Int J CARS (2013) 8:607–620
DOI 10.1007/s11548-013-0840-8

ORIGINAL ARTICLE

The Medical Imaging Interaction Toolkit: challenges and advances

10 years of open-source development

Marco Nolden · Sascha Zelzer · Alexander Seitel · Diana Wald · Michael Müller · Alfred M. Franz · Daniel Maleike · Markus Fangerau · Matthias Baumhauer · Lena Maier-Hein · Klaus H. Maier-Hein · Hans -Peter Meinzer · Ivo Wolf

Received: 10 January 2013 / Accepted: 28 March 2013 / Published online: 16 April 2013

Abstract
Purpose The Medical Imaging Interaction Toolkit (MITK) has been available as open-source software for almost 10 years now. In this period the requirements of software systems in the medical image processing domain have become increasingly complex. The aim of this paper is to show how MITK evolved into a software system that is able to cover all steps of a clinical workflow including data retrieval, image analysis, diagnosis, treatment planning, intervention support, and treatment control.
Methods MITK provides modularization and extensibility on different levels. In addition to the original toolkit, a module system, micro services for small, system-wide features, a service-oriented architecture based on the Open Services Gateway initiative (OSGi) standard, and an extensible and configurable application framework allow MITK to be used, extended and deployed as needed. A refined software process was implemented to deliver high-quality software, ease the fulfillment of regulatory requirements, and enable teamwork in mixed-competence teams.
Results MITK has been applied by a worldwide community and integrated into a variety of solutions, either at the toolkit level or as an application framework with custom extensions. The MITK Workbench has been released as a highly extensible and customizable end-user application. Optional support for tool tracking, image-guided therapy, diffusion imaging as well as various external packages (e.g. CTK, DCMTK, OpenCV, SOFA, Python) is available. MITK has also been used in several FDA/CE-certified applications, which demonstrates the high-quality software and rigorous development process.
Conclusions MITK provides a versatile platform with a high degree of modularization and interoperability and is well suited to meet the challenging tasks of today's and tomorrow's clinically motivated research.

Keywords Open-source · Medical image analysis · Platform · Extensible · Service-oriented architecture · Software process · Quality management · Image-guided therapy

M. Nolden (✉) · S. Zelzer · A. Seitel · D. Wald · M. Müller · A. M. Franz · M. Fangerau · M.Baumhauer · L. Maier-Hein · K. H. Maier-Hein · H.-P. Meinzer
Division of Medical and Biological Informatics (E130), German Cancer Research Center, Im Neuenheimer Feld 280, 69120 Heidelberg, Germany
e-mail: m.nolden@dkfz-heidelberg.de

I.Wolf
Mannheim University of Applied Sciences, Paul-Wittsack-Str. 10, 68163 Mannheim, Germany
e-mail: i.wolf@hs-mannheim.de

D. Maleike · M. Baumhauer
Mint Medical GmbH, Friedrich-Ebert-Str. 2, 69221 Dossenheim, Heidelberg, Germany
e-mail: d.maleike@mint-medical.de

Introduction

Research in the medical imaging domain has become increasingly complex over the years. The availability and variety of modalities has grown, offering more and more options for algorithmic research. Besides innovations in diagnostic imaging methods like high-angular-resolution diffusion MRI, PET-MRI or optical coherence tomography, intra-operative modalities such as C-Arms or tracking devices enter the operating room on a regular basis, enabling new methods of intra-operative support for all kind of interventions. This offers a variety of opportunities to improve patient

treatment, but at the same time presents the researcher with more and more challenges. The development of single methods has been replaced by whole pipelines of techniques working together, the static analysis of input data is often complemented with methods for interaction with the data. The site of operation has moved from the imaging researchers' laboratory to the radiological reading room or the operating room, bringing more live input data into the game and raising the expectations to the software with regard to usability, stability, and reliability.

Consequently, a software system designed for development of research software for a clinical environment must fulfill the requirements of the following clinically relevant tasks.

Data handling: Interfaces to clinical imaging systems should allow a seamless retrieval of the medical imaging data. New modalities such as diffusion tensor imaging offer new possibilities for quantitative imaging and diagnosis but also require specialized data handling, post-processing, and visualization methods. To make these methods accessible, a software environment should provide mechanisms to extend even low-level or internal parts like data representation or visualization.

Technical requirement: extensibility of data handling

Image analysis: Adding semantic value by image analysis is often essential for further use in complex application scenarios. A research environment should offer usable tools to the clinical user and thus support the development of new methods as well as their transfer into a usable software.

Technical requirement: support for method and tool development, e.g. rapid prototyping

Diagnosis support and treatment planning: After data acquisition and analysis the results can be taken to the physician, supporting the diagnosis and planning the treatment in an interactive way. Means of interaction and a certain level of usability should exist to enhance the acceptance by the clinician even in a research environment.

Technical requirement: interactivity and usability

Intervention support: The transfer of the planning results to the patient is increasingly supported by guidance systems that help the physician to follow the optimal path to target structures, identify and avoid risk structures, or supply additional information during the intervention.

Technical requirement: operating room device interfaces, live data processing, software robustness

Treatment control: The assessment of therapy results is becoming increasingly quantitative and thus image-based. An ideal environment also supports the structured acquisition and storage of follow-up measurements, supporting, for example, clinical studies.

Technical requirement: integration with clinical imaging systems

Furthermore, software applications are often directly used by clinicians or in a clinical environment even while still in research. This requires the software to be usable, reliable, and safe to use. Building functionally complex systems with these characteristics requires a well-defined software development process, eventually—depending on the usage scenario—leading to a complete quality management system. A recent publication by Ince et al showed the necessity of a research environment being open-source [10]. They argue that "with some exceptions, anything less than the release of source programs is intolerable for results that depend on computation." Making the source code of new methods publicly available should be compulsory. The fact that the manuscript was accepted as a Nature publication underlines the importance of this topic. Publishing source code is the *only* way to ensure that research results based on computation can be reproduced and others can build upon them. This aids collaboration between scientists and encourages the contribution to and enhancement of platforms. Many open-source packages exist for medical imaging—either specifically dedicated to the field or widely used for medical imaging purposes. Some packages are toolkits for typical tasks like image processing and analysis (ITK),[1] visualization (VTK),[2] tracking and related tasks in image-guided surgery (IGSTK) [5], real-time processing of image and video data (OpenCV) [3] or real-time simulation (SOFA) [1]. Other packages focus on specific topics like brain image analysis (SPM,[3] FSL,[4] freesurfer[5]). Some packages are even more specialized, created by a small team or even by a single person as the result of a research project.

Toolkit-level packages leave the responsibility for overall system design and—if more than one toolkit is required—the integration and interoperability of toolkits to the developer. Two other approaches try to provide additional support for the developer: development environment and extensible end-user oriented applications. Development environments provide developer-oriented applications, sometimes with visual programming tools, and interfaces to a number of toolkit-level solutions. Examples include SCIRun [19], OpenXIP,[6] MevisLab[7] and MATLAB[8] (the last three are not or only partially open-source). The other approach—extensible end-user oriented application—provides all the necessary basic methods in an integrated user interface, for example OsiriX

[1] http://www.itk.org.

[2] http://www.vtk.org.

[3] http://www.fil.ion.ucl.ac.uk/spm/.

[4] http://fsl.fmrib.ox.ac.uk/fsl/fslwiki/.

[5] http://surfer.nmr.mgh.harvard.edu/.

[6] http://www.openxip.org/.

[7] http://www.mevislab.de/.

[8] http://www.mathworks.com/.

[21], Slicer [20], Volview,[9] and Analyze.[10] Methods of extension are usually provided by one or more plugin interfaces, e.g. for command-line applications, shared objects or scripts. A more detailed overview of existing packages can be found in [25].

These approaches tackle problems on different levels and all have their strengths. The optimal solution would allow the developer to use the level of support that fits best to his problem, potentially even on different levels at different stages of a project. At the same time it should be lean and flexible enough not to impose a specific workflow on the developer.

In this article we show how the Medical Imaging Interaction Toolkit has evolved into a software system that is able to cover all steps of a clinical workflow including data handling, image analysis, diagnosis, treatment planning, intervention support, and treatment control.

