Journal of Digital Imaging (2020) 33:708–721
https://doi.org/10.1007/s10278-019-00304-1

ORIGINAL PAPER

iBEX: Modular Open-Source Software for Digital Radiography

Altay Brusan[1] · F. Aytaç Durmaz[1] · Alper Yaman[2] · Cengizhan Öztürk[1]

Published online: 16 December 2019

Abstract
A device-independent software package, named iBEX, is developed to accelerate the research and development efforts for X-ray imaging setups such as chest radiography, linear and multidirectional tomography, and dental and skeletal radiography. Its extension mechanism makes the software adaptable for a wide range of digital X-ray imaging hardware combinations and provides capabilities for researchers to develop image processing plug-ins. Independent of the X-ray sensor technology, iBEX could integrate with heterogeneous communication channels of digital detectors. iBEX is a freeware option for preclinical and early clinical testing of radiography devices. It provides tools to calibrate the device, integrate to health information infrastructure, acquire image, store studies on local storage, and send them to Picture Archiving and Communication System (PACS). iBEX is a unique open-source project bringing X-ray imaging devices' software into the scope of the open-source community to reduce the X-ray scanners' research effort, potentially increase the image quality, and cut down the production and testing costs of radiography devices.

Keywords Medical imaging · Radiography · X-ray imaging · Open-source software

Introduction

projectional radiography device is an assembly of a power source, X-ray tube, collimator, positioning mechanisms, X-ray sensitive receptor, and dose control tools. The power source generates a suitable range of electrical energy and conducts it to the X-ray tube to generate an X-ray beam. A collimator at the window of the X-ray tube housing influences the size and shape of the X-ray beam exiting the tube assembly and directed toward the patient body. Within the body, X-ray photons may be attenuated or directly pass through. The latter photons are called primary photons and positively contribute to the image formation at the detector. However, attenuated photons have the chance to find a way out and bring unwanted contributions at the detector. These photons are called scatter photons and are filtered by a specifically designed grid. A detector captures the received photons and forms an image [1], in which each pixel intensity is directly related to the amount of the incident X-ray photons.

The parameters of early projectional radiography scanners were manual and adjusted by an operator using electromechanical knobs and switches, but in time, devices have adopted such that the software's role in automation and control of the processes has increased. In 1970s, high-frequency (HF) power generators were introduced which outperformed three-phase power generators in terms of efficiency and reliability [2]. In this type of generators, an embedded software in a microcontroller was responsible for generating high-frequency switching signals, which were applied for controlling the output voltage. Indeed, it was the first appearance of a software unit within a radiography device. During the 1980s, electromechanical timers and knobs started to be replaced by digital and electronic counterparts [3]. By the early 1990s, a form of an embedded software component was presented in almost all radiography devices. At this decade, digital revolution completely changed the radiology devices, which result in (1) introduction of digital X-ray detectors, (2) significant reduction in storage costs, and (3) proliferation of digital communication infrastructures in medical centers. These advancements all together create a new ecosystem. The radiography devices need to be upgraded to fit to the new environment.

✉ Altay Brusan
altay.brusan@boun.edu.tr

F. Aytaç Durmaz
aytac@pievision.com

Alper Yaman
alper.yaman@ipa.fraunhofer.de

[1] Institute of Biomedical Engineering, Bogazici University, Kandilli Campus, Feza Gursey Bld., 34684 Cengelkoy-Istanbul, Turkey

[2] Fraunhofer Institute for Manufacturing Engineering and Automation IPA Department of Biomechatronic Systems, Stuttgart, Germany

Despite the successful history of the embedded solutions, they were not enough for these new demands. Due to this issue, embedded digital hardware was replaced by general-purpose computers, and software tools were developed to handle the necessary requirements and controlling tasks.

From then, researches are focused on optimizing the digital radiography systems, specifically with the aim to acquire better images from lower X-ray dose. As a result, receptor technology and material [4, 5], collimator design [6], acquisition protocol guidelines [7], and image processing and filtering [8] were all subject to further optimization. Also, studies were conducted to compare different imaging modalities [9] or image quality of the same modality from different vendors [10] and to update the standards [11].

These studies have highlighted the fact that despite a long history of the projectional radiography, it is still an active field of study with continuously emerging dynamic requirements. However, manufacturers develop their own hardware and accompanied software packages as a closed system, and some vendors limit access to critical sections within their software products. This approach is attributed to the tightly regulated ecosystem of the medical device environment. Figure 1 shows a common set of standards that are related to medical software.

A medical software, either as a part of a device or standalone medical application, is required to comply with several regulations, like US Food and Drug Administration (FDA) regulations on electronic records and electronic signatures, CGMP-CFR 21 Part 11 [12], or European Medicines Agency (EMA) 93/42/EEC directives [13]. A set of standards were prepared for software products: IEC 62304 (Medical Device Software – Life Cycle Processes) [14], which prescribes software engineering-related matters, and ISO/IEC/IEEE 29119 (software testing) family, which defines software testing standards that are usable in any development methodology; specifically, ISO/IEC/IEEE 29119-5 [15] promotes Keyword-Driven Testing (KDT) [16] as an efficient and consistent solution for evaluating software units. IEC 14971 [17] discusses a risk management framework within which the safety critical properties are hold and guarantees conformance to abstract specification of safe device operation. Additionally, there are standards that are not specific to software alone, but organizations should provide evidences of conformance, like ISO 13485 (Quality Management System). Companies that successfully pass this pathway for their products prefer to close their achievements for external intervention, and this effectively limits the flexibility for researchers.

In opposite to the closed-form manufacturing paradigm, various kinds of dedicated X-ray devices could be brought together, from off-the-shelf components made by different manufacturers. For example, a power source from vendor A and a detector from vendor B may be used to make a new X-ray scanner device. This assembling paradigm liberates the researchers from prefabricated device limitations and allow for customization of the radiography device based on their own needs and requirements. The only issue with the new paradigm is the lack of a device-independent and general-purpose software package which could interact with the components from different vendors. Such a software package would complete the chain of the customized X-ray device fabrication, and its requirements have already been defined by the American College of Radiology (ACR), the American Association of Physicists in Medicine (AAPM), and the Society for Imaging Informatics in Medicine (SIIM) [18].

The purpose of this study is to provide a multitude of modular open-source software tools, which is named iBEX, to facilitate the workflow necessary to build a functional digital radiography system. Its plug-in mechanism paves the way for integrating devices from different vendors and developing custom image processing algorithms. In order to reduce the platform compatibility issues, iBEX was written in C++, and

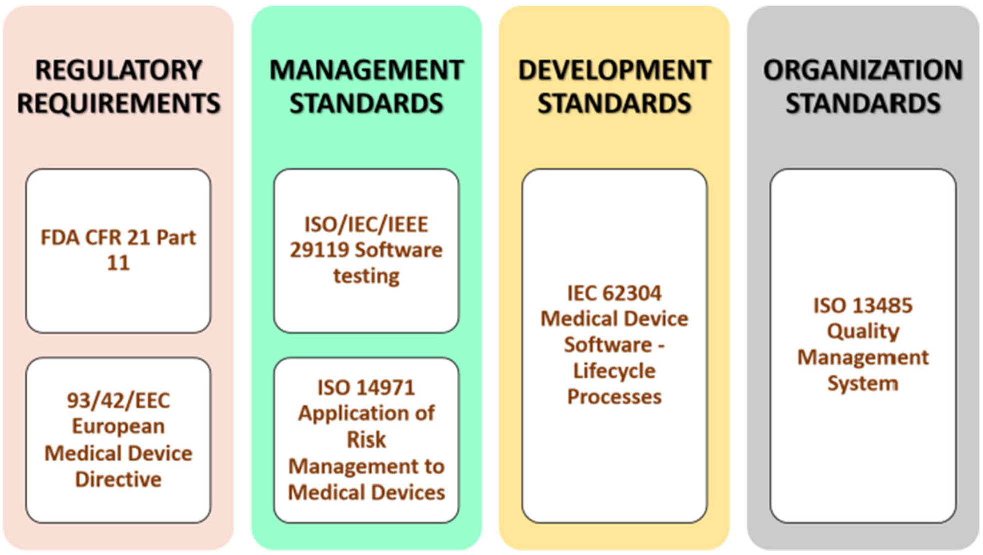

REGULATORY REQUIREMENTS
FDA CFR 21 Part 11
93/42/EEC European Medical Device Directive
MANAGEMENT STANDARDS
ISO/IEC/IEEE 29119 Software testing
ISO 14971 Application of Risk Management to Medical Devices
DEVELOPMENT STANDARDS
IEC 62304 Medical Device Software - Lifecycle Processes
ORGANIZATION STANDARDS
ISO 13485 Quality Management System

