US 20150143338A1

(19) **United States**

(12) **Patent Application Publication**

BIEHL et al.

(10) **Pub. No.: US 2015/0143338 A1**

(43) **Pub. Date: May 21, 2015**

(54) **SPATIAL LAYOUT OF HIERARCHICAL SHARED RESOURCES**

(71) Applicant: **Microsoft Technology Licensing, LLC**, Redmond, WA (US)

(72) Inventors: **Jacob T. BIEHL**, Champaign, IL (US); **George G. ROBERTSON**, Seattle, WA (US); **Gregory R. SMITH**, Bellevue, WA (US); **Mary P. CZERWINSKI**, Woodinville, WA (US)

(73) Assignee: **MICROSOFT TECHNOLOGY LICENSING, LLC**, Redmond, WA (US)

(21) Appl. No.: **14/587,569**

(22) Filed: **Dec. 31, 2014**

Related U.S. Application Data

(63) Continuation of application No. 11/678,494, filed on Feb. 23, 2007, now Pat. No. 8,949,769.

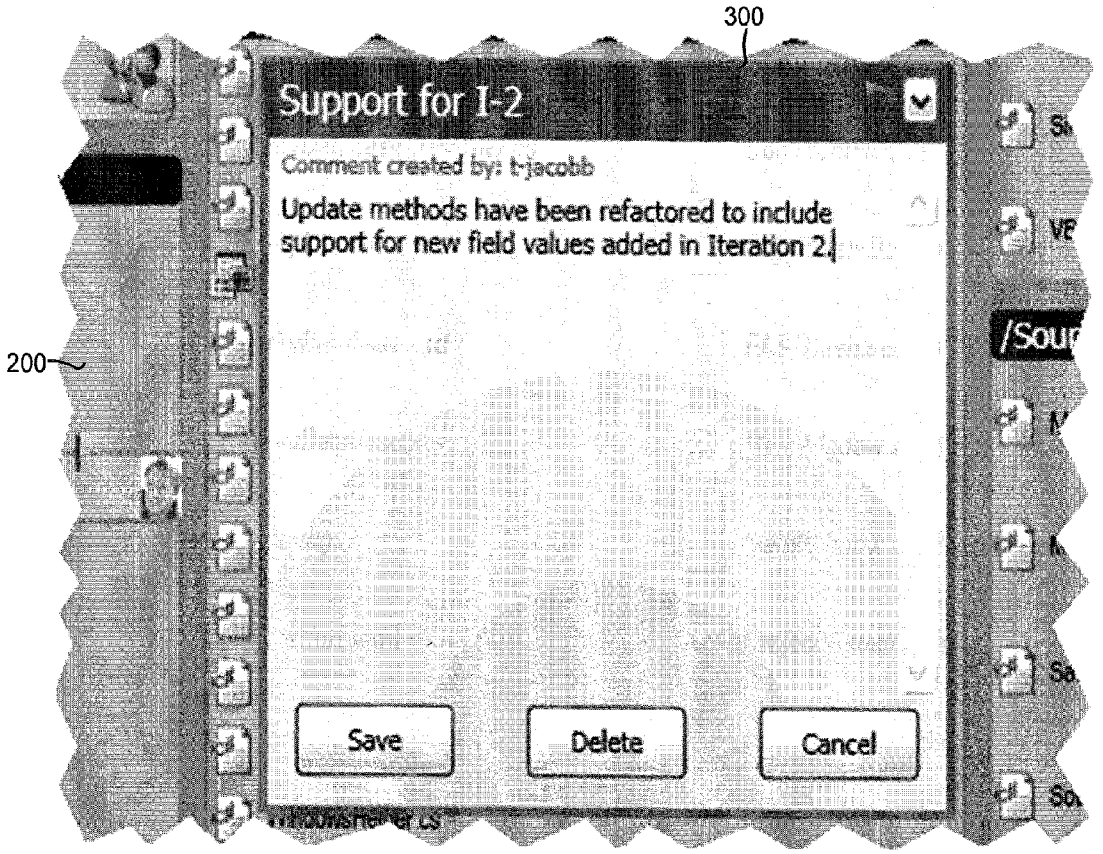
Publication Classification

(51) **Int. Cl.**
G06F 9/44 (2006.01)

(52) **U.S. Cl.**
CPC ***G06F 8/71*** (2013.01)

(57) **ABSTRACT**

A hierarchical shared resources spatial visualization system and method including a visualization runtime user interface that quickly and efficiently displays a spatial layout of a shared resource having a hierarchical nature. The user interface provides a spatial layout of the hierarchical shared resource and overlays salient activity information of a group's interaction with the shared resource. In software development, the user interface provides software teams with awareness of activity by other developers in the group regarding files in the shared source code base. The salient activity includes active file information (such as which files are open and by whom) and source repository actions (such as a developer's activity within a project's source repository system). Visual geometry and colors are employed to create a visually distinctive environment that is used to convey the salient activity information quickly and efficiently.