Medical Imaging Interaction Toolkit

First planned as an in-house solution for reliable software development in the medical imaging domain MITK was published and released to the public in 2005 [26]. It started primarily as a toolkit, focusing on support for interactive multi-view applications by combining and extending ITK and VTK. Since then it gained more and more users in the biomedical community. The initial scope of the toolkit expanded over the years, some of the original ideas were adapted and a lot of work went into the architecture to maintain a toolkit and development environment for a diverse range of applications.

MITK is implemented in C++ and released under a BSD-style open-source license that allows users to build applications using MITK without imposing any restrictions or obligations on them. It can be built on Windows, Mac OS and Linux, using the cross-platform, open-source build-system CMake.

MITK is based on ITK and VTK, reusing as much as possible from these toolkits. All other dependencies to third-party libraries are optional to ensure a small footprint if required. ITK is mainly used for its base concepts (smart pointers, time-stamps, pipelines). Through connections to its large collection of filters and file readers, MITK gets access to many file formats of the medical imaging community. The visualization pipeline of MITK allows the flexible combination of VTK-based and OpenGL-based rendering, thus offering the whole palette of common volume and surface visualization techniques. In addition to VTK it cares automatically for the synchronous update of multiple views of the data. The addition of customized rendering methods to support novel techniques such as, Q-Ball imaging [7] or the path planning of needle insertions [23], is easily possible. Processing of color images like pathological or histological data is supported, although the focus is on radiological images like computed tomography, magnetic resonance imaging, and ultrasound. The creation of augmented reality applications [2,8,15] is supported through methods for video background rendering and real-time processing, e.g. camera distortion correction.

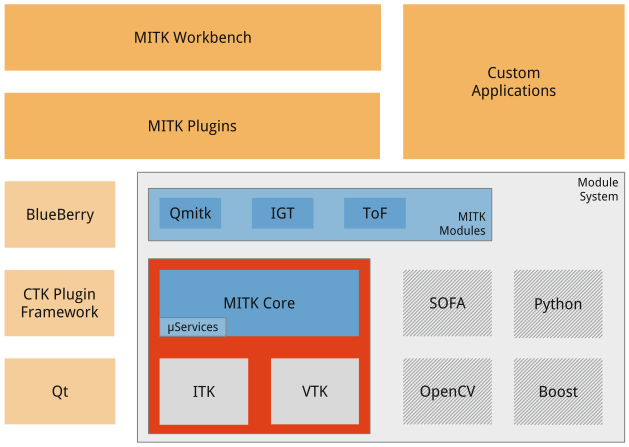

Fig. 1 MITK architecture: The *red* part shows the original structure of the toolkit, where the *MITK Core* extends ITK and VTK. The C++ micro services (section "C++ micro services") where added to the MITK Core to enable a service-oriented architecture at the toolkit level. Domain-specific code is added to separate MITK modules which depend on the MITK Core and optionally on third-party toolkits. The dependencies are handled by the MITK module system (section "The module system"). For application-level support, MITK provides the MITK Workbench (section "MITK application framework") which leverages the BlueBerry application framework (see Fig. 3), itself based on the CTK Plugin Framework (section "CTK Plugin Framework")

In the following sections we will describe how we extended MITK from its originally basic toolkit-oriented architecture (red part in Fig. 1) to a fully modularized environment. We will in turn explain the different concepts for modularization (Module system , C++ micro services and CTK Plugin Framework) that exist in MITK today, how medical data are retrieved and managed, the MITK Workbench and its rapid prototyping capabilities as well as further technical enhancements. Finally, we will describe parts of the software process we have established.

Extensibility and modularization

The growing scope of the toolkit and its increasing number of contributors require concepts ensuring the extensibility and modularization of the toolkit. A *module* in MITK is a C++ class library covering a specific problem domain. The employed concepts for managing and reducing dependencies of modules, improving encapsulation and fostering reuse of code within the toolkit will be presented in the following sections, constituting the cornerstones of subsequently

[9] http://www.volview.org/.

[10] http://www.analyzedirect.com/.

introduced features and techniques. These concepts enable MITK to comply with the technical requirement of *extensibility of data handling*.

The module system

On the build-system level the MITK *module system* provides facilities to manage inter-module as well as optional external dependencies, based on the following concepts:

Packages are third-party software packages that can be used together with MITK. Apart from ITK and VTK, all packages are optional, e.g. Qt,[11] Boost,[12] DCMTK,[13] OpenCV or SOFA. For each package a configuration option in CMake is created, `MITK_USE_<PACKAGE>`, that can be controlled by the user. For most packages there is also the option to download, configure, and build them automatically. Depending on the selected packages different modules are available.

Modules are created with a collection of CMake macros that offer several advantages to basic CMake commands:

- resolve (optional) dependencies to other modules and packages
- manage include paths, distinguish between internal and exported ones
- create configuration files that can be used by custom projects to facilitate the integration of MITK
- encapsulate low-level CMake code, so developers can create their own modules without knowledge of CMake and the build system is kept maintainable
- support unit testing and the creation of installers
- quality control, e.g. a `WARNINGS_AS_ERRORS` option to enforce a module to be free of compiler warnings

Using this module system, researchers in the medical imaging domain can extend MITK with their own modules with minimal effort. They can build upon a large collection of existing MITK modules and third-party packages.

C++ micro services

The extensive modularization of the code base leads to problems and anti-patterns that typically occur in any large and modularized C++ system [12]:

- different build configurations exist with different sets of provided functionality at runtime, e.g. by supporting different data formats
- an increased amount of factories and "manager objects" for rendering, interaction, or other central tasks are difficult to maintain. They are typically implemented as singletons or other static instances, possibly with inter-dependencies.[14]

We have developed a new approach to these problems: the C++ micro services. This is a C++ implementation of the service layer of the Open Services Gateway initiative (OSGi) , an industry-grade and mature dynamic module system originally designed for Java.[15] It provides mechanisms to gradually move to a service-oriented modular system. Since it does not depend on any other libraries it can be used even on the lowest toolkit level. Major features are

- type-safety for service interfaces and their implementations
- selection of services based on priorities and properties
- much less boiler-plate code for usage compared to typical factory implementations.

In addition to the general advantages of a service-oriented software architecture, MITK modules interfacing with hardware devices especially benefit from the usage of such a dynamic service layer. For example, tracking devices are represented by a common service interface and MITK modules provide implementations for specific devices by registering them as services. Connecting or disconnecting devices leads to the registration or un-registration of the corresponding service, and service properties reflect the current state of a connected device.

Though developed by the MITK project the C++ micro services library is independently available[16] and potentially useful for any modularized C++ project.

CTK plugin framework

The C++ micro services handle modularity on the toolkit level. On the application level the requirements go further. A modular application is composed of many components. These should be well separated from each other, preferably loaded or unloaded at runtime and should be allowed to extend each other.

We have designed and implemented a generic plugin framework to facilitate the development of service-oriented and modular systems. Since this can be used independently from other parts of MITK it was implemented as part of the *Common Toolkit* (CTK) and has therefore been named *CTK*

[11] http://qt-project.org/.

[12] http://www.boost.org/.

[13] http://dcmtk.org/.

[14] The static (de-) initialization order fiasco is well known to any maintainer of a large C++ software system.

[15] OSGi Alliance, http://www.osgi.org.

[16] http://cppmicroservices.org.

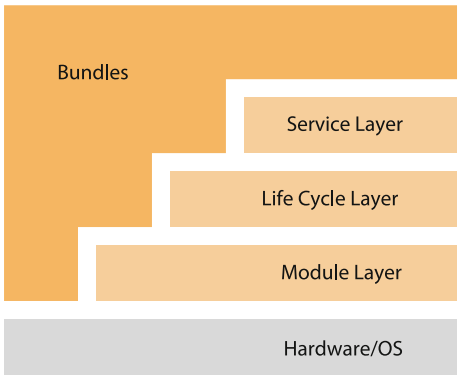

Fig. 2 The CTK Plugin Framework is conceptually divided into three layers, based upon the original OSGi specifications. The *Module layer* defines the units of modularization, a plugin meta-data format, a dependency management system, and a code sharing model. The *Life cycle layer* defines how the life cycle of plugins is controlled and is based on the module layer. Finally, the *Service layer* defines a collaborative model for plugins, integrated with the life cycle layer

Plugin Framework. CTK is a joint effort of several medical imaging groups that develop software platforms. It focuses on common-interest topics of the CTK community.[17]

Like the micro services it is a C++ implementation based on the OSGi specifications [18], but implements almost all specified layers (see Fig. 2).