Fig. 1 FDA- and EMA-standardized medical software development, management, and integration activities

it is portable to any state-of-the-art operating system and hardware architecture (i.e., ×86 or 64-bit system). Researchers and developers could download the source code from the project GitHub repository [19] and rebuild it for their own scanners.

SOLID principles [20] and Model-View-Controller (MVC) design patterns [21] were applied during iBEX development, rendering the output source code neat, safe, and efficient. These goals were evaluated by means of automated testing tools and profiling widgets. Furthermore, iBEX functionality and reliability were assessed on an innovative full-body time delay integration (TDI)-based X-ray scanner.

In the following sections; iBEX architecture and major workflows are presented, and then its novel extension methods are discussed in detail. In the results section, iBEX analysis results are presented. Finally, at the discussion section, the iBEX, current position, and its future in the medical open-source development are discussed.

Materials and Methods

This section reviews iBEX internal from normal users and technical developers' perspectives. Normal users get benefit of main workflow and high-level functionalities of iBEX solely for imaging and device calibration purposes without dealing with technical details behind. On the other hand, technical developers access the full iBEX code. These users are normally responsible for developing, installing, updating, configuring, and customizing iBEX for a new imaging device.

The project development follows IEC 62304 standard [14]. This standard discusses a generic application lifecycle management (ALM) for medical software and its minimum required criteria, without involving methods and realization techniques. Any methodologies that meet these minimums could be acceptable. Among the available options, Rational Unified Process (RUP) [22] has been selected for iBEX realization. First, the requirement analysis document of the project was prepared. Based on that document, the project was divided into three milestones. Then, a design document of the first milestone was sketched out. In an iterative and incremental process, the requirements document, design artifacts, and prepared source code were gradually updated. This process continued until all targets of a given milestone were achieved. Then, the next milestones were started and continued till all the requirements were satisfied. In this section, the focus was on the final outputs and the details of each iteration were omitted.

Requirement Analysis

The requirement analysis started from ACR-AAPM-SIIM guideline [18] and Digital Imaging and Communications in Medicine Information Object Definition (DICOM IOD) of digital radiography [23]. Based on these guidelines, a list of required workflows was identified; then, the list was enriched by some of the other workflows that are commonly available in modern digital radiography scanners (Table 1). This list was the base of all analysis, implementation, and test designs.

System Design

iBEX had a micro-kernel architecture (Fig. 2). All major workflows (Table 1) and a plug-in manager were implemented within the core package.

Internally, iBEX core contained VTK [24], DCMTK [25], CTK [26], Qt [27], Log4Qt [28], and SQLite [29] packages. Each package had a specific responsibility within the kernel. VTK library was used for image rendering, DCMTK was applied for DICOM transactions and file format conversion, and CTK has built-in plug-ins and tools for managing PACS connection and provided extra Qt widgets for handling and viewing images. The plug-in manager was based on Qt plug-in engine and log-engine leverages Log4Qt package. Finally, SQLite was applied for managing database.

Implementation Concerns

iBEX code was developed based on well-known best practices and guidelines. The source code clarity and reusability were boosted by applying separation of concern principle [30] and implementing MVC design pattern. The view objects were responsible only for interacting with the user interface, model objects stored and retrieved data, and controller objects handled the data flow in between view objects and model objects.

The events were divided into systemic and application categories. Each category is logged separately. Systemic events include technical details and mostly designed for debugging iBEX itself. The application events observe users' interactions like the last time a user signed in. The log files are stored in a rolling fashion in text format, and when their sizes reach a predefined amount (current limit is 10 MB), they are overwritten.

DICOM standard has a database model [31] for DICOM compatible medical X-ray scanners. In this model, each *patient entity* accomplishes at least one *study entity*, each study contains zero or more *series entities*, and each series contains zero or more *instance entities* (i.e., image). iBEX database implements this model with solidifying two design choices: first, all the patient's imaging tasks are completed in one session. So, each study contains one and only one series. Second, each instance (i.e., image) is independent of the others and stored in a separate file.

Extension Mechanisms

Plug-in manager of iBEX core supports two types of plug-ins: *device* and *algorithm*. The device plug-in provides a mechanism to integrate a device driver, for example, a detector

Table 1 Digital radiography common requirements

Workflow	Description
Register new patient	Capture the patient's demographic information and record it in the local database
View image	Load local image file (with tiff or dcm extension) and display it on the screen
Select imaging task	Opens the local task list to select a task
Quick scan	Starts to acquire a study without registering the patient. It is useful for emergency scans
Configure devices	Updates parameters of components (such as detector binning mode or energization level of power source)
Apply image filtering	Updates the images by applying a filter plug-in (one could also make changes on the filter parameters before its application)
Manage local database	Checks the database for data integrity and controls the capacity
Decide on image	After acquiring an image, the user can decide to accept or reject the image. If the image was rejected, then the acquisition is expected to be repeated
Log event	Events within the system are recorded in different levels (e.g., warning, information, error, fail)
Send to PACS	The finished studies are stored in a remote PACS server. The server connection settings should be configurable
Configure machine	Peripheral components' calibration parameters are updated
Make DICOM file	Converts the raw image into a DICOM file
Create account	User credentials are updated
Update worklist	Modality worklist is automatically (and manually) updated

driver, and algorithm plug-in is for merging new image processing algorithms into iBEX. As it is shown in Fig. 3, IDevice and IAlgorithm interfaces provide the common language in between iBEX core and external plug-ins. During iBEX core boot period, plug-in engine automatically searches for the files that implement one of these interfaces and loads them as extension plug-ins.

The IDevice interface standardizes the commands and messages that is sent and received in between iBEX core and peripheral devices (Fig. 4). This interface identifies three types of operations for each device: *read*, *write*, and *execute*. To accomplish each of these commands, the plug-in may use a setting value. IDevice could discriminate error and normal messages. These notifications are fired from the device plug-ins and contain the situation information. They are captured and handled within iBEX core. Some device plug-ins may have a user interface to display on the screen. iBEX core, by means of the "Get Widget" function, checks whether there is any user interface widget within the IDevice plug-in or not. If there is any, then it would load and fuse it to its own interface during the boot time. Additionally, IDevice interface discriminates volatile and permanent settings. Volatile settings are