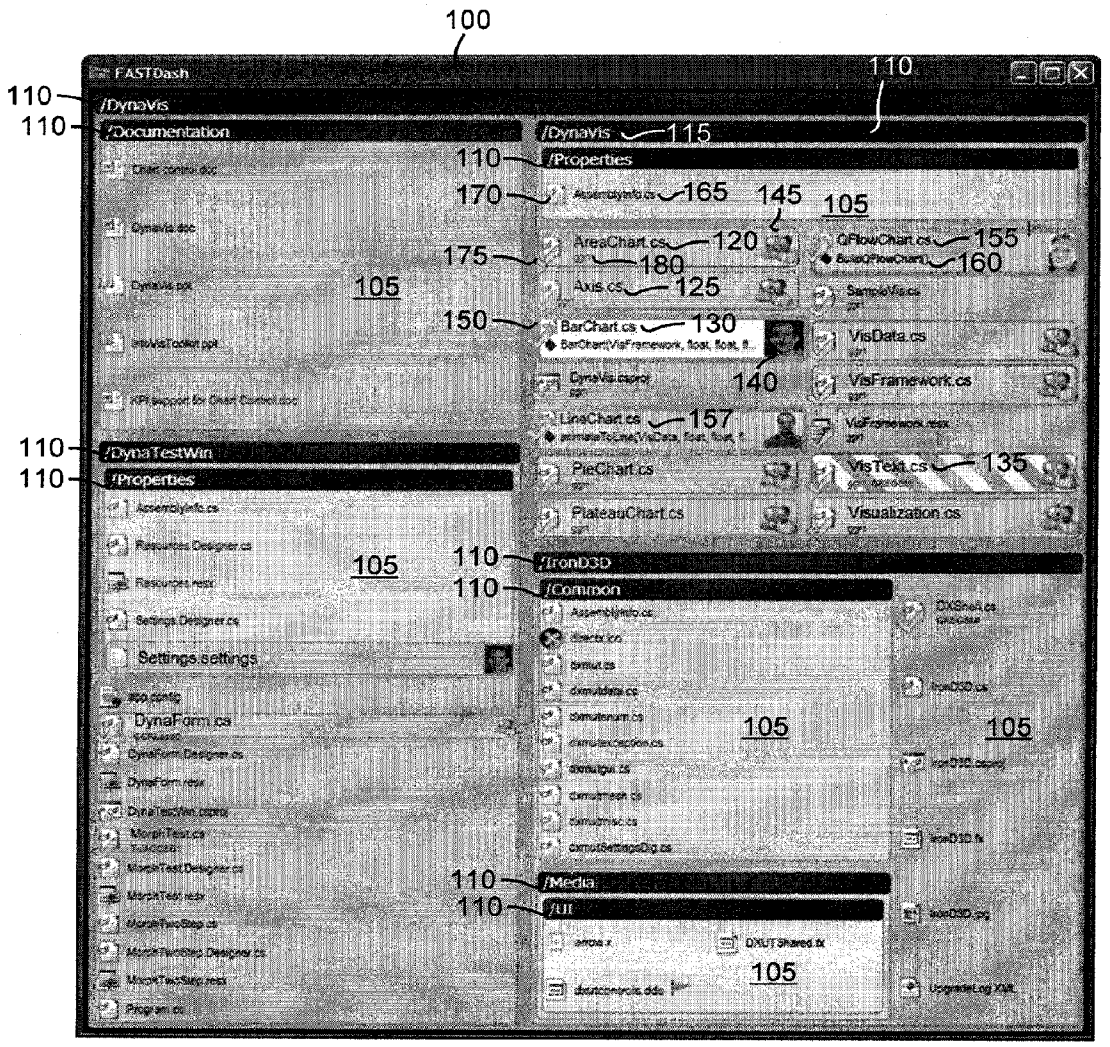

100
FASTDash
/DynaVis
/Documentation
/DynaTestWin
/Properties
/DynaVis
/Properties
/IronD3D
/Common
/Media
/UI
110
105
115
120
125
130
135
140
145
150
155
157
160
165
170
175
180
AreaChart.cs
Axis.cs
BarChart.cs
LineChart.cs
PieChart.cs
PlateauChart.cs
QFlowChart.cs
VisData.cs
VisFramework.cs
VisText.cs
Visualization.cs
DynaForm.cs
Settings.settings

FIG. 1

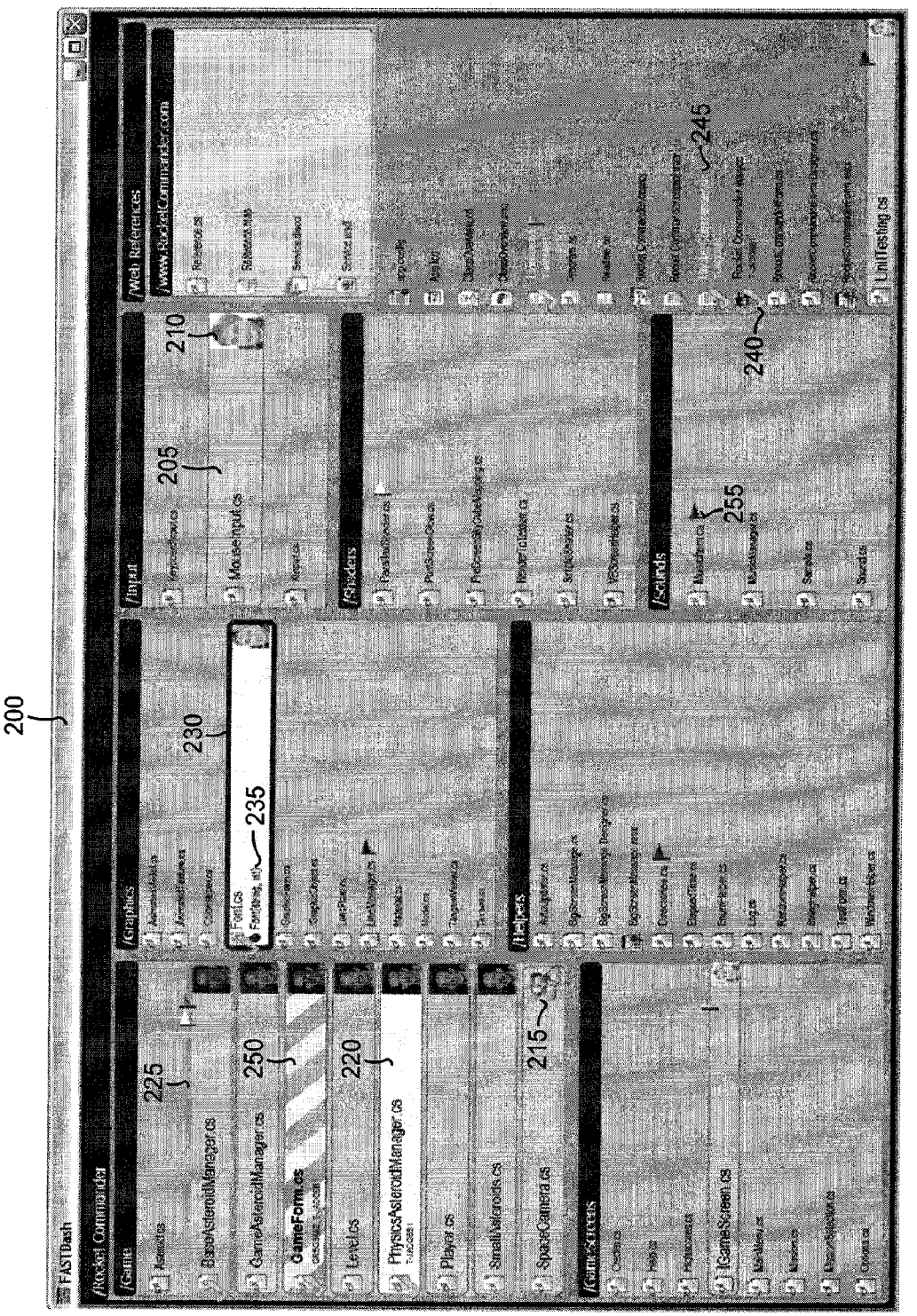

200
FASTDash
/Rocket Commander
/Game
/Graphics
/Input
/Web References
/www.RocketCommander.com
/Helpers
/Shaders
/Solids
/GameScreens
GameForm.cs
PhysicsAsteroidManager.cs
Font.cs
MouseInput.cs
UnitTesting.cs
205
210
220
225
230
235
240
245
250
255