So-called *bundles* are the physical units of modularization, and they have an associated life cycle which is controlled by the CTK Plugin Framework. They can be dynamically started and stopped, adding and removing services and code from the running system. In the CTK context, bundles are also often called *plugins*.

Medical data management

Data handling is of high importance for a seamless integration of software in clinical workflows. This includes basic data retrieval and handling, interfaces to *operating room devices* and handling of *live data*. These improvements cope with the requirements for intervention support and treatment control as described in "Introduction" section. Detailed technical documentation of these concepts can be found online.[18]

Data retrieval and handling

DICOM data retrieval and import: The low-level DICOM handling capabilities of DCMTK, GDCM, and ITK were improved to enable easy PACS retrieval and the local management of DICOM file collections as well as enhance the grouping of image slices for the correct assembly of 3D and 3D+t datasets from CT, MR, and Ultrasound. Part of this work was done in collaboration in the scope of the Common Toolkit.

Protected image access: Low-level image processing toolkits like ITK, VTK, and OpenCV often deal directly with C/C++ pointers to image data. Iterator concepts exist but can include a serious runtime penalty, removing the advantages gained through the usage of C++. In a larger environment the arbitrary or even concurrent access to image data can lead to unpredictable results. We introduced the concept of image accessors to control, read, and write access on a higher level while still offering direct pointers when necessary. This is comparable with semaphores controlling the access to shared memory regions. Another benefit is the possibility of removing, compressing, or otherwise processing image data on devices with limited capabilities and only providing the classic continuous memory representation of an image when it is actually accessed.

DataStorage: The original MITK concept of a DataTree was replaced with a more generic DataStorage, offering better methods to express semantic relationships between data entities in the application. *Predicates* can be used to find DataNodes based on the data type, relationships, and properties, all combinable with logical operations, e.g. "give me all DataNodes of type $binaryimage$ that are derived from node x!"

Handling of operating room-related devices and live data

Increasing intra-operative use of devices, such as endoscopes or tracking hardware, especially in image-guided therapy systems required extensions of MITK that provide support for live data acquisition and processing. The following parts were introduced for handling live data within MITK:

- OpenCVVideoSupport: Allows grabbing of video data from devices like endoscopes or video cameras and provides methods to access the video data in MITK-specific data formats. The third-party library OpenCV is used for hardware communication.
- MITK-IGT: Released as the image-guided therapy toolkit within MITK, it provides methods of communicating with tracking hardware, such as optical or electromagnetical tracking devices, processing of tracking data by means of a navigation pipeline and building navigation applications in a rapid prototyping manner. Devices generating navigation data are available as micro services.
- MITK-US: Adds support of intra-operative live ultrasound data. Similar to MITK-IGT, data processing can be easily implemented with a pipeline structure, and common GUI elements allow rapid prototyping of applications using US as intra-operative imaging modality.

[17] http://commontk.org.

[18] http://mitk.org/Documentation.

Again, US devices are made available as services, preventing dependencies on device specific implementation code.
– MITK-ToF: Handling of range data, such as Time-of-Flight (ToF) data, mostly used for intra-operative surface acquisition is supported through the ToF modules within MITK. Besides hardware communication for image acquisition, data processing is again possible via a pipeline concept and application development is simplified by reusable GUI elements. As in the MITK-IGT and MITK-US extensions, ToF devices are modeled as service objects.

Rapid prototyping

To support an efficient implementation of applications for clinical tasks like image analysis, diagnosis support, and treatment planning, we added rapid prototyping capabilities to MITK. The new capabilities were designed to enable *support for method and tool development* and *interactivity and usability* as detailed in the requirements for a software system in a clinical environment (section "Introduction").

MITK application framework

Originally MITK was not intended as an application framework and thus only contained basic support for building complete applications. It merely provided an example of how to build applications using MITK including a simple extension mechanism referred to as *functionalities*. It was purely a configuration/build time mechanism using a single class interface (`QmitkFunctionality`) and a few singletons for access to some central application objects. To overcome the limitations of this approach we created a C++ equivalent of the Eclipse Rich Client Platform [14]:[19] *BlueBerry*. Inheriting most concepts, the key features are as follows:

– A consistent and thoroughly designed user interface concept, leading to applications which by default have a uniform look and feel.
– A ready-to-use application frame for rapid prototyping.
– A modularized architecture based on the CTK Plugin Framework. BlueBerry applications are created by orchestrating a set of plugins, each of which contributes specific functionality.
– Communication mechanisms allowing sharing of information between unrelated plugins.

BlueBerry is build as a set of plugins, leveraging the service-oriented and dynamic nature of the CTK Plugin Framework. The block diagram in Fig. 3 gives a high-level system overview. Logically, BlueBerry can be divided into a *runtime system* that is independent of any user interface technology and the *workbench*, a generic application frame. Like Eclipse RCP, the workbench supports a consistent user interface design based on the concepts of views (control areas), editors (data viewing and editing areas), and perspectives. The standard configuration of the workbench allows the user to dynamically change the layout, to open and close views and editors as well as to detach them from the main application window. The concept of "perspectives" can be used to create predefined layouts for related tasks or restrict the possible layout changes by the user, e.g. for workflow-oriented applications. The BlueBerry runtime system can also be used as the base of custom applications, even for non-medical-imaging tasks.

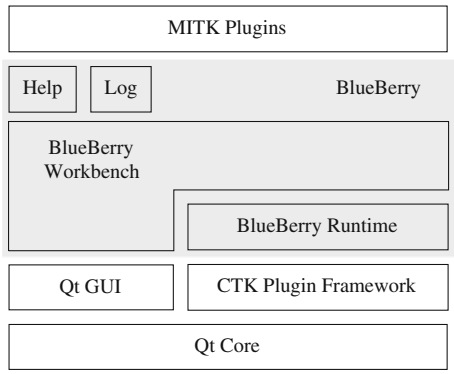

Fig. 3 BlueBerry is build on *top* of the CTK Plugin Framework and logically divided into a runtime system and a generic application frame called workbench

To allow for rapid prototyping and provide the developers with an easy way to create their own application-level extensions—either as a thin wrapper around some new algorithmic method or as a rich GUI for conducting some workflow—MITK contains a plugin generator. It allows the creation of a plugin with a GUI view that already contains some sample code to get access to an image the user has selected in the application. Using this tool a developer gets a quick start into adding own functionality to the MITK workbench, whether it is a new algorithmic method for segmentation or the start of a whole new application with its own interaction mechanisms and user interface concepts.

Python prototyping

Another rapid prototyping capability was introduced into MITK by creating language bindings of MITK and its included packages i.e. ITK, VTK, and OpenCV for the Python language [22]. The process relies on that of WrapITK [13], i.e. we employ the combination of CMake, GCC-XML, and CableSwig to create the language bindings during the

[19] The Eclipse Rich Client Platform is best known as the basis for the Eclipse IDE, http://wiki.eclipse.org/Rich_Client_Platform.

CMake generation process. A plugin containing an interactive Python console provided by CTK allows for convenient prototyping featuring, e.g. drag&drop of data from the DataStorage, code completion, a command history, a variable stack, and a script editor. As a result, images can for example be processed with Python using, e.g. ITK, VTK, or OpenCV code and be shown in the MITK application as usual.

Software development process

Building and maintaining a large software system is always a challenge. MITK started reusing all the tools from the Kitware process, e.g. as decribed in the ITK Softwareguide [9], and added several enhancements. Specific requirements for MITK resulted from the fact that its user and contributor base had very different levels of experience, from computer science students to experienced post-docs. At the same time a high level of software and process quality was necessary so MITK could serve as a basis for the development of medical products. Finally we faced the challenge of designing a release process that works in a predictable way despite the sometimes limited or unprojectable resources in a research environment. First results on this were described in [16], but they had to be improved to get established on a more regular basis.