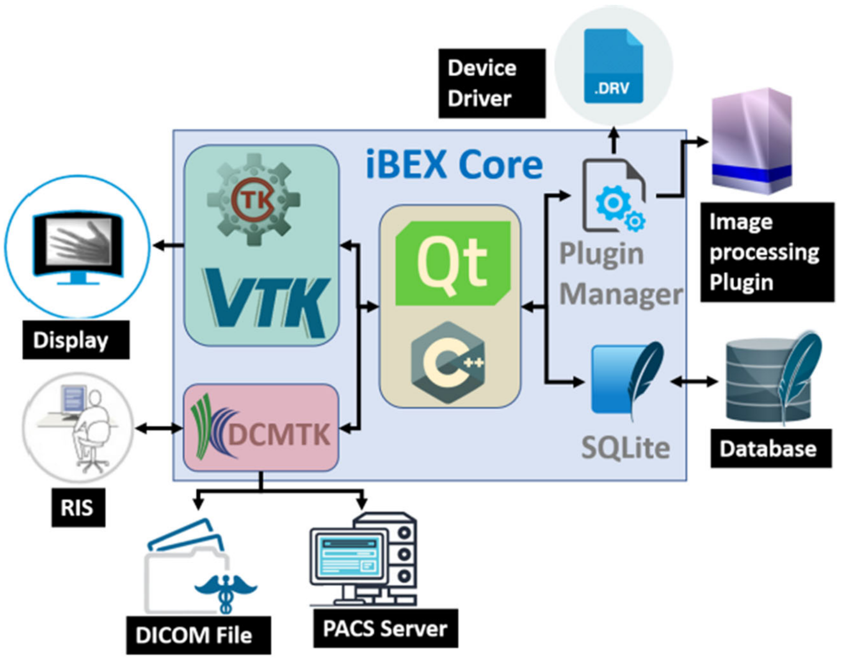


Fig. 2 iBEX core package is the kernel of the system. It includes all major workflows and a plug-in manager

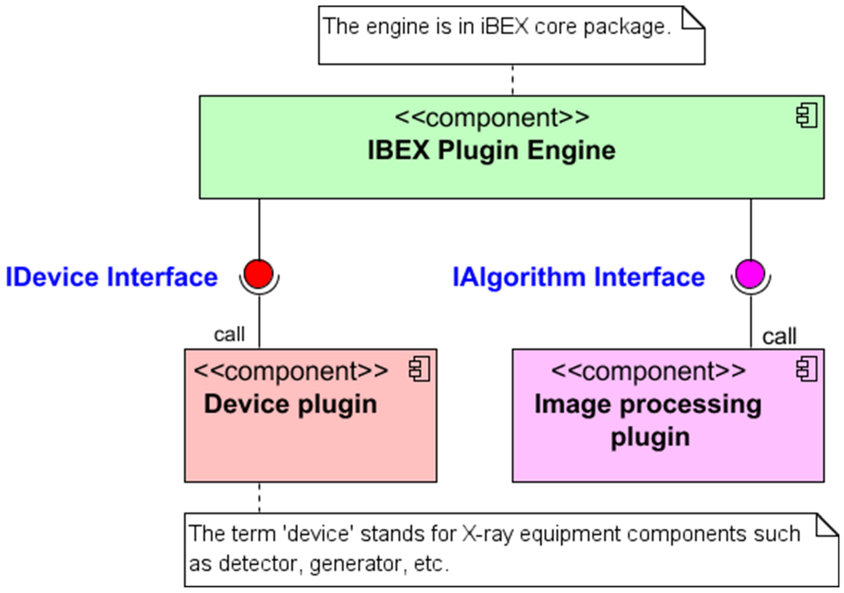


Fig. 3 iBEX extension mechanisms support device and algorithm extensions via IDevice and IAlgorithm interfaces

parameters restricted to the current session and disappear when the session is ended (or due to power loss). On the other hand, the permanent settings are stored within the device firmware, and they are not cleared even after the session is finished. To handle this latter type of settings, iBEX core requests a parser in order to help with permanent settings. Some permanent settings, like detector gain and offset, could critically affect the device performance. Manufacturers prefer to block the end user access completely or intentionally to make it hard to manipulate those calibration parameters. However, if the user is authorized and provided a way to update the critical calibration parameters, then it is feasible to change them by means of an IDevice plug-in.

The IAlgorithm interface is specialized for image processing (Fig. 5). iBEX core passes the raw images to an IAlgorithm filter via "Set Input" command. The processed image is reloaded back to the iBEX core with "Get Output." The core can trigger the algorithmic operation of the plug-in with "Start" and "Stop" command and through the "Update Parameter" informs the IAlgorithm plug-ins with the most updated filtering parameters. The plug-ins could inform the operation status to the iBEX core via firing notification messages (i.e., Finish, Error, Stop, and Start), and the iBEX core handles these messages appropriately. Like IDevice, IAlgorithm plug-ins may contain a graphical interface widget and, likewise, the "Get Widget" function returns this interface from the plug-in into the core.

IDevice and IAlgorithm interfaces intrinsically differ from each other in the way that they handle the commands. It is reasonable to assume that the device commands are executed in relatively shorter time than algorithmic operations, which may take a while to complete. Based on this, IDevice operations (read, write, and execute commands) are conducted synchronously, while IAlgorithm operations (start and stop commands) are asynchronous. In other words, whenever iBEX core issues a device operation command to a device plug-in, it blocks itself and waits until the command is finished or a

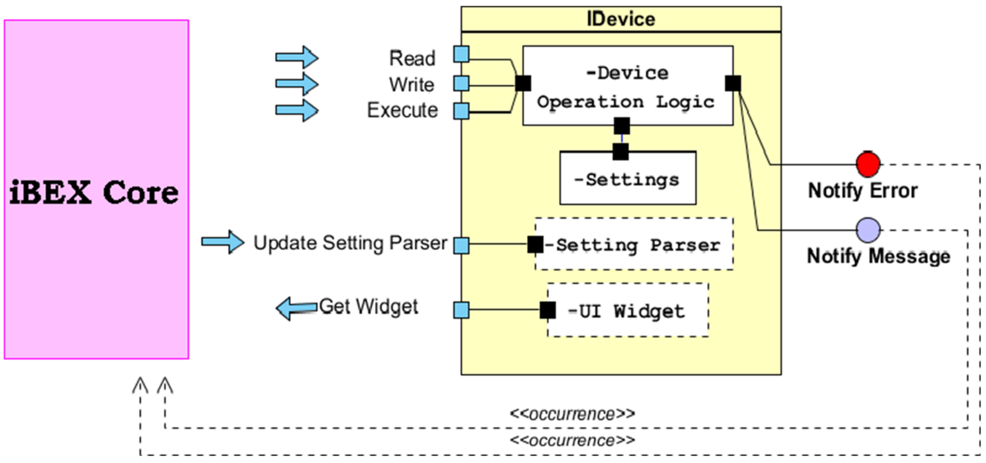


Fig. 4 IDevice interface. This interface describes a common context in between iBEX core and a device plug-in