FIG. 2

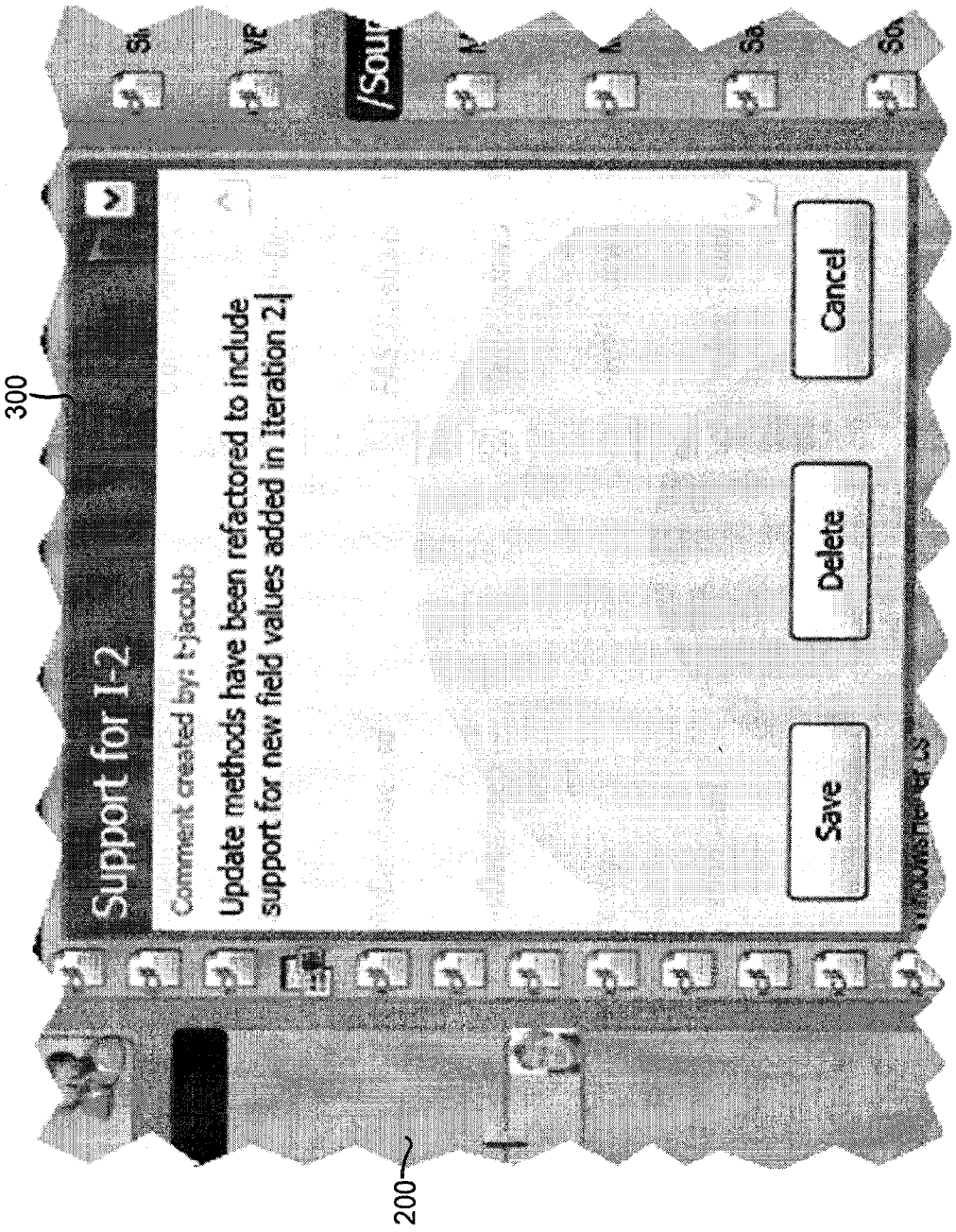

300
200
Support for 1-2
Comment created by: tjacobb
Update methods have been refactored to include support for new field values added in iteration 2.
Save
Delete
Cancel