Collaborative development

Large software systems with many contributors require concepts for a collaborative development model. For MITK, the following approaches were introduced:

Continuous integration and repository lock: The continuous integration was extended to enable a repository lock based on the dashboard status: In the dashboard driver script a simple call to a web service was implemented to submit the number of build errors and test failures via a http request. This web service aggregates the results from the clients and displays them on a web page similar to the official dashboard. The current dashboard manager can decide which clients "lock" the repository in case of a failure. These are usually the continuous clients for Windows, Linux, and Mac OS X. We implemented a automatic check in the source code repository that rejects all commits except for the ones that are explicitly marked as compilation fixes. This mechanism helps in a cross-platform and mixed-experience developer team to minimize times when the current trunk or master version does not build on all platforms. The locked repository also adds a social component, encouraging developers to fix their errors as soon as possible.

Version control and branchy workflow: To decouple the work on different topics a branchy workflow based on the *git* distributed version control system [4] was designed. Every feature or bug fix is developed in a separate branch in the version control system and can only be merged into the master if several checks are passed. This has the advantage that developers can exchange and manage their intermediate results using the central repository without interfering with each other.

Several server-side scripts were developed that check the intended changes before accepting them in the main repository. First, the naming scheme and the overall structure of branches as well as certain coding styles are verified. Further checks support the goals of the quality management and the release process that are described in the next sections.

Quality management

MITK is the basis for projects that require a general safety statement and is also used in several products that have a CE mark or FDA clearance as a medical product. Two obligatory standards are required for certification in the European Union: ISO 13485 for quality management systems and ISO 14971 for risk management. They specify mainly actions on the management level that focus on the actual product and general workflows and responsibilities during the production process. As such they are not specifically oriented to software development but have their roots in the production of medical device hardware of any kind, from surgical instruments to pacemakers.

More important and useful for the software developer is the relatively recent standard "IEC 62304—Medical Device Software—Software Lifecycle Processes", harmonized both by the European Union and the FDA. It specifies requirements for a process which is suitable for developing and maintaining a safe software system.

Traceability: One important requirement of IEC 62304 is traceability:[20] every change in the software shall be traceable to a change request, which in turn is based on a new feature request or a necessary bug fix. All feature requests and bugs are recorded in the MITK Bugzilla issue tracker.[21] To make sure that this policy is followed we implemented another pre-commit hook that checks the assignability of commits to tickets in the bug tracker. From the commit messages and

[20] IEC 62304, B.8.2 Change control: CHANGE REQUESTS can be approved by a change control board or by a manager or technical lead according to the software configuration management plan. Approved CHANGE REQUESTS are made traceable to the actual modification and VERIFICATION of the software. The requirement is that each actual change be linked to a CHANGE REQUEST and that documentation exists to show that the CHANGE REQUEST was approved. The documentation might be change control board minutes, an approval signature, or a record in a database.

[21] http://bugs.mitk.org.

the branch structure the bug id is derived and the Bugzilla database is checked for the correct bug status.

Change tracking: If the requested change affects files in the core libraries, there is another check for a flag in the database that the change has been approved by a technical lead developer. One criterion for approval is a written change request, filed in the MITK wiki[22] and directly linked from the ticket. A template prompts for the motivation for the change, the root cause of the problem, the proposed solution, and the intended verification measure. This acts as a documentation of corrective and preventive action (CAPA)—a well-known principle from good manufacturing practice.

Release process

MITK had irregular releases for a long time but with a growing number of external users, both from research and industry, the demand for fixed release versions has increased. Earlier attempts were successful but also showed a significant impact on the regular work of the developers [16].

The new release procedure is strictly date-based: at the beginning of the 3-month cycle the release day will be defined and different roles assigned to people:

- release manager: overall responsibility
- testing manager: distribution and collection of manual checklists, filing of bugs
- promoter: editing of changelogs, release notes, websites, upload of installers

Two weeks before that day the repository is put into release mode, which means that only master merges approved by the current release manager are allowed. From this point on manual checklists are completed, verifying software functions on the application level (MITK Workbench with basic plugins). Bugs are filed and classified by the testing manager. As many bugs as possible will be fixed; the remaining ones are documented as known issues and will be prioritized in the next release cycle. Since the master branch is locked, there is no need for an early release branch. The release branch will be created right after the final test and fix day and usually contains only updates to version numbers and small installer customizations.

Results

Since its initial release MITK has seen a constant growth regarding both the size of its code base (Fig. 4) and the number of contributors (Fig. 5). The introduction of the module system in the year 2009 led to a leaner MITK core responsible for the basic toolkit functionality. More specialized source code was moved to several newly created modules.

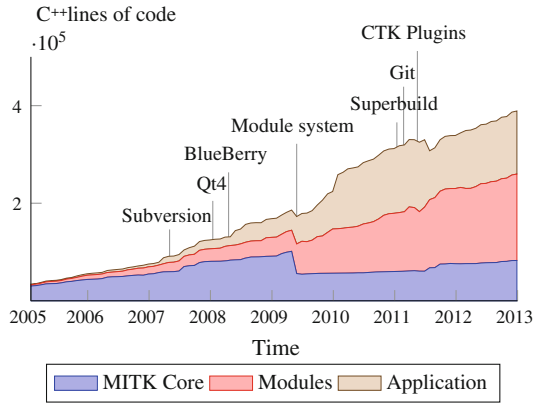

Fig. 4 Temporal development of the MITK code base with respect to the lines of source code. The code base is logically divided into three parts. The *MITK Core* contains the basic toolkit functionality, *Modules* represents domain-specific functionalities, and *Application* groups application-level code. Additionally, milestones in the MITK development are depicted, e.g. transitions to different version control systems or the introduction of new modularization techniques

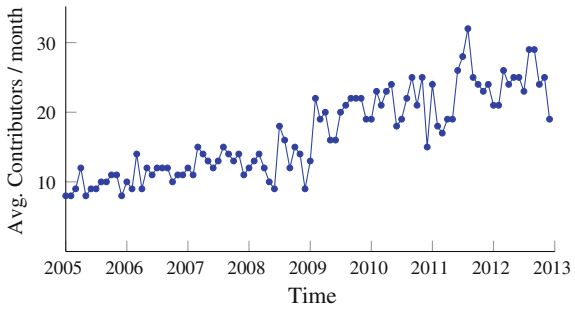

Fig. 5 Contributor count for the MITK. Over the last 8 years, a steady increase in contributors can be observed

The established **software development processes** have led to regular releases of the toolkit and various applications as open-source software. Furthermore, the branch-based workflow allowed to generally maintain a high-quality software with a growing number of contributors of various backgrounds and experience levels (Fig. 5). The acceptance in the open-source community is established, as evidenced, for example, by the number of posts on the MITK Users List, which reached slightly more than 2,000 over the past 2 years.

MITK offers the user different levels of support for his research:

- *use the MITK Workbench* and available plugins as a ready-made application, for example, to perform a study requiring semi-automatic segmentation or to record some live data from trackers during an intervention

[22] http://mitk.org/ChangeRequests.