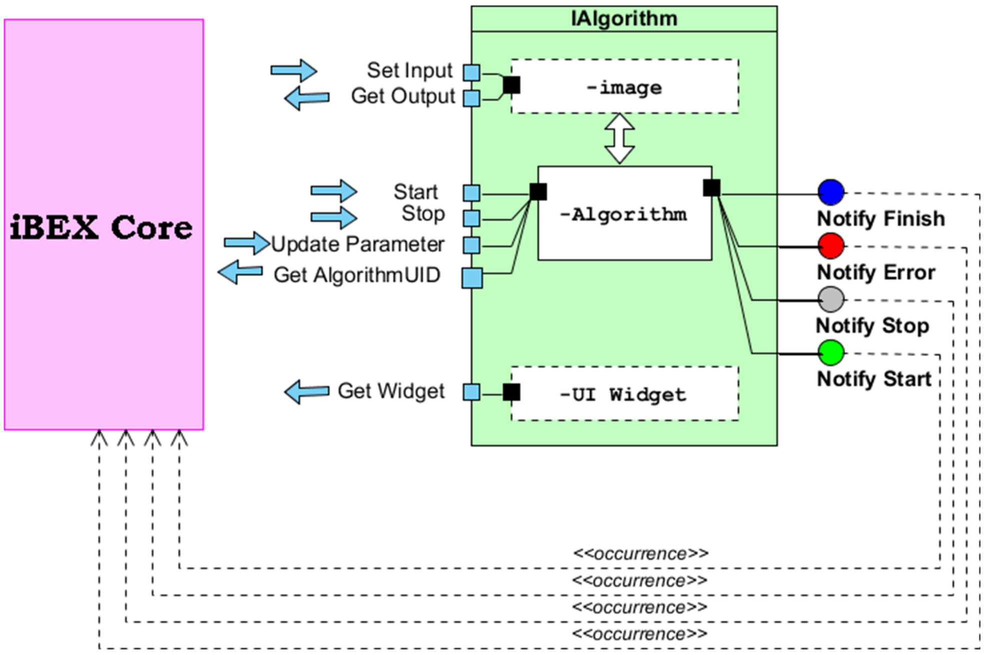
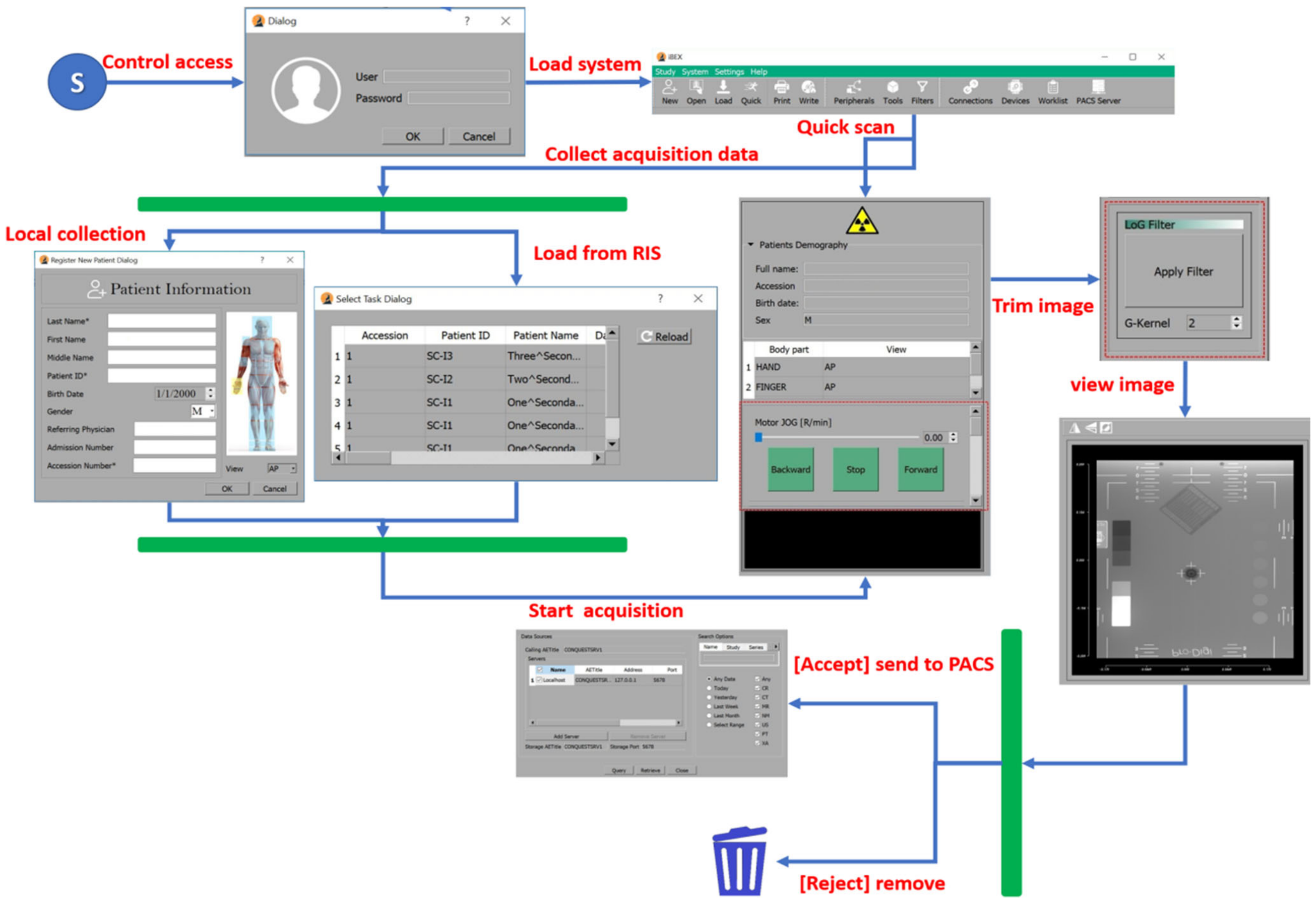


Fig. 5 IAlgorithm interface. Image, algorithm, and UI widget are IAlgorithm properties. iBEX core interacts with a filtering plug-in via a set of functions defined in the interface

notification message is returned. On the other hand, when iBEX core starts (or stops) a filtering task within an IAlgorithm plug-in, it immediately returns to its own execution thread. It is the plug-in's responsibility to inform the core with appropriate notification messages, such as start, stop, progress, or error notifications.



	Accession	Patient ID	Patient Name	Da
1	1	SC-I3	Three^Secon...	
2	1	SC-I2	Two^Second...	
3	1	SC-I1	One^Seconda...	
4	1	SC-I1	One^Seconda...	
5	1	SC-I1	One^Seconda	

Fig. 6 iBEX internal main workflow. This workflow is used by normal users

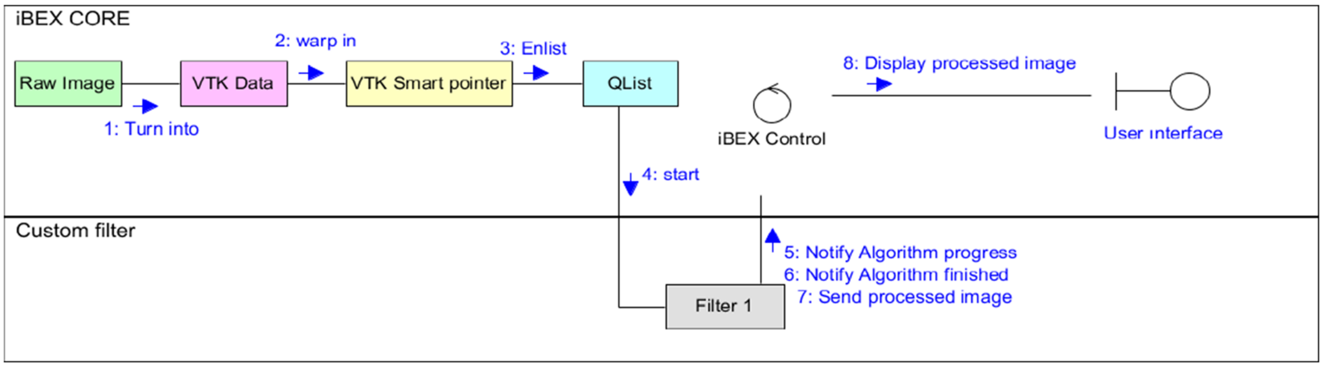


Fig. 7 Data flow pipeline between core and first custom filters. If another filter exists, then iBEX automatically activates. The final filter output is displayed on the screen

Main Workflow

iBEX main workflow is shown in Fig. 6. After successful log-in to the system, the main menu shows up. Each item on this menu targets a section of the common workflow (Table 1). iBEX provides three methods to start a study:

1. *Register a new patient*: the patient's DICOM critical demographic information is first collected and registered at the local database.
2. *Fetch from Radiology Information System (RIS)*: imaging tasks and patient demographics are fetched from the RIS system.
3. *Quick operation*: imaging task is performed with no need to collect demographic details. This is useful for emergency and anonymous studies.