FIG. 3

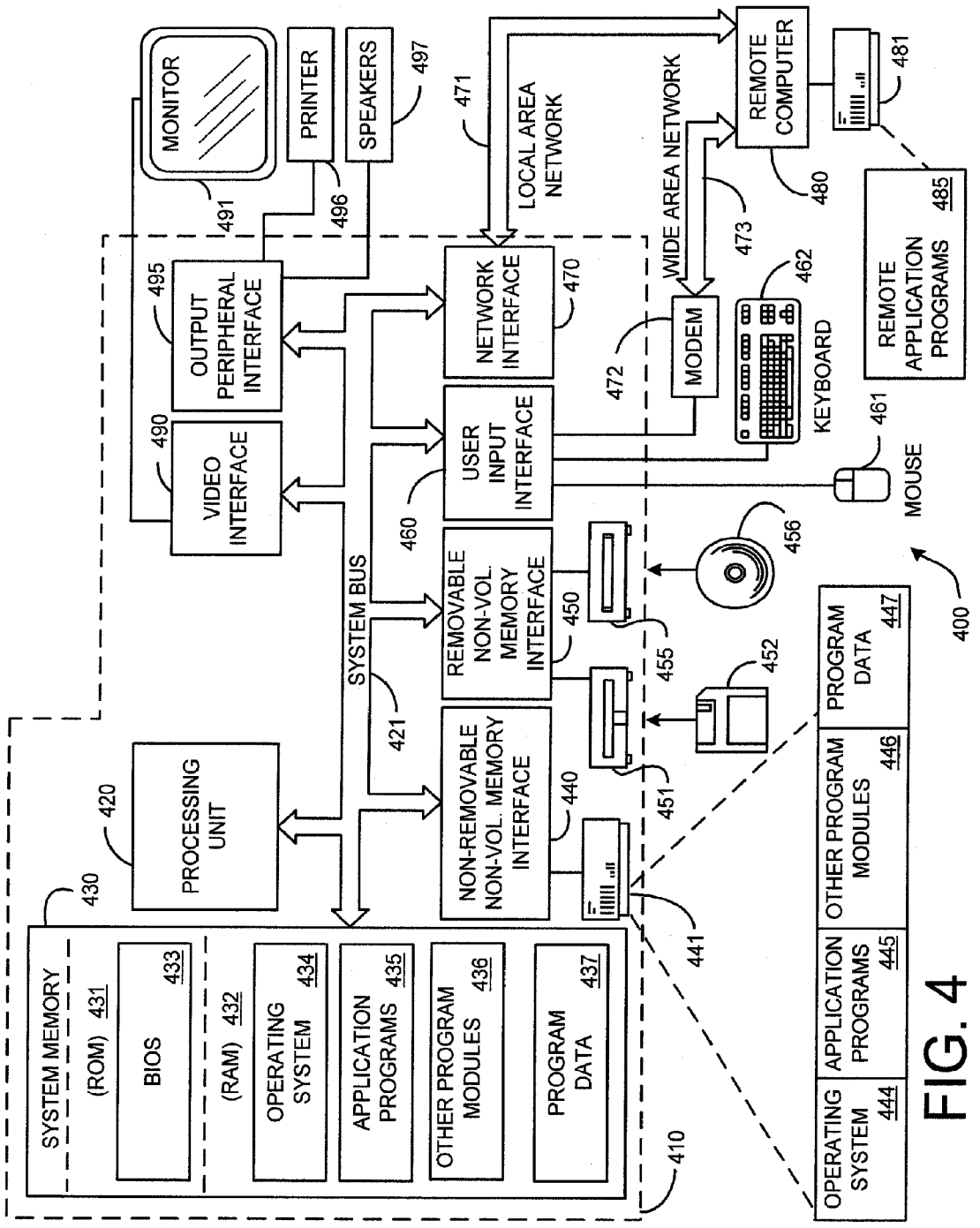

SYSTEM MEMORY
(ROM) 431
BIOS 433
(RAM) 432
OPERATING SYSTEM 434
APPLICATION PROGRAMS 435
OTHER PROGRAM MODULES 436
PROGRAM DATA 437
410
430
PROCESSING UNIT
420
SYSTEM BUS
421
NON-REMOVABLE NON-VOL. MEMORY INTERFACE
440
REMOVABLE NON-VOL. MEMORY INTERFACE
450
460
USER INPUT INTERFACE
VIDEO INTERFACE
490
OUTPUT PERIPHERAL INTERFACE
495
NETWORK INTERFACE
470
MONITOR
491
PRINTER
496
SPEAKERS
497
LOCAL AREA NETWORK
471
WIDE AREA NETWORK
473
MODEM
472
KEYBOARD
462
MOUSE
461
REMOTE COMPUTER
480
481
REMOTE APPLICATION PROGRAMS
485
441
451
455
456
452
400
OPERATING SYSTEM 444
APPLICATION PROGRAMS 445
OTHER PROGRAM MODULES 446
PROGRAM DATA 447

FIG. 4

SPATIAL LAYOUT OF HIERARCHICAL SHARED RESOURCES

BACKGROUND

[0001] There is a growing demand for software companies to produce reliable, defect-free software. While the causes of software defects are varied, a significant proportion of errors are caused during the software development stage by a poor collective understanding of project status and of fellow team member activities.

[0002] Software programmers spend significant time gaining and maintaining awareness of fellow developers' activities. Unlike other collaborative jobs where workers can observe each others' activities as they physically move and perform actions within their shared workspace, programmers use electronic tools to operate in a shared virtual environment where other programmers' activities are difficult (or even impossible) to observe directly. This significantly increases the effort for programmers to gain and maintain an awareness of how their workspace is changing.

[0003] Many software development teams leverage innovative methodologies to increase awareness and cooperation among programmers. These methodologies increase awareness and team interaction by employing shortened, highly iterative development cycles and co-locating programmers in the same physical workspace. However, programmers still lack adequate tools to support sufficient ongoing awareness of other group members' actions. For example, even with the methodologies currently in use, key pieces of awareness information (such as who is working with which source files) are difficult to determine.

[0004] Several tools have been created to enhance an individual's awareness and understanding of large software shared code bases. One such tool provides a line-based view of the code. Each line of code is mapped to a thin row in a column which represents a code file. Rows are colored to represent a particular attribute of the code line. For example, lines that have recently changed appear red, and blue if they are the least recently changed. Another awareness tool provides miniaturized views of code files that can be arranged spatially. Highlights are shown on the sections of files that the programmer has open in the code editor. Programmers can also click on the thumbnails to navigate within the code base. Yet another awareness tool provides programmers with cumulative information about how fellow developers have navigated a code base in the past. This enables programmers to build on the actions of others, to better understand and navigate the code themselves. However, while these awareness tools focus on improving understanding and navigation of a large code base (such as what files contain which classes and methods, or what sections of code have changed in the past), they are still unable to provide a real-time awareness of fellow team members' activities within the shared code base. For example, these awareness tools cannot indicate who has which file open, or what files are currently being edited.

[0005] Another type of awareness tool provides a line-oriented view but combines activity information as one of the visualization attributes. Some awareness tools provide programmers with a per-line level view of activity, which are good for learning specific changes within a single or small set of code files. However, this line-oriented view has difficulty accommodating large code bases.

[0006] Another type of awareness tool enables programmers to view a historical evolution of changes for an entire code base. Using a space-filling representation of the code base, the visualization highlights changes using animations of different code snapshots over time. However, these types of awareness tools do not allow programmers to quickly determine and maintain an awareness of the activities currently occurring in the code base or to coordinate the ongoing activities of programmers in a shared code base.

[0007] There also are tools that enhance group awareness. Group awareness, which is essential to effective collaboration, is the understanding of who you are working with, what is being worked on, and how your actions affect others. A common method for gaining and maintaining this awareness is through the use of shared artifacts. Several group awareness tools exist that convey status of shared artifacts. Some of these group awareness tools allow collaborators to share views of artifacts currently open on their screens with other collaborators. Other types of group awareness tools extend the capabilities to gain artifact awareness when working within a physical workspace composed of multiple shared and personal devices. Still other types of group awareness tools enable users to collaboratively work inside a shared document workspace. However, these group awareness tools lack a holistic view of the entire shared workspace. In addition, these tools merely replicate a collaborator's local workspace to remote users and do not extract and visualize only key individual activity information that is useful in an overview of workspace activity.

SUMMARY

[0008] This Summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used to limit the scope of the claimed subject matter.

[0009] The hierarchical shared resources spatial visualization system and method disclosed herein includes an interactive visualization that improves a software development group's awareness of each others' activities. The system and method use spatial representation of the shared code base to highlight developers' current activities in relation to the code. A visualization runtime user interface allows a developer to quickly determine which team members have source files checked out, which files are being viewed, and what methods and classes are currently being changed. The user interface can be annotated, allowing developers to supplement activity information with additional status details. The user interface provides immediate awareness of potential conflict situations, such as two developers editing the same source file.

[0010] The hierarchical shared resources spatial visualization system and method differs from existing awareness tools in at least two ways. First, the hierarchical shared resources spatial visualization system and method provides a holistic view of the entire shared workspace. Second, the hierarchical shared resources spatial visualization system and method does not replicate a collaborator's local workspace to remote users, but rather extracts and visualizes just the key individual activity information useful in an overview of workspace activity. The hierarchical shared resources spatial visualization system and method can be used in any work environment where many shared artifacts exist within the workspace, where those artifacts are usually accessed by multiple users, and where there is a need for collaborators to maintain awareness of how those artifacts are being used.