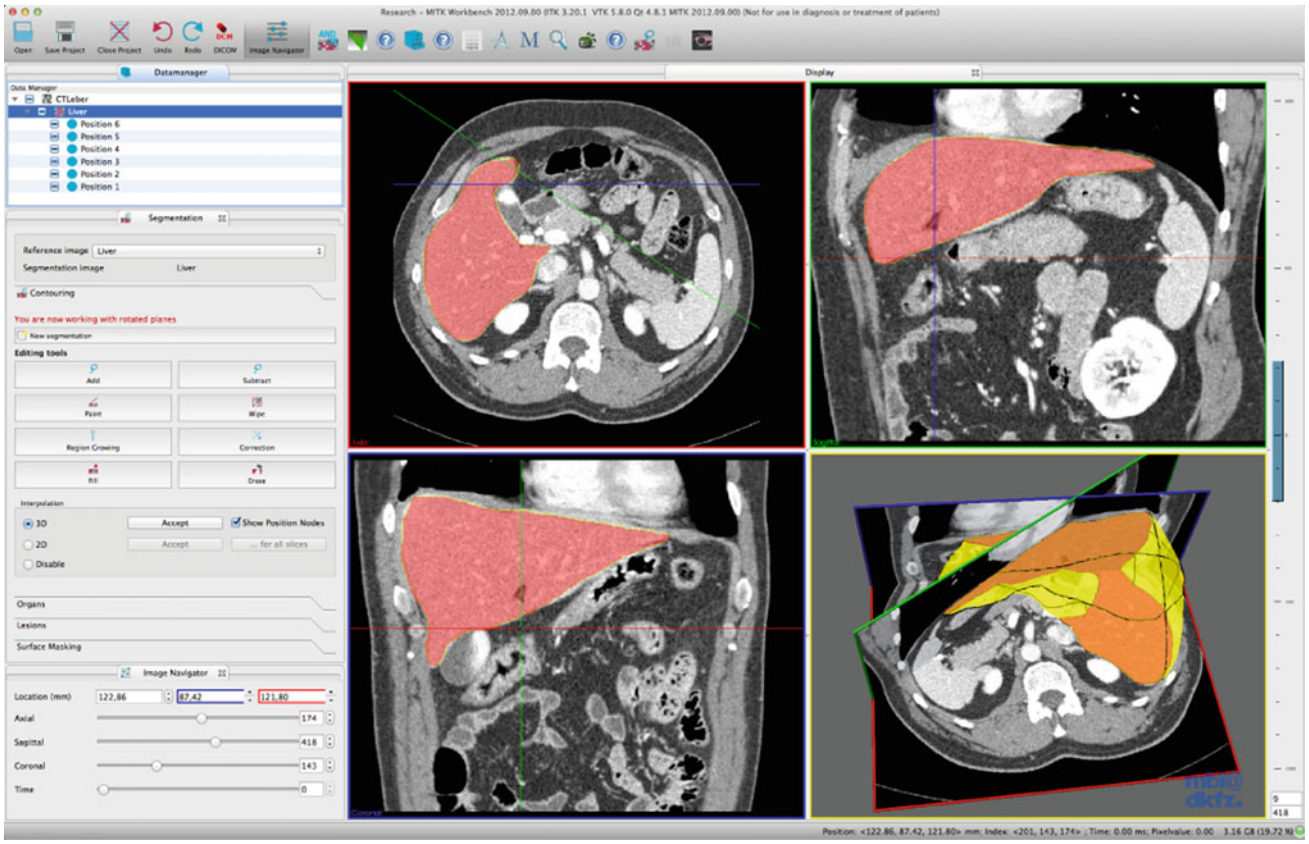

Fig. 6 The *MITK Workbench* exemplary shown for the use case of interactive image segmentation. The screenshot shows a typical configuration of several views on the *left* and an editor containing a multiplanar reconstruction (MPR) on the *right*

– *extend the MITK Workbench* as a platform and extend it with own processing methods and interactive workflow tools, reusing the integrated plugins for, e.g. connectivity to clinical system or data analysis
– *integrate MITK as toolkit* to leverage some of its functionality, e.g. data handling, visualization, or standardized interfaces to special devices like trackers or video systems. If necessary, enhance the toolkit by custom data formats, visualization types, tracking devices, etc.

These different approaches are possible through the modularization and extension methods described in "Extensibility and modularization " section. For example, the *C++ micro services* have been used inside the MITK extensively for the image-guided therapy (IGT), ultrasound (US), and Time-of-Flight (ToF) extension modules. Service interfaces were defined for the different data providers, for example a tracking service. Implementations of this interface, for example for NDI (Waterloo, Canada) or Claron (Toronto, Canada) trackers can register themselves as a service and allow application-wide access to the relevant devices.

The feasibility to develop applications for the various clinical tasks identified in the introduction has been shown by implementation of several applications which will be discussed in the following paragraphs. For more projects realized with MITK within the medical imaging domain please refer to see http://mitk.org/Projects for details.

Data retrieval and **basic image analysis** were made widely available with the development of the MITK Workbench (Fig. 6) which is based on the *CTK Plugin Framework* and the BlueBerry application framework. As an environment for easily creating medical image processing applications it comes with a defined set of basic plugins to perform basic image review and analysis:

– DICOM Import and Retrieval
– Volume visualization
– Measurement and image statistics
– Interactive segmentation tools, slice-based and 3D

Around 30 more plugins exist in the open-source distribution of MITK and can be enabled at build time, offering various image processing tools or reference implementations for the usage of specialized modules, e.g. IGTTracking [6] or the Time-of-Flight tutorial [24].

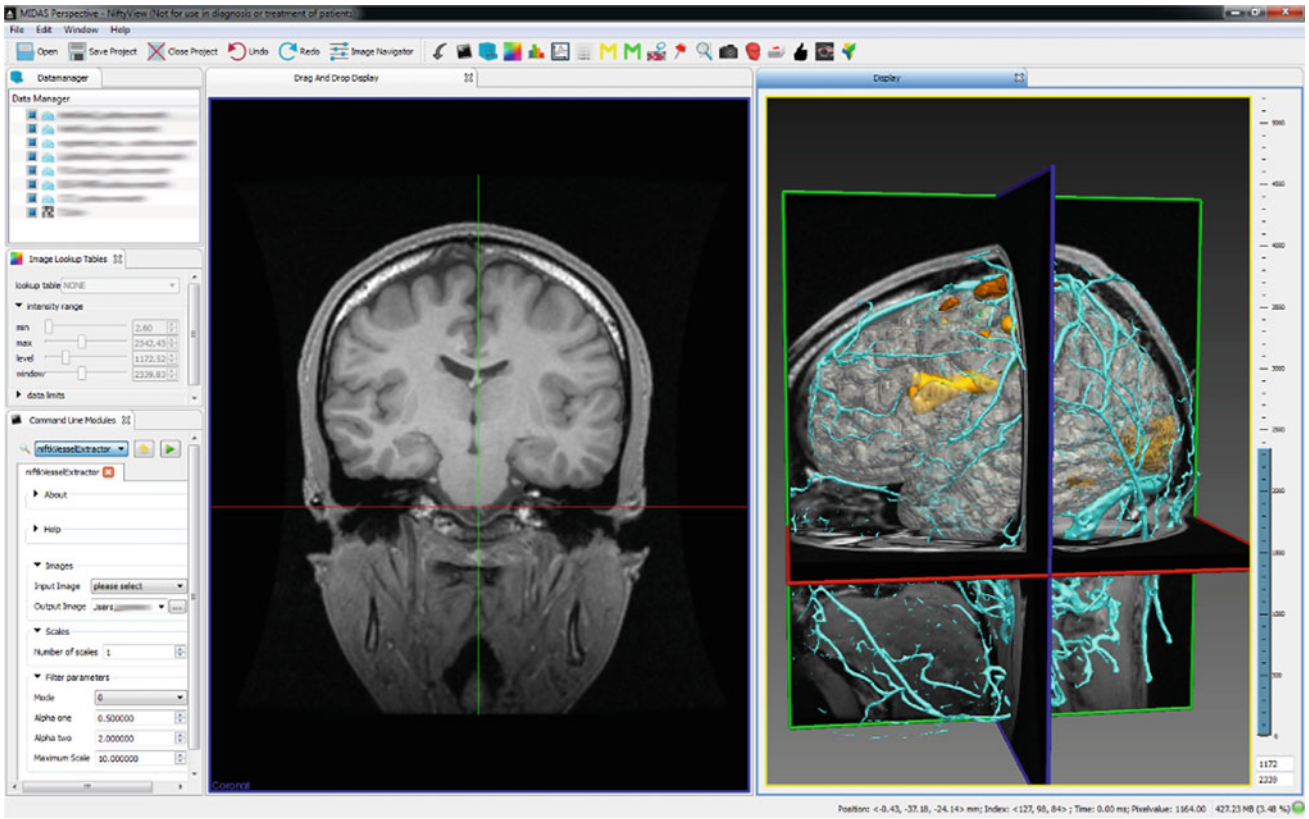

Fig. 7 Research Environment *Nifty View*, Image kindly provided by Matt Clarkson, The Center of Medical Image Computing, University College London, UK

Building upon the MITK Workbench several other projects have created complete applications:

Image analysis and diagnosis on diffusion-weighted MR images can be performed with *MITK-Diffusion*, developed by Fritzsche et al. [7]. It is a customized Workbench-based application with a large collection of plugins, directly aimed at radiologists for research on diffusion-weighted MR images. It is available from the MITK homepage or through NITRC,[23] a public repository for neuroimaging tools and resources [11]. The underlying image processing methods are available separately as open-source MITK modules.