The "Examination dialog" controls the examination exposure parameters and flows. After successful acquisition, image processing task is initiated via the filters' widget. The processed image is shown in the image viewer section of the "Examination dialog," and if it is acceptable, then it is sent to

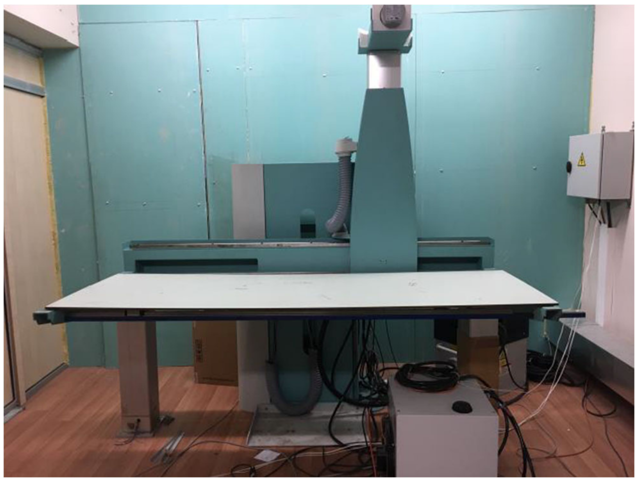
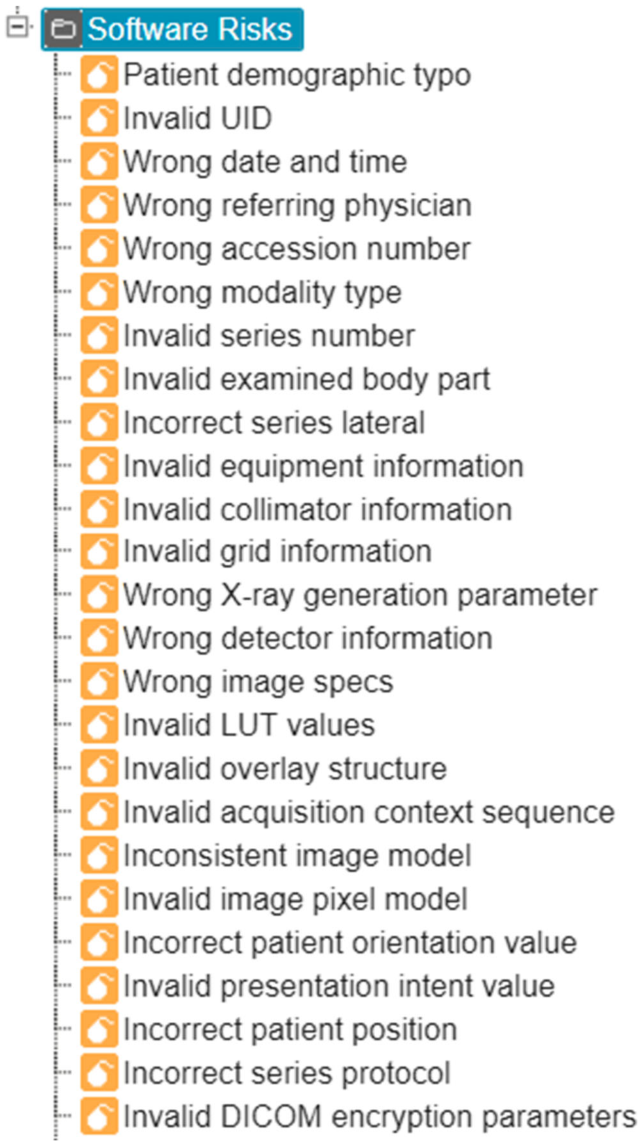


Fig. 8 Developed full-body X-ray scanner. In this device, a TDI detector, high-frequency power generator, and three servo motor drivers are required to be controlled with iBEX



Software Risks
- Patient demographic typo
- Invalid UID
- Wrong date and time
- Wrong referring physician
- Wrong accession number
- Wrong modality type
- Invalid series number
- Invalid examined body part
- Incorrect series lateral
- Invalid equipment information
- Invalid collimator information
- Invalid grid information
- Wrong X-ray generation parameter
- Wrong detector information
- Wrong image specs
- Invalid LUT values
- Invalid overlay structure
- Invalid acquisition context sequence
- Inconsistent image model
- Invalid image pixel model
- Incorrect patient orientation value
- Invalid presentation intent value
- Incorrect patient position
- Incorrect series protocol
- Invalid DICOM encryption parameters

Fig. 9 Risks in association with software. These risks could negatively affect the device functionality

Table 2 Risks FMEA matrix before applying any risk reduction mechanisms

Likelihood						
	6 frequently	**1**	**1**			
	5 probable	**2**	**22**		**23**	
	4 occasional			**30**	**10**	**13**
	3 remote	**19**	**16**		**5**	**4**
	2 improbable	**13**				
	1 unbelievable	**2**				
		1 negligible	**2** marginal	**3** critical	**4** very critical	**5** catastrophic
		Severity				

PACS server; otherwise it is eliminated. Other workflows such as updating PACS and RIS servers' connection settings or registering new users are also available but are not shown in Fig. 6.

Filtering Pipeline

The digital X-ray images are large and require a delicate flow pipeline to prevent memory leakage or system crash. To optimize the data transfer in between iBEX core and IAlgorithm plug-ins, the raw images were encapsulated in multi-layer data structure (Fig. 7).

First, the raw images were converted into raw "vtkImageData" type. This data type is consistent with the other image processing and filtering tools of the VTK library. Then, to activate the VTK memory management, the "vtkImageData" was wrapped with a smart pointer and then enlisted in Qt list data structure. This list is passed to the first filter plug-in. When it is finished with processing, it notifies the core with the filtered image. This flow is repeated for the next filter until all filtering tasks are finished. Then iBEX core fetches the processed image from the last filter and displays the output on the screen.

Tests, Risks, and Measures

In harmony with IEC 14971, risk management activity is placed underneath of the project management discipline. Risks and their associated harm were studied before any piece of code was written. Then, candidate routines, practices, and control mechanisms that could avoid or reduce harm severity were devised and reported on failure mode and effects analysis (FMEA) document – as a part of analysis-and-design artifact. By this effort, iBEX would always stay within a tolerable harm zone.

In accordance with ISO/IEC/IEEE 29119-5 standard, iBEX tests were accomplished based on two-layer KDT model: *domain layer* and *test interface layer*. Each domain layer keywords associate with a group of tests at interface layer and represent the highest abstraction level. However, test interface layer keywords stand for the most atomic interactions and specify the lowest level of abstraction.

After development, the released executable was tested on an innovative full-body TDI X-ray scanner that we had developed (Fig. 8). In this device, one Gulmay CF-series power generator through a serial port, a Delta ASD-b2 motor driver through a MODBUS RTU channel and one Teledyne Argus TDI detector through an ethernet channel were connected to the workstation computer. Additionally, on the workstation there were a PACS server [32] and a RIS emulator [33] to simulate real environment. This case study is used to profile iBEX runtime resource management and measure the code coverage [34]. To best of our knowledge, no medical software standards suggest rigid boundaries, methods, or criteria for quantitatively measure the medical software quality. However, commonly used criteria such as count of activated functions [35], number of passed conditions [35], amount of used memory, CPU peak and GPU load were measured for iBEX.