[0011] In general, the hierarchical shared resources spatial visualization system and method includes a front-end visualization runtime embodied in a user interface, a software development platform plug-in, and a structured query language (SQL) database. The user interface is a spatial layout of a shared resource having a hierarchical nature (such as shared source code files). In other words, for a shared source code base the folders in the code base are displayed as well as the files in each folder.

[0012] Overlaid on the spatial layout is the salient activity information relating to actions of developers in the group. These salient activities include active file information and source repository actions. Visual geometry (such as borders, checkmarks, and icons) and color (such as colored backgrounds and borders) are employed to create a visually distinctive environment that is used to convey the salient activity information quickly and efficiently. The visualization runtime user interface also allows developers to attach comments to specific files and marks these comments with flags. Developers are able to view or edit a comment by clicking on a particular flag.

[0013] The visualization runtime user interface enables immediate access to key information for group awareness. Such information includes which code files are changing, who is changing them, and how they are being used. The user interface can be displayed on a large display in the shared workspace or on a second personal display so that developers are provided a common, shared source for team activity awareness.

[0014] It should be noted that alternative embodiments are possible, and that steps and elements discussed herein may be changed, added, or eliminated, depending on the particular embodiment. These alternative embodiments include alternative steps and alternative elements that may be used, and structural changes that may be made, without departing from the scope of the invention.

DRAWINGS DESCRIPTION

[0015] Referring now to the drawings in which like reference numbers represent corresponding parts throughout:

[0016] FIG. **1** illustrates a first exemplary embodiment of the visualization runtime user interface of the hierarchical shared resources spatial visualization system and method disclosed herein.

[0017] FIG. **2** illustrates a second exemplary embodiment of the visualization runtime user interface of the hierarchical shared resources spatial visualization system and method.

[0018] FIG. **3** illustrates a dialog box as it appears when a developer is viewing, updating, or adding a comment to a file.

[0019] FIG. **4** illustrates an example of a suitable computing system environment in which the hierarchical shared resources spatial visualization system and method may be implemented.

DETAILED DESCRIPTION

[0020] In the following description of the hierarchical shared resources spatial visualization system and method, reference is made to the accompanying drawings, which form a part thereof, and in which is shown by way of illustration a specific example whereby the hierarchical shared resources spatial visualization system and method may be practiced. It is to be understood that other embodiments may be utilized and structural changes may be made without departing from the scope of the claimed subject matter.

I. Operational and System Details

[0021] The hierarchical shared resources spatial visualization system and method includes three components: (1) a front-end visualization runtime embodied in a user interface; (2) a plug-in for the software development platform (such as Visual Studio® by Microsoft® Corporation) being used by the development group; and (3) a structured query language (SQL) database (such as SQL Server 2005 by Microsoft® Corporation). The hierarchical shared resources spatial visualization system and method also uses a source control management (SCM) system for source file data. The hierarchical shared resources spatial visualization system and method supports existing SCM systems and alternative repository systems (such as Concurrent Versions System (CVS) and Subversion (SVN)). Each of these components will now be discussed in detail.

[0022] Software Development Platform Plug-in and SQL Database

[0023] The software development platform plug-in is responsible for providing activity data to a central SQL server for use by the visualization runtime user interface. In one embodiment, the plug-in uses the standard Visual Studio® plug-in application programming interface (API) to monitor which files are open, the current development modality (such as edit or debug), which file the developer is currently working in (known as the "file in focus"), and on which method or class the developer is working. Each running instance of the plug-in maintains a connection to the central SQL database and reports its active files to this server. In one embodiment, the plug-in identifies itself by machine name, user, and process ID. The plug-in also periodically polls the SCM system for file checkout status and uses hash signatures of the local and repository files to determine if a checked out file has been changed.

[0024] Multiple instances of the plug-in for a given workstation cooperate so that only one plug-in takes responsibility for reporting these source repository actions to the central server at any given time. In one embodiment, in order to keep all active files and source repository states consistent and current, the server maintains a list of reporting processes and periodically cleans up after processes that have unexpectedly stopped reporting. The polling and server communication of the plug-in is performed on a background thread to avoid interfering with a developer's user interface experience.

Visualization Runtime User Interface

[0025] The visualization runtime user interface provides developers with several pieces of key information to enhance awareness of programming activities of a group of developers in a software development environment. At the foundation of the user interface is a spatial layout or representation of the shared source code base. Within the representation, folders are visualized as nested rectangles and the size of each rectangle is based on the number of files contained in the folder. It should be noted that in alternate embodiments other metrics may be used to determine the size of the various rectangles and sub-rectangles. By way of example, these alternate metrics include the importance to the project of a file or file group, or an amount of activity in a file or file group. Files are represented in the user interface by a textual label of the file's

name and icon. Representing files and folders in this way has the advantage of allowing the layout to remain spatially stable as items are added and removed from the shared code base. While some embodiments of the visualization runtime user interface focus on files in a code base, alternate embodiments include a visualization of the work items assigned to users in a development project. In these embodiments, for example, size then can be determined by a length of time working on a particular item.

[0026] Salient activity information by the group using the shared source code base is overlaid on the spatial representation. In general, there are two types of activities that are overlaid: (1) active file actions; and (2) source repository actions. Active file actions are based on a developer's activity within his or her current software development platform session or sessions. File action information is useful because it helps convey the presence and interest of a developer within areas of the shared code base.

[0027] In one embodiment, the visualization runtime user interface is designed to be viewed on a shared display (such as a hallway display or projected display in a common work area) where it can be seen by everyone working on a project. In other words, each developer within the group working on the shared code base can see the user interface. In alternate embodiments, the user interface also can be displayed on an individual display, or on both a shared display and individual displays. The visualization runtime user interface queries the SCM system at startup to build the source file hierarchy, and subsequently begins polling the central SQL database for team activity. Thus, as an independent executable process, the visualization runtime user interface can easily be placed on a shared screen in a group's workspace, on an individual programmer's workstation, or on a small peripheral display.

[0028] Color is used in the user interface to indicate various types of activity. The basic idea of color is to provide a distinctive visual property that quickly and efficiently conveys the importance of salient activity information relating to the shared code base. Thus, if a first activity on the shared code base is more important for the development group to be aware of than a second activity, the visual property (or color) that represents the first activity is more distinct than the visual property (or color) of the second activity. In certain embodiments, that may mean that the color of the first activity is brighter or has a higher contrast than the color of the second activity. Colors used in the user interface should be distinct from each other and easily visible on a display device. In some embodiments, a user can select which colors represent which features and activities.

