the expanded queries are registered together i. The appropriate SQL query is then executed based on the parameters. Use of statement patterns allow actual statements e. Thus, allowing statements to be prepared and statically bound to the data store a. The application of statement patterns is also useful when a number of parameters is not known, but is based on a number of possible input parameter combinations.

Chapter 7 : Statement generation using statement patterns - International Business Machines Corporation

Charles Z. Mitchell: Engineering VAX Ada for a multi-language programming environment.