Table 3 Risks FMEA matrix after applying risk control mechanisms

Likelihood						
	6 frequently					
	5 probable					
	4 occasional					
	3 remote	**3**	**5**	**23**		
	2 improbable	**18**	**52**	**8**	**13**	
	1 unbelievable	**16**		**14**		**1**
		1 negligible	**2** marginal	**3** critical	**4** very critical	**5** catastrophic
		Severity				

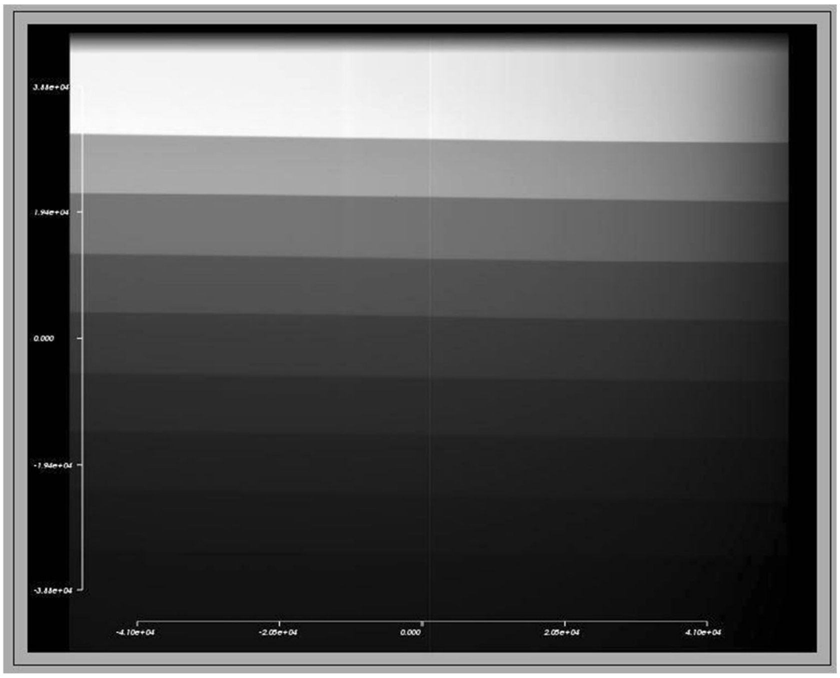


Fig. 10 Sample output of the step phantom

Results

Using the common requirements (Table 1), iBEX was logically organized into a set of domains, and for each domain, a keyword was assigned. Respectively, each domain is divided into atomic operation units and tested individually. A summary of these tests is available in appendix (Table 5). Some tests are accomplished by automated testing tool, marked as AUTO, and the others are done manually, labeled MAN. Except a few warning messages, majority of tests passed successfully. The reason of the warnings was in the fact that the VTK is in transition from openGL to openGL2 backend technology and there is inconsistency in between VTK and CTK widgets. Also, VTK Qt widget plug-in works just in release mode compilation, and this causes additional warnings during the tests.

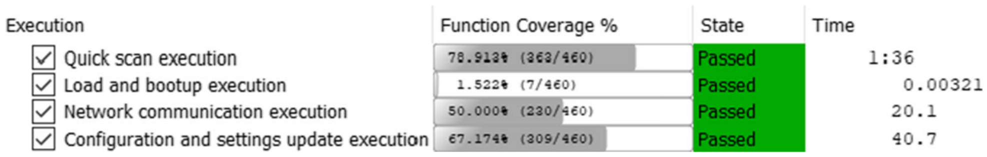
The project risks root in different factors such as software related, device component originated, human errors, or environment. A partition of risks that were classified to be in association with software is shown in Fig. 9.

In collaboration with medical experts, all identified risks were categorized in failure mode and effects analysis (FMEA) matrix (Table 2). Red blocks are unacceptable zones, and the associated risks could cause severe harm on the patient or frequently happened. This type of risks requires tighter security measures during the software design. For example, the operator may enter a wrong X-ray generation parameter which leads to harmful over (or under) shoot exposure on the patient. To minimize this risk, (1) all exposure parameters on the user interface should have measurement unit, (2) the font and its size should be legible, (3) the parameters' order on the screen should be static such that it reduces the chance of human mistakes, (4) there should be hard-coded boundaries on exposure parameters, and (5) ask the operator to verify the exposure before starting examinations. Risks within the orange blocks are tolerable. For example, invalid look-up table (LUT) values may change the image appearance on the screen; however, most DICOM viewers have an option to manipulate LUT boundaries, and physicians can adjust them on their device before deciding. Additionally, a verified DICOM encoder, like DCMTK, takes care about this issue. Finally, the green area shows the zones within which the risks happen remotely or their severity is not critical. Example of this kind of risks is invalid grid information. This type of information is used to augment the DICOM file with device specification and does not carry therapeutic or diagnostic value.

Table 4 Memory, CPU, and GPU utilizations reflect the runtime resource management quality criteria

Snap	Memory (mb)	Peak CPU(%)	GPU utilization(%)
Baseline	22	17	2.8
PACS transaction	25.42	14	4.8
RIS transaction	25.43	11	3.2
Patient registration	26.34	4	3.7
Quick scan	231.88	25	5.2

Fig. 11 iBEX function coverage on prototype device. Number of iBEX functions involved in execution depends on the execution path



Execution	Function Coverage %	State	Time
✓ Quick scan execution	78.913% (363/460)	Passed	1:36
✓ Load and bootup execution	1.522% (7/460)	Passed	0.00321
✓ Network communication execution	50.000% (230/460)	Passed	20.1
✓ Configuration and settings update execution	67.174% (309/460)	Passed	40.7

The following strategies were contemplated to reduce harm:

• Apply standard compatible tools such as DCMTK.

• Require the user to verify steps.

• Hard code DICOM-predefined constants within the source code.

• Design user interface to reduce the chance of miss interpretation.

• Provide detailed documentations regarding device technical specification.

Based on the expert bodies' judgment, the above approaches could reduce the red zone risks' severity or likelihood and put them within the tolerable boundaries. The FMEA matrix of iBEX after risk management activity is shown in Table 3. One can see there are no risks within intolerable region.

The *functionality* and *reliability* of iBEX were evaluated on a novel TDI full-body scanner. For each component, a device plug-in was developed and a denoising filter implemented to reduce the image noise. Sample output of a step phantom is shown in Fig. 10.

A summary of the runtime resource consumption is given in Table 4. After successful log-in and before activating any process, a snapshot of allocated resources is captured. This snapshot constitutes the base line, and during the execution, other snapshots were captured to compare the resource consumption over the time. A sudden memory and CPU peak was observed in quick-scan snapshot. This is because the image processing and device communication happened simultaneously, and they are resource-hungry processes. Comparing CPU and GPU utilizations clearly depicts that most of the processing tasks are done by CPU rather than GPU. This is because the fact that VTK and CTK widgets were compiled for CPU.

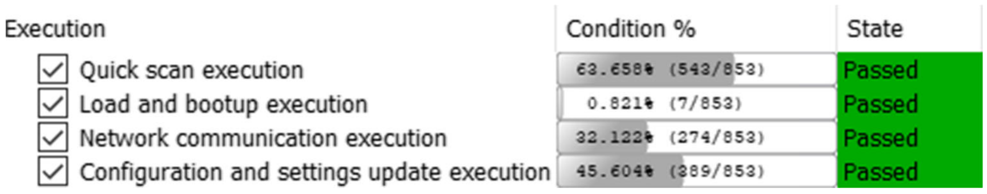
A code coverage tool was applied to measure the percentage of the iBEX source code that involved in action on the prototype scanner. For this, five different execution scenarios were contemplated, and for each scenario, the percent of involved function to the total number of functions was measured (Fig. 11).