[0029] Borders and backgrounds of various colors are shown on certain files if the file has been opened inside of the software development platform. Different colors indicate different types of activities that are being performed by developers. A colored border around a file indicates that a developer has the file currently in view in his development environment. The use of color allows developers to quickly obtain a contextual footprint of where other developers in the group are working. Thus, if several developers are in the same part of the code base, potential conflicts and activity overall can be easily understood by the group using the visualization runtime user interface. It should be noted that although certain colors are shown in the following figures to indicate certain activity, these colors can easily be changed manually by a user or automatically by the system.

[0030] In general, the user interface is a map of the shared code base. FIG. **1** illustrates a first exemplary embodiment of the visualization runtime user interface **100** of the hierarchical shared resources spatial visualization system and method. In the embodiment of FIG. **1**, three developers are shown working in a shared code base. The user interface **100** includes a first set of display areas **105** for listing names of a first level of a set of hierarchical shared resources. In the embodiment shown in FIG. **1**, the set of hierarchical shared resources is a shared code base and the upper levels correspond to folders in the shared code base. Thus, at the top of each of the first set of display areas **105** are folder names **110**.

[0031] As also shown in FIG. **1**, each of the first set of display areas **105** includes a second set of display areas for listing names of a second level of the hierarchical shared resource. The second level shared resource names are below the first level shared resource names in the hierarchy. Moreover, it should be noted that additional levels of the hierarchical shared resource indicate and correspond to additional levels of the folder hierarchy. In the exemplary embodiment of FIG. **1**, the lowest level shared resource names are file names, and the files are contained in the respective folder. This means that in addition to appearing in the lowest level, some of the files shown in FIG. **1** appear at higher hierarchical levels, such as at level **2**, level **3**, and level **4**. The file names are shown in the user interface **100** as individual line items within each folder. To avoid clutter, each file name shown does not have a reference number in FIG. **1**.

[0032] In the user interface **100**, the larger rectangular areas are folders and the individual line items are files. Thus, it is easy to see the hierarchy of the entire shared code base. Referring to the upper right corner of FIG. **1** in the DynaVis folder **115** is where activity is occurring. A blue background in the files QFlowChart.cs **155** and LineChart.cs **157** in the DynaVis folder **115** indicates that those files are open inside a developer's Visual Studio® client. In this exemplary embodiment, a blue background indicates that the file is open, and a yellow background indicates that the file is currently being edited. For example, in FIG. **1** the file BarChart.cs **130** is currently being edited. When a file is being edited, this means that the file is different on the developer's computer from what exists in the source repository. Anytime a developer is working he is changing code. When a developer is confident that the code is working and satisfies all the requirements that he has set out to satisfy, then he will go back and check the file into the source repository. Until then, the file is in an "edit" state as represented in the user interface by a yellow background. This communicates to other developers that before they start editing that file or start depending on that file that they should communicate with the developer that currently has the file checked out. At the very least other developers should be aware that changes are occurring to the file.

[0033] Checkmarks are also used in the user interface **100** to indicate source repository activity, namely, whether a file has been checked out for editing. In the exemplary embodiment of FIG. **1**, a yellow checkmark is used. For example, a yellow checkmark **175** adjacent the AreaChart.cs **120** file means that the file currently is checked out by a developer. A user identification **180** in small type below the name of the file is used to identify the developer who has the file checked out.

[0034] A hashed background of alternating bands of one color and another color indicates files that have been checked out from the repository by two or more people. For example,

in FIG. **1** the VisText.cs file **135** has alternating bands of gray and red hash marks to indicate that the file **135** has been checked out from the repository by two people. Source repository systems allow this to occur, but within the culture of a team of developers this activity can potentially lead to conflicts in the codebase if the two people are not coordinating on the nature and locations of their individual changes. The hierarchical shared resources spatial visualization system and method system uses the user interface to bring this potentially dangerous situation to the attention of the developers.

[0035] To the right of some of the file names in FIG. **1** are user images and icons. If a file name has an icon or picture of a single developer, this indicates that a single person has the file open in the current state. However, if a file name has an icon or image of two or more people, then it means that multiple persons have the file open. For example, in FIG. **1** the file BarChart.cs **130** has an image of a single developer **140**. This indicates that a single developer has that file open. In contrast, the file AreaChart.cs **120** has an icon of two persons **145**, meaning that at least two persons have that file open. An executable process in the hierarchical shared resources spatial visualization system and method system allows a user to hover over a user icon such that a list appears in the user interface **100** of all users that have that particular file open.

[0036] Borders also are used in the user interface to indicate activity. In the exemplary embodiment of FIG. **1**, a gold border around a file means the file is in focus. In focus means that the file that a developer is currently looking at on his display. A developer may have many files open in the software development platform, but file in the top window is the file that the developer is currently viewing, and is shown on the user interface **100** by the gold border. For example, in FIG. **1** the file BarChart.cs **130** has a gold border **150** indicating that the file is currently in focus by the single developer who has the file open. In addition, below the name of the file is text that indicates where the developer is in the file. For example, in FIG. **1** at the file called QFlowChart.cs **155** the single developer who has the file open is at the location in the file called BuildQFlowChart( ) **160**. In addition, in an exemplary embodiment a ruby border around a file means the file is in focus and is being debugged. This is discussed below and shown in FIG. **2**. In this situation, the file is being debugged and the code is being tested and not generated.

[0037] The backgrounds and borders discussed above are a set of overlays that indicate salient activity information of the group of developers working on the shared code base. This overlay of salient activity information pops up on the user interface **100** whenever a developer affects the shared code base. In addition, this salient activity information is updated on the user interface **100** automatically and in real time.

[0038] Next to each file name is an icon that is appropriate for that file. For example, each .cs file is shown as a C# (C-sharp) icon in FIG. **1**. For example, the AssemblyInfo.cs file **165** in FIG. **1** includes a C# icon **170** to indicate that the file is a .cs file. An executable process in the hierarchical shared resources spatial visualization system and method system allow a user to customize the icon to indicate the file type. Thus, the file type icon can be customized by the user. This feature connects what users see with the file type.

[0039] FIG. **2** illustrates a second exemplary embodiment of the visualization runtime user interface **200** of the hierarchical shared resources spatial visualization system and method. The embodiment of FIG. **2** illustrates the active file and source code repository activities for a software development project having two active developers. The salient activity information conveyed in FIG. **2** will now be discussed in detail.

[0040] One type of salient activity information includes active file information. This includes which files are open and by whom. For example, as shown in the exemplary embodiment of FIG. **2**, a blue background **205** indicates a file currently is open by a developer. A photograph **210** of the single developer who has the file open is shown to indicate presence for open files. As stated above with reference to FIG. **1**, a group icon **215** is shown when multiple developers have a file open. In addition, which files are currently being edited is also represented. For example, in FIG. **2** a yellow background **220** indicates that the file is open and is being edited by a developer.