The Center for Medical Image Computing, University College London, UK, is using the MITK Workbench to create NiftyView, a working environment for their clinical collaborators[24] (Fig. 7), featuring a variety of analysis methods.

A **treatment planning** application for 3D ultrasound-guided prostate biopsies has been realized by Eigen, Grass Valley, CA. It is a 510(k) cleared software based on the MITK Workbench (Fig. 8) that adds a planning solution to their 3D ultrasound-guided prostate biopsy platform.[25]

One example of an **intervention support** system is the MITK-based Computer Aided Medical Diagnosis And Surgery System (CAMDASS), developed by Space Applications Services S.A., (Zaventem, Belgium), in cooperation with the European Space Research and Technology Center (ESTEC) (Noordwijk, the Netherlands), evaluating the feasibility to support astronauts during space flight performing medical procedures [17]. More applications are described, e.g. in [2,8,23] and [15].

Besides the MITK Workbench-based applications there exist projects that directly build on top of the toolkit MITK. $mint Lesion^{TM}$ for example is a certified application allowing for **treatment control** developed by the Mint Medical GmbH, Dossenheim/Heidelberg, Germany. It supports radiologists and oncologists in assessing tumor measurements according to several standards, such as RECIST, WHO, and Choi. It is certified as a medical product in the European Union and the United States and uses various MITK modules for image data handling, measurements, and visualization (Fig. 9). Enhancements and bug fixes are regularly contributed back to the open-source MITK.

The *Graphical Interface for Medical Image Analysis and Simulation* (GIMIAS)[26] is a research environment using

[23] http://www.nitrc.org/projects/mitk-diffusion/.

[24] http://cmic.cs.ucl.ac.uk/home/software/.

[25] http://www.eigen.com/.

[26] http://gimias.org/.

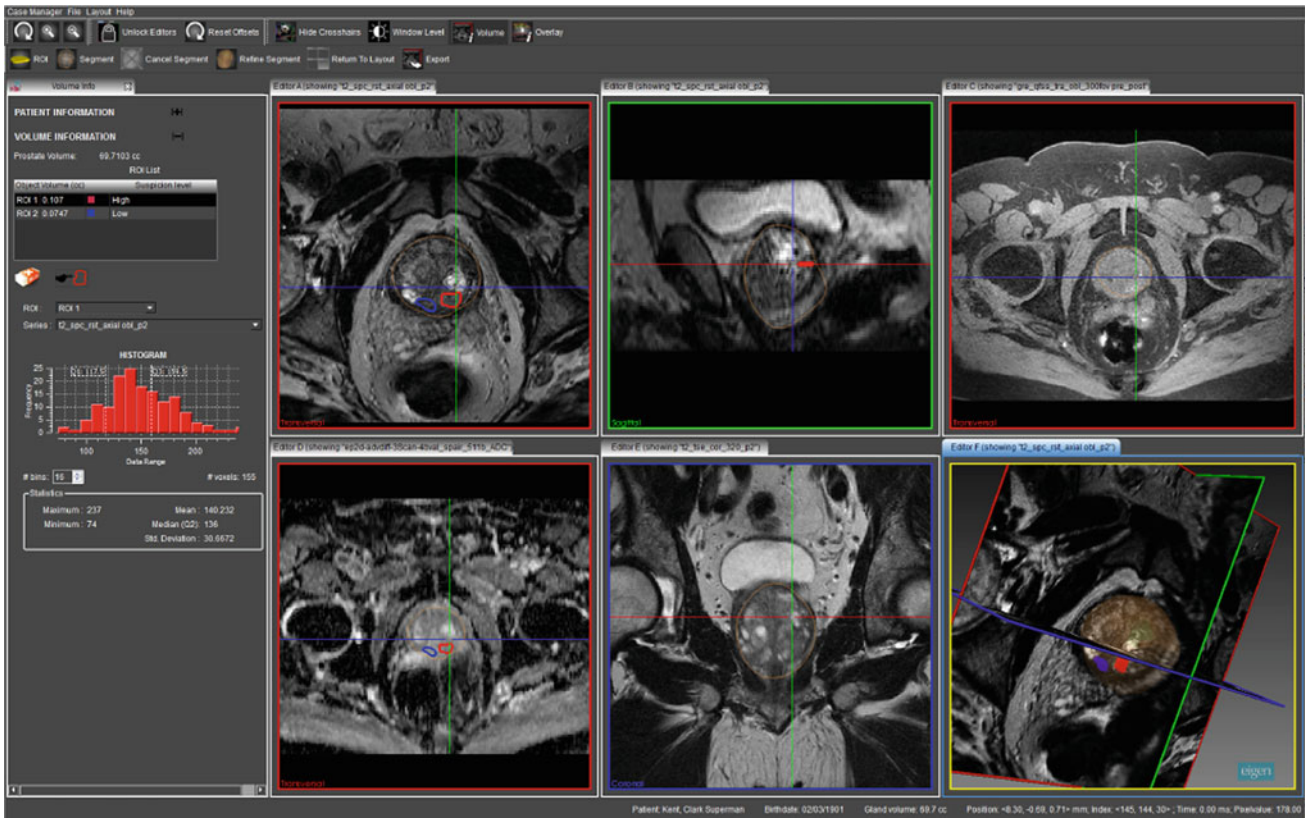

Fig. 8 510(k) cleared application *ProFuse*, used for planning of 3D ultrasound-guided prostate biopsy. Image kindly provided by Eigen, Grass Valley, CA, USA

MITK (Fig. 10). It is being developed by the Center for Computational Image and Simulation Technologies in Biomedicine (CISTIB) at Universitat Pompeu Fabra, Barcelona, Spain. It shows the flexibility of the MITK approach by using wxWidgets instead of Qt as a GUI toolkit, combining its own application framework with the MITK modules.

Discussion

Medical image processing research has become increasingly demanding over the past decades due to the introduction of new imaging modalities, the growing number of algorithms available for processing the mass of imaging data, more complex clinical workflows, and the trend to intra-operative use of image analysis methods. Meeting the requirements for efficiently conducting research in this interdisciplinary environment is a challenge for many existing software environments.

The MITK presented in this work met these requirements by introducing novel modularization concepts and an application frame suited for both rapid prototyping and sustainable development. The connectivity to clinical storage systems was improved, and interfaces to acquire live data from devices in the operating room were created. The flexible user interface of the MITK Workbench with its configurable layout, using multiple views and screens, allows a quick adaption to various clinical workflows. Finally, the general software robustness was increased. MITK proved useful in the commercial development of intervention planning and treatment control solutions and carries the aspects of quality management for medical devices into the software development process onto the toolkit level, something that is not very common in the open-source community. Furthermore, in combination with the modularization it is possible to manage software parts of different quality levels in one system, fostering technology transfer from experimental research to medical products.

Development of medical imaging software was shown to be supported by MITK on different levels. Applications using MITK at the toolkit level typically make use of its data management, visualization, and data processing capabilities by accessing the low-level interfaces. In this case many complex tasks like creating a suitable user interface or combining the functionality of existing MITK modules have to be tackled by the application developer. At the next level, the MITK Workbench provides an extensible application framework for creating custom medical imaging applications. This