The load and boot process required the minimum amount of functions to complete. This process was finished in almost 300 ms. Network communication (i.e., PACS/RIS communication workflow) required almost 20 s, and configuring the device settings took 40 s. The maximum amount of iBEX which is approximately equal to 80% of iBEX got involved to accomplish a quick scan. From another perspective, Fig. 12 summarizes the condition pathways of each execution scenario.

Discussion

iBEX is an open-source and platform-independent software for digital X-ray scanners. In contrast to [36], iBEX is developed to be platform-independent realization of an X-ray workstation software package. Its development process was adopted to be compatible with IEC 62304, ISO/IEC/IEEE 29119-5, and ISO 14971 medical device standards. Project documents, samples, and help files were prepared in RUP templates. These documents, when accompanied with test results of the whole X-ray scanner, could provide evidence of compliance with certifications like FDA or CE. Nonetheless, we must emphasize that iBEX is not evaluated in (or applied for) pre/clinical and human examinations. Our current iBEX implementation is limited to phantom imaging and development of advanced image processing techniques required by our line scanner setup. Manufacturers and researchers can continue customizing and developing both iBEX and their device in compliance with pertinent regulations to finish up with a medical grade certified X-ray device.

Fig. 12 Percent of if-else statements passed to accomplish each execution scenario depends on its complexity



Execution	Condition %	State
✓ Quick scan execution	63.658% (543/853)	Passed
✓ Load and bootup execution	0.821% (7/853)	Passed
✓ Network communication execution	32.122% (274/853)	Passed
✓ Configuration and settings update execution	45.604% (389/853)	Passed

Open-source community should take notice the importance of the full iBEX package, including the complete set of development notes, developer guide, manuals, risks analysis, source code test results, and examples. Similarly, any contribution must come with necessary additional user and developer documentations. In the future, either changes may happen to iBEX core itself or new plug-ins will be developed. Whenever iBEX core is updated, source code tests and risks analysis should be re-evaluated and reflected on the relevant project documents. In this way, iBEX core consistency is preserved, and the updated documents can be used to provide evidence of compliance with medical standards. In the same manner, suspicious plug-ins will be detected by inconsistent risk analysis or incomplete source code test results. Any discrepancy between documented claims and compiled code could be a sign of a vulnerable contribution, and the other open-source community members could report it to fix the issue or completely remove it from the project repository.

The idea of plug-in extension is not novel to the medical imaging. For example, in magnetic resonance imaging (MRI) modality, manufacturers with an agreement provide an extension mechanism for researchers to integrate their custom sequences and/or reconstruction algorithms. Usually, in those devices, the researchers' contributions are executed in a controlled manner, and they are utilized only after the institutional approval (which may include the ethical review by a formal body for animal and/or human research) has been acquired. Likewise, iBEX core utilized the Dependency Inversion Principle (DIP) [20] in order to isolate the plug-ins from the functionalities that are critical for the safety of the device. Communication in between plug-ins and core is formalized with IDevice and IAlgorithm interfaces, and the core has the data flow control in hand. Additionally, each IDevice plug-in runs within their dedicated isolation environment, and they cannot interfere with each other's operation. This separation protects the whole system against outlier device plug-ins and eases the debugging process. The IAlgorithm plug-ins are also sandboxed; furthermore, they receive a separate execution thread. Thus, the core is protected against system blocking image processing algorithms. Despite all these arrangements, there is still a chance for abnormal incidents. It is the responsibility of the developer to accomplish comprehensive testing, which includes plug-in tests, plug-in and core junction tests, and whole scanner device tests. Finally, for protecting the system against unauthorized plug-ins, a plug-in certification and signing mechanism could be added to plug-in manager. This protection mechanism would be complex, and from our point of view, it is in contrast with *open-to-extension* spirit of iBEX; however, for commercial devices, it seems to be unavoidable.

Among the list of exerted packages, VTK, SQLite, CTK, and Log4Qt are open source and freely available on their corresponding owner's websites; but Qt and DCMTK are exception in this list. Qt is distributed under both open-source freeware for noncommercial projects and commercial licenses. iBEX developers, specifically commercial users, should adhere to legal obligations enforced by the licensing agreements. Currently, DCMTK free evaluation period is for 4 months, and any further use requires a full license agreement. Also, some device component manufacturers may ask for a nondisclosure agreement (NDA) to protect their Application Programming Interfaces (API's). In such situations, developers should avoid to publicly share their plug-ins. For example, in our full-body scanner case, Gulmay power generator API was under protection, and its plug-in's implementation file (i.e., .cpp file) was removed before uploading into the repository.

iBEX plug-in mechanism breaks the limitation of *as-is* X-ray machine and turns it into a plug-integrate-play (PiP) device. In other words, researchers, based on their budget and workload, can select their own scanner's configuration and optimize its design. The details of how to develop a new plug-in and its source code is available on GitHub [12] repository readme section. The repository contains both device and algorithm plug-ins which could be used as template for developing plug-ins.

iBEX is an ongoing project and open to improvements. As seen in Fig. 11, execution time of configuration and settings update is not negligible and could be improved. There are some limitations in the current release. For example, there is no platform-independent programming library available for writing a CD or DVD. So, this tool is not yet implemented in iBEX. Medical printers' specifications differ from conventional printers, and to the best of our knowledge, there is no platform-independent medical printer emulator or library. However, in the future, these tools are feasible to be developed by the community.

Conclusion

iBEX, as a free and open-source project, aimed to encapsulate all the required functionalities of a medical grade digital X-ray workstation. By opening the doors of the X-ray imaging device software to open-source community and developing high-quality and well-documented software which are compatible with medical software standards, it will be possible to have more stable, robust, and flexible imaging scanners with lower costs in a short period of time. Eventually, this would bring higher-quality workstations at reduced costs for health centers and democratize X-ray device development.

Appendix

A summary of the KDT results is given in Table 5

Table 5 iBEX test table. Tests are organized in two-tier KDT framework proposed in [15]

Domain layer	Test interface layer	Type	Note
RegisterPatient	`Test_ UpdatePatientDemographicInfo();`	AUTO	PASS
	`Test_UpdatePatientId();`	AUTO	PASS
	`Test_UpdateReferringPhysician();`	AUTO	PASS
	`Test_UpdateAdmissionNumber();`	AUTO	PASS
	`Test_UpdateAccessionNumber();`	AUTO	PASS
	`Test_GetPatientDemographicInfo();`	AUTO	PASS
	`Test_GetPatientId();`	AUTO	PASS
	`Test_GetReferringPhysician();`	AUTO	PASS
	`Test_AppendRegion();`	MAN	PASS
	`Test_RemoveRegion();`	MAN	PASS
	`Test_ClearForm();`	AUTO	PASS
	`Test_GetAnatomicRegionList();`	AUTO	PASS
ViewImage	`Test_DisplayImage();`	MAN	WARN
	`Test_ClearImageViewer();`	MAN	PASS
	`Test_OnThumbnailChanged();`	MAN	PASS
	`Test_OnVerticalFlipToggled();`	MAN	PASS
	`Test_OnHorizontalFlipToggled();`	MAN	PASS
SelectTask	`Test_OnNewTaskRowIsSelected();`	MAN	PASS
QuickScan	`Test_OnActivateExamination();`	MAN	PASS
ConfigNetworkConnection	`Test_GetPACSServerName();`	AUTO	PASS
	`Test_GetStorageServerName();`	AUTO	PASS
	`Test_GetPACSServerport();`	AUTO	PASS
	`Test_GetWorklistServerAETitle();`	AUTO	PASS
	`Test_GetWorklistServerIP();`	AUTO	PASS
	`Test_GetWorklistServerPort();`	AUTO	PASS
	`Test_SetWorklistServerAETitle();`	AUTO	PASS
	`Test_SetWorklistServerIP();`	AUTO	PASS
	`Test_SetWorklistServerPort();`	AUTO	PASS
ManageDatabase	`Test_WriteToTable();`	AUTO	PASS
	`Test_ReadFromTable();`	AUTO	PASS
	`Test_ChangeFieldInTable();`	AUTO	PASS
	`Test_SearchInTable();`	AUTO	PASS
	`Test_RecordConsistency();`	AUTO	PASS
	`Test_AutoSqlStringMaker();`	AUTO	PASS
LogEvent	`Test_SystemDebug();`	AUTO	PASS
	`Test_SystemInfo();`	AUTO	PASS
	`Test_SystemWarn();`	AUTO	PASS
	`Test_SystemError();`	AUTO	PASS
	`Test_SystemFatal();`	AUTO	PASS
CommunicateWithPACS	`Test_ SendingSingleDicomFileToPacs();`	MAN	PASS
	`Test_SendingDirectoryContents();`	MAN	PASS
	`Test_ConnectingPacs();`	MAN	PASS
ManageDeviceMetaData	`Test_UpdateManufacturer();`	AUTO	PASS
	`Test_UpdateInstitutionName();`	AUTO	PASS