[0041] The user interface **200** also indicates which file a developer currently is working on and what development modality (such as edit or debug) the programmer is using. For example, in FIG. **2** a gold border **225** indicates that a developer has the file in focus within the software development platform. Moreover, a ruby border **230** indicates that a developer has the file in focus and is in debug mode in the software development platform. If available and appropriate, the user interface **200** also indicates which method or class the programmer is currently editing, viewing, and debugging. For example, in FIG. **2** method/class information **235** for a file is provided when available.

[0042] Another type of salient activity information includes source repository actions. This type of activity is based on a developer's activity within a project's source repository system. Repository actions are useful because they allow fellow developers to understand how the shared source code base is currently changing.

[0043] This source repository information includes which files are checked out and to whom. For example, as stated above the yellow background **220** indicates that the file is open and is being edited by a developer. Moreover, a yellow checkmark **240** indicates that a file has been checked out. The salient activity information also includes which checked out files are different from the current version in the repository. For example, a file name in yellow text **245** indicates that a file has changed since it was checked out. The activity information also includes whenever there are potential checkout conflicts, such as when two or more developers have the same file checked out. For example, a hashed background **250** indicates that a file is currently checked out by two or more developers.

[0044] In addition to the development activities, the visualization runtime user interface also allows developers to attach comments to specific files. For example, in FIG. **2** a marker for existing comments is shown as a flag **255** near the file's name. This flag **255** indicates that a comment has been placed on the file. Each flag can have a different color as set by the creator of the comment or the system. These flags are mechanisms for developers to annotate files and have them easily recognized in the user interface. Developers can attach different messages to different flags.

[0045] Developers can view or edit a comment by clicking on a particular flag. FIG. **3** illustrates a dialog box **300** as it appears when a developer is viewing, updating, or adding a comment to a file. An executable process allows a developer to add new comments by right-clicking on a particular file and selecting the Add Comment option (not shown). Once the developer clicks on the file the dialog box **300** appears. The

developer enters a comment, closes the dialog box **300**, and then the flag **255** appears. In addition, the developer can click on the flag **255** to view or edit a flag using the dialog box **300**.

[0046] In the discussion above file checkout was performed at the source repository. In alternate embodiments, the user interface includes an executable process that allows a developer to check files in and out from the developer's station. If desired, this embodiment is optimal when the user interface is used as a personal tool to keep aware of the actions of others in the group.

[0047] In some embodiments the layout of the visualization runtime user interface is controlled by an algorithm called a squarified tree map. This algorithm is well-known by those having ordinary skill in the art. The squarified tree map is a formula for recursively dividing up display device real estate. If a display device has x by y pixels that can be filled, the squarified tree map is given hierarchical data and determines a layout for that data. For any given hierarchy of raw data, the screen will be divided up in a certain manner. The hierarchy is determined in part by giving a weight to each file. Virtually any feature that is important can be a weight. In the end result the weight will have some correlation with the amount of display device space that the associated file is given. In some embodiments, the metric that was used for weight was the file size. In this embodiment larger files received more display device space. Other embodiments included other metrics that were used to allocate the space. For example, the largest files may not be the most interesting files. One thing that makes a file large is that it is a large image format. But developers typically do not work on image files, so using the file size as a metric may not be useful. Thus, in some embodiments a metric used is the amount of file activity.

II. Exemplary Operating Environment

[0048] The hierarchical shared resources spatial visualization system and method is designed to operate in a computing environment. The following discussion is intended to provide a brief, general description of a suitable computing environment in which the hierarchical shared resources spatial visualization system and method may be implemented.

[0049] FIG. **4** illustrates an example of a suitable computing system environment in which the hierarchical shared resources spatial visualization system and method may be implemented. The computing system environment **400** is only one example of a suitable computing environment and is not intended to suggest any limitation as to the scope of use or functionality of the invention. Neither should the computing environment **400** be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in the exemplary operating environment.

[0050] The hierarchical shared resources spatial visualization system and method is operational with numerous other general purpose or special purpose computing system environments or configurations. Examples of well known computing systems, environments, and/or configurations that may be suitable for use with the hierarchical shared resources spatial visualization system and method include, but are not limited to, personal computers, server computers, hand-held (including smartphones), laptop or mobile computer or communications devices such as cell phones and PDA's, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronics, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and the like.

[0051] The hierarchical shared resources spatial visualization system and method may be described in the general context of computer-executable instructions, such as program modules, being executed by a computer. Generally, program modules include routines, programs, objects, components, data structures, etc., that perform particular tasks or implement particular abstract data types. The hierarchical shared resources spatial visualization system and method may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote computer storage media including memory storage devices. With reference to FIG. **4**, an exemplary system for the hierarchical shared resources spatial visualization system and method includes a general-purpose computing device in the form of a computer **410**.

[0052] Components of the computer **410** may include, but are not limited to, a processing unit **420** (such as a central processing unit, CPU), a system memory **430**, and a system bus **421** that couples various system components including the system memory to the processing unit **420**. The system bus **421** may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures include Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA) local bus, and Peripheral Component Interconnect (PCI) bus also known as Mezzanine bus.

[0053] The computer **410** typically includes a variety of computer readable media. Computer readable media can be any available media that can be accessed by the computer **410** and includes both volatile and nonvolatile media, removable and non-removable media. By way of example, and not limitation, computer readable media may comprise computer storage media and communication media. Computer storage media includes volatile and nonvolatile removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data.

[0054] Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical disk storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by the computer **410**. Communication media typically embodies computer readable instructions, data structures, program modules or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media.

[0055] Note that the term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared

and other wireless media. Combinations of any of the above should also be included within the scope of computer readable media.

[0056] The system memory **440** includes computer storage media in the form of volatile and/or nonvolatile memory such as read only memory (ROM) **431** and random access memory (RAM) **432**. A basic input/output system **433** (BIOS), containing the basic routines that help to transfer information between elements within the computer **410**, such as during start-up, is typically stored in ROM **431**. RAM **432** typically contains data and/or program modules that are immediately accessible to and/or presently being operated on by processing unit **420**. By way of example, and not limitation, FIG. **4** illustrates operating system **434**, application programs **435**, other program modules **436**, and program data **437**.

[0057] The computer **410** may also include other removable/non-removable, volatile/nonvolatile computer storage media. By way of example only, FIG. **4** illustrates a hard disk drive **441** that reads from or writes to non-removable, nonvolatile magnetic media, a magnetic disk drive **451** that reads from or writes to a removable, nonvolatile magnetic disk **452**, and an optical disk drive **455** that reads from or writes to a removable, nonvolatile optical disk **456** such as a CD ROM or other optical media.