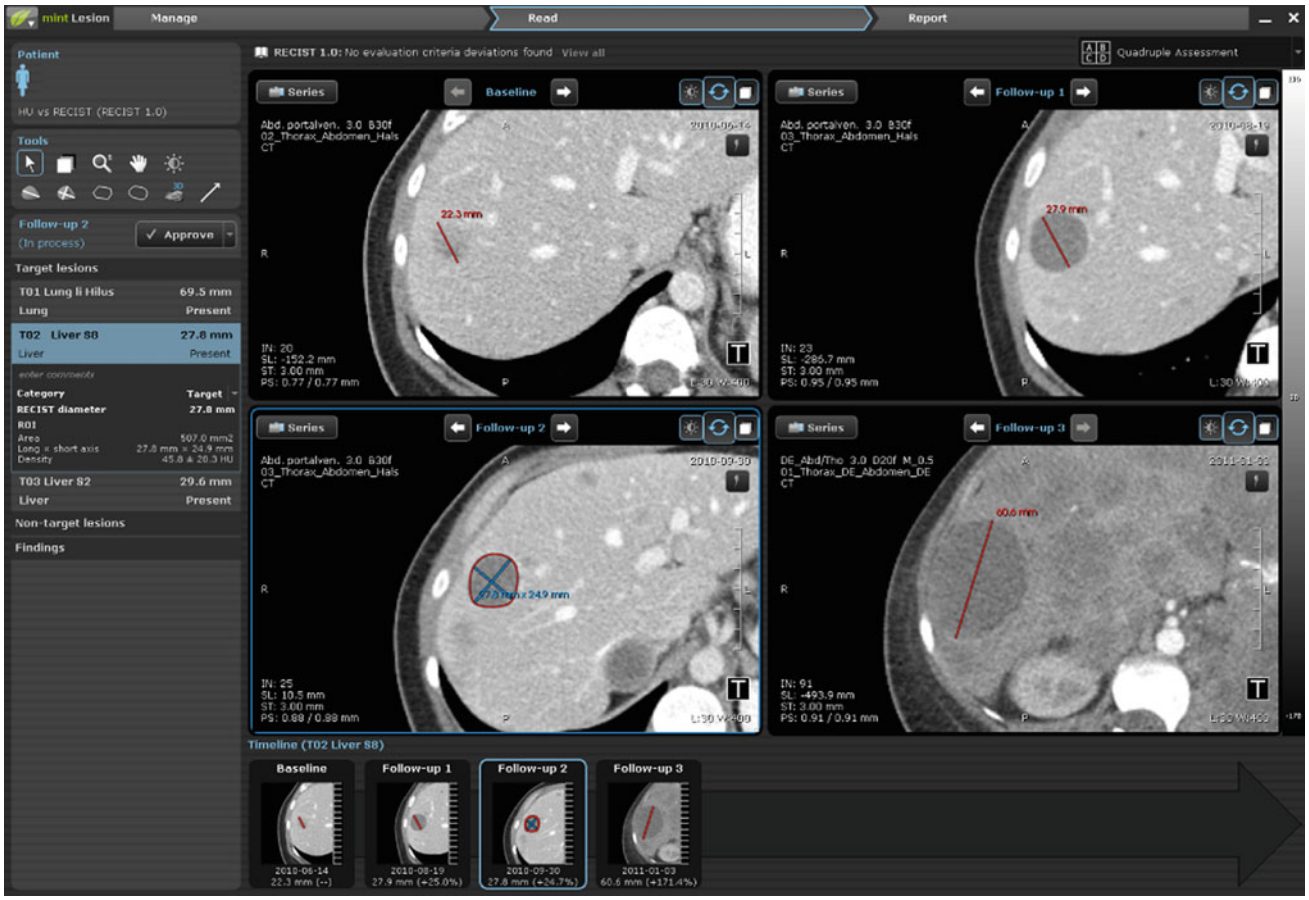

Fig. 9 FDA/CE-certified product *mintLesion*^{TM} for assessment of tumor measurements according to standards such as RECIST, WHO, and Choi, using MITK for image data handling, measurements, and visualization. Image kindly provided by Mint Medical GmbH, Dossenheim/Heidelberg, Germany

is the preferred way to rapidly develop software prototypes by reusing many of the existing MITK plugins. Its flexible and customizable user interface makes it well suited for usage in different clinical settings. Even without own development the MITK Workbench can be used as a research tool to conduct or support different studies. The online documentation features several examples and howtos to find the best suited approach.

Current technical solutions in a clinical setting often lack the usability that is necessary to get accepted by the physicians. Novel approaches like the use of today's powerful mobile devices open up new possibilities to create human–computer interfaces that fit seamlessly into the existing clinical workflows. In [15] we already showed that MITK is a well-suited environment to conduct research in this area.

As briefly—and by no means exhaustively—described in the introduction, many other software systems exist that support medical imaging researchers in their tasks. They all have their strengths, their right to exist and established user communities, which means ongoing projects and successful solutions. Because of this a mere comparison of features and performance figures would not make much sense in this context. We nevertheless want to identify a few exemplary keypoints in a comparison to other solutions:

Proprietary solutions like Matlab, Mevislab, and Analyze are powerful and widespread, but restricted in their usage. Some are free-as-in-beer to use but not *open-source*: the researcher is subjected to vendor lock-in or license changes. The publication of own results as open-source is possible but the verifiability by other scientists can be limited for the same reasons.

OsiriX is especially well accepted by radiologists for its exceptional usability. But since it is not *cross-platform* it also locks the user, and its licensing policy (GPL) limits the developer.

OpenXIP was released cross-platform and, because it is based on an industry-grade prototyping platform, has a strong feature set. However, only the library is open-source, while the visual programming tool is closed-source though free to use, and to date there is no visible public *community*

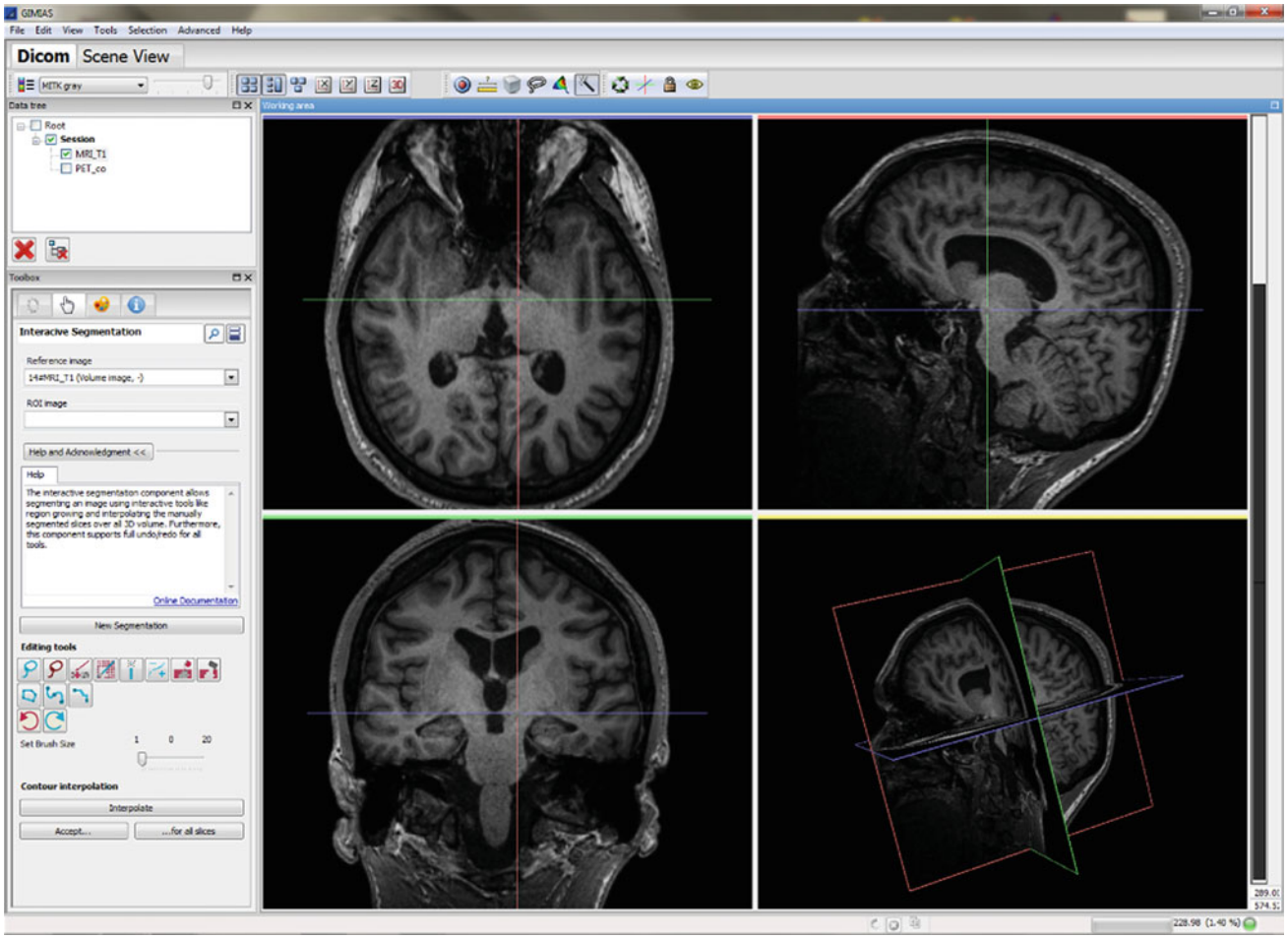

Fig. 10 *Graphical Interface for Medical Image Analysis and Simulation (GIMIAS)* using MITK at toolkit level

supporting the project, which makes it difficult for a potential user to evaluate.