Table 5 (continued)

Domain layer	Test interface layer	Type	Note
	`Test_UpdateStationName();`	AUTO	PASS
	`Test_ UpdateManufacturersModelName();`	AUTO	PASS
CreateDICOMFile	`Test_CreateDICOMFile();`	MAN	WARN
UpdateWorklistFromRIS	`Test_AppendRecord();`	MAN	PASS
	`Test_ModalityWorkListConnection();`	MAN	PASS
DevicePlug-in	`Test_LoadDevicePlugins();`	MAN	PASS
	`Test_UpdateDemoDeviceSettings();`	MAN	PASS
	`Test_UpdateConnectionSettings()`	AUTO	PASS
	`Test_GetConnectionSettings()`	AUTO	PASS
	`Test_ExecuteDemoDevicePlug-in();`	MAN	PASS
	`Test_DeviceCommunicationError();`	MAN	WARN
AlgorithmPlug-in	`Test_ImageViewerLoadPlugins();`	MAN	PASS
	`Test_UpdateAlgorithmParameter();`	MAN	PASS
	`Test_ExecuteAlgorithmPlug-in();`	MAN	PASS

References

1. Sprawls P: Physical principles of medical imaging, second edition. Michigan: Aspen Publishers, 1993
2. Wolbarst AB: Physics of radiology. Appleton & Lange: Michigan, 1993
3. Seeram E: X-ray imaging equipment : An introduction. Charles C Thomas Pub Ltd: IL, 1985
4. Chen Q, Wu J, Ou X, Huang B, Almutlaq J, Zhumekenov AA, Guan X, Han S, Liang L, Yi Z, Li J, Xie X, Wang Y, Li Y, Fan D, Daniel BLT, Angelo HA, Mohammed OF, Bakr OM, Wu T, Bettinelli M, Yang H: All-inorganic perovskite nanocrystal scintillators. Nature 561(7721):88–93, 2018
5. Körner M, Weber CH, Wirth S, Pfeifer K-J, Reiser MF, Treitl M: Advances in digital radiography: Physical principles and system overview. RadioGraphics 27(3):675–686, May 2007
6. Lee E, Meyer J, Sandison G: Collimator design for spatially-fractionated proton beams for radiobiology research. Phys. Med. Biol. 61(14):5378–5389, Jul. 2016
7. Radiographic Protocol Guide. wikiRadiography. [Online], Available at http://www.wikiradiography.net/page/Radiographic+Protocol+Guide. Accessed 09 July 2019.
8. Alava J. P., Antonio J.: Computer Vision and Medical Image Processing: A brief survey of application areas. JAIIO 44, ISSN: 2451–7585, 2015.
9. Seibert J. A., Fundamentals of CR, DR and PACS. IAEA, reference number: 44073100, 2012.
10. Samei E, Flynn MJ: An experimental comparison of detector performance for direct and indirect digital radiography systems. Med. Phys. 30(4):608–622, Mar. 2003
11. American College of Radiology, available at https://www.acr.org/-/media/ACR/Files/Practice-Parameters/elec-practice-medimag.pdf. Accessed 12 March 2019.
12. Part 11, Electronic Records; Electronic Signatures - Scope and Application. FDA, New Hampshire,2003
13. council directive 93/42/EEC concerning medical devices. COUNCIL, Geneva, 1993.
14. IEC 62304:2006 - Medical device software – Software life cycle processes. ISO, Geneva, 2006.
15. ISO/IEC/IEEE 29119–5:2016 Software and systems engineering — Software testing — Part 5: Keyword-Driven Testing. ISO, Geneva, September 2013.
16. Test Language - Introduction to Keyword-Driven Testing, [Online], Available: http://www.methodsandtools.com/archive/archive.php?id=108. Accessed 09 July 2019.
17. ISO 14971:2007 Medical devices — Application of risk management to medical devices. ISO, Geneva, 2007.
18. Norweck J., Seibert A., Andriole K., Clunie D., Curran B., Flynn M., Krupinski E., Lieto R., Peck D. J., Mian T. A., Wyatt M.: ACR–AAPM–SIIM Technical Standard for Electronic Practice of Medical Imaging. J Digit Imaging, DOI: https://doi.org/10.1007/s10278-012-9522-2, September 20, 2012.
19. iBEX project repository, [Online] Available: https://github.com/altaybrusan/ibex/tree/FastDev.Accessed: 12 March 2019.
20. Hall GM: Adaptive code: Agile coding with design patterns and SOLID principles, first edition. Redmond Washington: Microsoft press, 2017
21. Gamma E, Helm R, Johnson RE, Vlissides J: Design patterns : Elements of reusable object-oriented software, first edition. Indianapolis: Addison-Wesley, 2009
22. Kroll P, Kruchten P: The rational unified process made easy: A Practitioner's guide to the RUP, 4th edition. Pearson Education: Boston, MA, 2004
23. NEMA organization. DICOM. [Online] **Available at** http://dicom.nema.org/dicom/2013/output/chtml/part03/sect_A.26.html. .
24. Kitware. VTK. [Online] Available at https://vtk.org/download/. Accessed 2 November 2019.
25. OFFIS computer science institute. DCMTK. [Online] **Available at** https://dicom.offis.de/index.php.en.
26. Common Toolkit. CTK. [Online] **Available at** .
27. Qt. Qt Framework. [Online] **Available at** https://www.qt.io/download. .
28. Log4Qt. Log4Qt package. [Online] Available at http://log4qt.sourceforge.net/. Accessed 2 November 2019.
29. SQLite. SQLite library. [Online] Available at https://www.sqlite.org/download.html. Accessed 2 November 2019.
30. Laplante P.: What every engineer should know about software engineering. CRC Press. ISBN 0849372283, 2007.

31. Pianykh OS: Digital imaging and communications in medicine (DICOM) : a practical introduction and survival guide, second edition. Berlin Heidelberg: Springer-Verlag, 2012
32. Conquest DICOM software. [Online] Available: https://ingenium.home.xs4all.nl/dicom.html. Accessed: 12 March 2019.
33. DVTk· RIS Emulator. [Online] Available: https://www.dvtk.org/dicom/ris-emulator/. Accessed: 12 March 2019.
34. Gao J, Tsao H-SJ, Wu Y: Testing and quality Assurance for Component-based Software, first edition. Artech House: Norwood, MA, 2003
35. Atlassian. An introduction to code coverage. [Online] Available at https://www.atlassian.com/continuous-delivery/software-testing/code-coverage. Accessed 2 November 2019.
36. Cosic D: An open medical imaging workstation architecture for platform-independent 3-D medical image processing and visualization. IEEE Trans. Inf. Technol. Biomed. 1(4):279–283, 1997