[0058] Other removable/non-removable, volatile/nonvolatile computer storage media that can be used in the exemplary operating environment include, but are not limited to, magnetic tape cassettes, flash memory cards, digital versatile disks, digital video tape, solid state RAM, solid state ROM, and the like. The hard disk drive **441** is typically connected to the system bus **421** through a non-removable memory interface such as interface **440**, and magnetic disk drive **451** and optical disk drive **455** are typically connected to the system bus **421** by a removable memory interface, such as interface **450**.

[0059] The drives and their associated computer storage media discussed above and illustrated in FIG. **4**, provide storage of computer readable instructions, data structures, program modules and other data for the computer **410**. In FIG. **4**, for example, hard disk drive **441** is illustrated as storing operating system **444**, application programs **445**, other program modules **446**, and program data **447**. Note that these components can either be the same as or different from operating system **434**, application programs **435**, other program modules **436**, and program data **437**. Operating system **444**, application programs **445**, other program modules **446**, and program data **447** are given different numbers here to illustrate that, at a minimum, they are different copies. A user may enter commands and information (or data) into the computer **410** through input devices such as a keyboard **462**, pointing device **461**, commonly referred to as a mouse, trackball or touch pad, and a touch panel or touch screen (not shown).

[0060] Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, radio receiver, or a television or broadcast video receiver, or the like. These and other input devices are often connected to the processing unit **420** through a user input interface **460** that is coupled to the system bus **421**, but may be connected by other interface and bus structures, such as, for example, a parallel port, game port or a universal serial bus (USB). A monitor **491** or other type of display device is also connected to the system bus **421** via an interface, such as a video interface **490**. In addition to the monitor, computers may also include other peripheral output devices such as speakers **497** and printer **496**, which may be connected through an output peripheral interface **495**.

[0061] The computer **410** may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer **480**. The remote computer **480** may be a personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer **410**, although only a memory storage device **481** has been illustrated in FIG. **4**. The logical connections depicted in FIG. **4** include a local area network (LAN) **471** and a wide area network (WAN) **473**, but may also include other networks. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

[0062] When used in a LAN networking environment, the computer **410** is connected to the LAN **471** through a network interface or adapter **470**. When used in a WAN networking environment, the computer **410** typically includes a modem **472** or other means for establishing communications over the WAN **473**, such as the Internet. The modem **472**, which may be internal or external, may be connected to the system bus **421** via the user input interface **460**, or other appropriate mechanism. In a networked environment, program modules depicted relative to the computer **410**, or portions thereof, may be stored in the remote memory storage device. By way of example, and not limitation, FIG. **4** illustrates remote application programs **485** as residing on memory device **481**. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

[0063] The foregoing Detailed Description has been presented for the purposes of illustration and description. Many modifications and variations are possible in light of the above teaching. It is not intended to be exhaustive or to limit the subject matter described herein to the precise form disclosed. Although the subject matter has been described in language specific to structural features and/or methodological acts, it is to be understood that the subject matter defined in the appended claims is not necessarily limited to the specific features or acts described above. Rather, the specific features and acts described above are disclosed as example forms of implementing the claims appended hereto.

1-20. (canceled)

21. A method performed by at least one computing device, the method comprising:

displaying a user interface for visualization of a set of files of a shared code base;

allowing a group of users to access the files of the shared code base using the user interface;

displaying activity information on the user interface, the activity information identifying a first user from the group of users that has an individual file open, the individual file included in the set of files of the shared code base;

allowing the first user to associate a comment with the individual file;

displaying the user interface to a second user from the group of users indicating that the first user currently has the individual file open; and

enabling the second user to view the associated comment.

22. The method of claim **21**, wherein the activity information includes information related to checkout conflicts.

23. The method of claim **21**, further comprising:
automatically updating the user interface over time to reflect activity of the group of users.

24. The method of claim **21**, wherein the activity information indicates that another individual file included in the set of files of the shared code base is at least one of: (a) currently being debugged, (b) currently open, (c) currently checked out, (d) currently checked out by multiple users from the group of users, (e) currently being viewed, or (f) currently being edited.

25. The method of claim **21**, wherein the individual file is opened by multiple users at a time.

26. The method of claim **21**, further comprising accessing a database storing data reflecting the activity of the group of users to obtain the activity information.

27. The method of claim **21**, further comprising determining changes in another individual file of the shared code base.

28. A system comprising:
a processing unit; and
one or more volatile or nonvolatile computer-readable storage devices having stored thereon computer-executable instructions which, when executed by the processing unit, cause the processing unit to:
display a user interface for visualization of a set of files of a shared code base;
allow a group of users to access the files of the shared code base using the user interface;
display activity information on the user interface, the activity information identifying a first user from the group of users that has an individual file open, the individual file included in the set of files of the shared code base;
allow the first user to associate a comment with the individual file;
display the user interface to a second user from the group of users indicating that the first user currently has the individual file open; and
enable the second user to view the associated comment.

29. The system of claim **28**, wherein the activity information includes information related to checkout conflicts.

30. The system of claim **28**, wherein the computer-executable instructions cause the processing unit to:
automatically update the user interface over time to reflect activity of the group of users.

31. The system of claim **28**, wherein the activity information indicates that another individual file included in the set of files of the shared code base is at least one of: (a) currently being debugged, (b) currently open, (c) currently checked out, (d) currently checked out by multiple users from the group of users, (e) currently being viewed, or (f) currently being edited.

32. The system of claim **28**, wherein the individual file is opened by multiple users at a time.

33. The system of claim **28**, wherein the computer-executable instructions cause the processing unit to:
access a database storing data reflecting the activity of the group of users to obtain the activity information.

34. The system of claim **28**, wherein the computer-executable instructions cause the processing unit to:
determine changes in another individual file of the shared code base.

35. The system of claim **28**, embodied as a single computer or multiple computers.

36. A method performed by at least one computing device, the method comprising:
identifying actions performed on files of a shared code base that is shared by multiple users, wherein the actions relate to development or version control of the files of the shared code base;
producing a first instance of a visualization user interface comprising activity information that reflects the actions relating to the development or the version control of the files of the shared code base;
providing the first instance of the visualization user interface to a first user of the multiple users;
detecting that a second user has associated a comment with an individual file of the shared code base; and
causing the first instance of the visualization user interface provided to the first user to be updated with a comment indication conveying that the second user associated the comment with the individual file of the shared code base.

37. The method of claim **36**, wherein the activity information includes information related to checkout conflicts.

38. The method of claim **36**, further comprising:
automatically updating the first instance of the visualization user interface over time to reflect subsequent actions performed on the files of the shared code base.

39. The method of claim **36**, wherein the activity information indicates that another individual file of the shared code base is at least one of: (a) currently being debugged, (b) currently open, (c) currently checked out, (d) currently checked out by at least two of the multiple users, (e) currently being viewed, or (f) currently being edited.

40. The method of claim **36**, further comprising:
determining changes in another individual file of the shared code base.

* * * * *