The 3D Slicer is cross-platform and open-source and features a flexible and mature plugin system. The community is very large and active. Projects built upon the Slicer platform cover the whole range of medical image processing and intervention support. Compared to MITK its approach is generally more application-centric though this has been recently addressed by the possibility of running modules on their own without the application main window.

Since all these "issues" are, on the whole, signs of different philosophies there is no possible conclusion of *A-is-better-than-B-because-of-X*. The direct comparison of features of the different solutions may not be appropriate. But as communities and foci of research change, there will be a fluctuation of existing and new environments, all having their individual strengths. Solving everything on one's own or reinventing the wheel again and again is not a viable solution. The combination of strengths should be the goal.

First efforts in this direction have been made by developers of MITK, 3D Slicer, MAF3, medInria, GIMIAS, and other research software environments[27] by contributing to the *Common Toolkit* initiative in order to make as much of their work as possible available and usable for others. This is an important step in the right direction, leveraging work from others on the platform level, enhancing methods of interoperability between platforms and thereby opening new opportunities for technically sustainable collaborations in the research community.

It is our strong belief that modularization and interoperability of existing successful solutions help to meet the challenging tasks of today's and tomorrow's clinically motivated research. The open-source MITK, from its toolkit-level to the extensible application frame, has been designed in this spirit.

Acknowledgments We wish to thank the contributors to MITK, which cannot all be listed here. There have been more than one hundred over the time, more than fifty active ones in the last twelve months, thank you! Special thanks to Matt Clarkson for last minute proof-reading!

[27] http://www.commontk.org/index.php/The_Team.

Conflict of interest None.

References

1. Allard J, Cotin S, Faure F, Bensoussan P, Poyer F, Duriez C, Delingette H, Grisoni L (2007) SOFA: an open source framework for medical simulation. In: Medicine meets virtual reality (MMVR 15)
2. Baumhauer M, Simpfendörfer T, Stich BM, Teber D, Gutt C, Rassweiler J, Meinzer HP, Wolf I (2008) Soft tissue navigation for laparoscopic partial nephrectomy. Int J Comput Assist Radiol Surg 3:307–314
3. Bradski G, Kaehler A (2008) Learning OpenCV: computer vision with the OpenCV library. O'Reilly, Ireland
4. Chacon S (2009) Pro git. Apress, New York City
5. Enquobahrie A, Cheng P, Gary K, Ibanez L, Gobbi D, Lindseth F, Yaniv Z, Aylward S, Jomier J, Cleary K (2007) The image-guided surgery toolkit IGSTK: an open source C++ software toolkit. J Digit Imaging 20(Suppl 1):21–33. doi:10.1007/s10278-007-9054-3
6. Franz AM, Seitel A, Servatius M, Zöllner C, Gergel I, Wegner I, Neuhaus J, Zelzer S, Nolden M, Gaa J, Mercea P, Yung K, Sommer CM, Radeleff, BA, Schlemmer HP, Kauczor HU, Meinzer HP, Maier-Hein L (2012) Simplified development of image-guided therapy software with MITK-IGT. In: SPIE medical imaging 2012: image-guided procedures, robotic interventions, and modeling, vol 8316, p 83162J (8 pages). doi:10.1117/12.911421
7. Fritzsche KH, Neher P, Reicht I, Bruggen T, Goch C, Reisert M, Nolden M, Zelzer S, Meinzer H, Stieltjes B (2012) Mitk diffusion imaging. Methods Inf Med 51(5):441–448
8. Gergel I, Tetzlaff R, Meinzer HP, Wegner I (2011) An electromagnetic navigation system for transbronchial interventions with a novel approach to respiratory motion compensation. Med Phys 38:6742–6753
9. Ibanez L, Schroeder W, Ng L, Cates J (2005) The ITK software guide, 2nd edn. Kitware, Inc. ISBN 1-930934-15-7
10. Ince DC, Hatton L, Graham-Cumming J (2012) The case for open computer programs. Nature 482(7386):485–488. doi:10.1038/nature10836
11. Kennedy DN, Haselgrove C, Buccigrossi R, Grethe JS (2009) Software development for neuroimaging: promoting community access and best practices through nitrc. In: ISBI. IEEE, pp 1146–1149
12. Lakos J (1996) Large-scale C++ software design. Addison-Wesley professional computing series. Addison-Wesley. http://books.google.de/books?id=AuMpAQAAMAAJ
13. Lehmann G, Pincus Z, Regrain B (2006) WrapITK: enhanced languages support for the insight toolkit. Insight J 1
14. McAffer J, Lemieux J, Aniszczyk C (2010) Eclipse rich client platform. Eclipse Series. Pearson Education. http://books.google.de/books?id=fbxdpDTeELoC
15. Müller M, Rassweiler MC, Klein J, Seitel A, Gondan M, Baumhauer M, Teber D, Rassweiler JJ, Meinzer HP, Maier-Hein L (2013) Mobile augmented reality for computer-assisted percutaneous nephrolithotomy. Int J CARS 1–13. doi:10.1007/s11548-013-0828-4
16. Neuhaus J, Maleike D, Nolden M, Kenngott HG, Meinzer HP, Wolf I (2009) A quality-refinement process for medical imaging applications. Method Inform Med 48(4):336–339. doi:10.3414/ME9232
17. Nevatia Y, Chintamani K, Meyer T, Blum T, Runge A, Fritz N (2011) Computer aided medical diagnosis and surgery system: towards automated medical diagnosis for long term space missions. In: 11th symposium on advanced space technologies in robotics and automation (ASTRA). esa
18. OSGI Alliance (2009) OSGi Service Platform, core specification, release 4, version 4.2. Technical report, OSGI Alliance
19. Parker SG, Johnson CR (1995) SCIRun: a scientific programming environment for computational steering. SC conference 0, 52. doi:10.1109/SUPERC.1995.66
20. Pieper S, Halle M, Kikinis R (2004) 3D Slicer. In: IEEE international symposium on biomedical imaging: from Nano To Macro, pp 632–635
21. Rosset A, Spadola L, Ratib O (2004) OsiriX: an open-source software for navigating in multidimensional dicom images. J Digit Imaging 17:205–216. doi:10.1007/s10278-004-1014-6
22. Saruji D, Müller M, Meinzer HP (2011) Schnelles Prototyping für die medizinische Bildverarbeitung. In: Handels H, Erhardt J, Deserno T, Meinzer HP, Tolxdorff T (eds) Bildverarbeitung für die Medizin, pp 199–203. Lübeck, Germany
23. Seitel A, Engel M, Sommer CM, Radeleff BA, Essert-Villard C, Baegert C, Fangerau M, Fritzsche KH, Yung K, Meinzer HP, Maier-Hein L (2011) Computer-assisted trajectory planning for percutaneous needle insertions. Med Phys 38(6):3246–3259
24. Seitel A, Yung K, Mersmann S, Kilgus T, Groch A, dos Santos T, Franz A, Nolden M, Meinzer H, Maier-Hein L (2012) MITK-ToF: range data within MITK. Int J Comput Assist Radiol Surg 7(1):87–96
25. Wolf I (2011) Toolkits and software for developing biomedical image processing and analysis applications. In: Deserno TM (ed) Biomedical image processing, biological and medical physics, biomedical engineering. Springer, Berlin, pp 521–544. doi:10.1007/978-3-642-15816-2_21
26. Wolf I, Vetter M, Wegner I, Böttger T, Nolden M, Schöbinger M, Hastenteufel M, Kunert T, Meinzer HP (2005) The medical imaging interaction toolkit. Med Image Anal 9(6):594–604. doi:10.1016/j.media.2005.04.005