Pers Ubiquit Comput (2015) 19:623–633
DOI 10.1007/s00779-015-0849-9

ORIGINAL ARTICLE

A development framework for mobile healthcare monitoring apps

Pilar Mata[1] · Austin Chamney[1] · Gary Viner[2] · Douglas Archibald[2,3] · Liam Peyton[1]

Received: 17 November 2014 / Accepted: 26 May 2015 / Published online: 25 June 2015

Abstract Developing healthcare monitoring apps is non-trivial as it requires a balance between simple, easy-to-use interfaces, and powerful business intelligence reporting capabilities, both of which must be integrated into the day-to-day tasks and procedures of clinical practice. This paper presents a development framework for building and deploying mobile healthcare monitoring apps. The framework combines an application development methodology (that ensures adoption and effectiveness of apps when deployed) with an application architecture and component library (to simplify and reduce the development effort needed to implement, deploy and maintain such apps). The development framework is evaluated using a case study of a mobile healthcare monitoring app developed and deployed in collaboration with a team of healthcare researchers and doctors to support the training of residents in family medicine.

Keywords Application architecture · Business intelligence · Development methodology · User-centered design · Healthcare · Practice profiles · Performance measurement

✉ Liam Peyton
lpeyton@uottawa.ca

[1] Faculty of Engineering, University of Ottawa, 800 King Edward Avenue, Ottawa K1N 6N5, Canada

[2] Faculty of Medicine/Department of Family Medicine, University of Ottawa, 43 Bruyère Street, Ottawa K1N 5C8, Canada

[3] Bruyère Research Institute, 85 Primrose Avenue, Ottawa K1R 7G5, Canada

1 Introduction

Increased capabilities of mobile devices have led to greater use in healthcare. Physicians have adapted to these mobile technologies and incorporate them in their daily lives [1]. They are also used during clinical practice training for students and residents [2]. Mobile apps can be used to collect data and send reports to clinicians for applications like real-time monitoring of emergency room wait times [3] or monitoring of clinical training experience [4].

Developing healthcare monitoring apps is non-trivial as it requires a balance between simple, easy-to-use interfaces, and powerful business intelligence (BI) reporting capabilities, both of which must be integrated into the day-to-day tasks and procedures of clinical practice. There are often interoperability issues that affect the adoption and effectiveness of healthcare monitoring apps [5]. These are often due to differences between the design of these apps and what occurs in actual healthcare settings [6] which result from a disconnect between clinicians and the development team. Furthermore, traditional application architectures like .Net and J2EE support a generic approach to implementation that leads to high code complexity [7], and a higher level of effort is required to maintain and customize the app to the specific and evolving needs of clinicians. The databases that result from such frameworks are not optimized to support powerful BI reporting. Often the development of suitable reporting capabilities can be as lengthy and difficult as creating the app itself [8].

This paper presents a development framework for building and deploying mobile healthcare monitoring apps. The framework combines an application development methodology (that ensures adoption and effectiveness of apps when deployed) with an application architecture and component library (to simplify and reduce the development

effort needed to implement, deploy, and maintain such apps). The development framework is evaluated using a case study in which a Resident Practice Profile (RPP) app was developed and deployed in collaboration with a team of healthcare researchers and doctors at the University of Ottawa to support the training of residents in family medicine. RPP is also evaluated in comparison with similar apps built with a different application development methodology and architecture. This paper extends and expands upon previous work published in relationship to the case study [9].

2 Background

There is increasing pressure on healthcare organizations to deploy monitoring apps that measure performance related to quality of care goals [10, 11]. A critical aspect of such apps is to provide seamless easy-to-use interfaces that minimize the data collection burden and maximize the value that reports provide while ensuring user acceptance and adoption [12].

An application architecture is "an integrated set of software artifacts … (that) provide a reusable architecture for a family of related applications" [13] so that new applications can be developed quickly. Traditional web application architectures, such as J2EE [14], provide a three-tier architecture (browser-based user interface, application server, and database) where most development effort is directed toward code that runs on the application server in a manner that makes it difficult to directly address user interface or reporting effectiveness. AJAX [15] is newer technology, which expands the development effort to the browser-based user interface to provide more seamless, and easy-to-use interfaces. However, reporting is still an issue as there is no direct relationship between data collected in these user interfaces and the effectiveness of reporting based on that data.

Databases that are optimized for reporting are usually organized into a star schema [16] in which data are logged into a central fact table, linked to peripheral tables that define dimensions along which the data can be analyzed. Third-party business intelligence tools and dashboards [17] can provide flexible, generalized reporting against databases defined in this way.

While the technical aspects of the application development process are critical, it is also fundamental to ensure user acceptance and adoption of the technology by following an appropriate application development methodology. User-centered design actively involves "users for a clear understanding of user and task requirements, iterative design and evaluation, and a multi-disciplinary approach" [18]. This approach is particularly relevant in the healthcare domain given implementation success/failure rates have been cited as one important issue in the literature [19, 20]. The main reason for this is the disconnect between clinicians and the development team. Barriers to adoption, in particular issues related to usability, can be detected and addressed through the use of think-aloud sessions [21] in which users are encouraged to verbalize their thought processes, which are recorded, while they perform tasks using the application under development.

Recent research suggests there is a need for healthcare monitoring apps to measure performance of resident physicians. Recent work highlights the gap between the quantity and type of clinical problems residents see during their training and those required by program guidelines [22]. Practice profile apps [23, 24] have appeared that allow medical practitioners to log their experiences electronically to provide feedback. Just in Time Medicine [4] was developed by researchers at the University of Michigan to enable physicians to provide assessments of medical students on their clinical experience, while LogMD [25] was developed in conjunction with the Association of Canadian University Departments of Anesthesia to enable physicians to track their experience against international benchmarks.

3 Development framework for mobile healthcare monitoring apps

The development framework that we propose is based on an application development methodology that engages stakeholders and users throughout in a three-phased, iterative process of application modeling, implementation, and evaluation to ensure user acceptance and adoption. Within our development framework, this methodology is supported by an application architecture that simplifies the complexity and effort of the implementation phase. Our proposed application architecture characterizes mobile healthcare monitoring apps in terms of a simple architecture optimized for BI data collection and reporting, so that such apps can be assembled and configured using a library of reusable templates and components.

3.1 Application development methodology

The main idea of the application development methodology that we are proposing is to engage stakeholders and users throughout in a three phase iterative process of modeling, implementation, and evaluation to ensure user acceptance and adoption. The methodology focuses on lightweight development and evolution of mobile healthcare monitoring apps that are easily deployed which capture the minimum amount of data in easy-to-use mobile

app forms needed to compute the metrics required to provide insight into healthcare performance with mobile accessible reports.

Figure 1 shows the main tasks in the three phases of our methodology. The methodology starts by an initial evaluation of current clinical practice. The feedback from this evaluation is used to model how clinical practice should be monitored. This model is in turn used to implement a mobile healthcare monitoring app that collects data to monitor the ongoing status of clinical practice through reports. After each implementation phase, there is a thorough evaluation phase in which stakeholders and users are systematically engaged to use and evaluate the mobile app while it is still under development and issues can be addressed. This cycle of evaluate, model, implement, and evaluate again is repeated with increasingly systematic and precise evaluations refining both the monitoring model, and the implementation of the mobile healthcare monitoring app. The focus is to ensure adoption and usability of the monitoring app to improve clinical performance.

The model phase consists of understanding what the desired goals for healthcare performance are; defining what metrics can be used to quantify and communicate how well those goals are being met; and identifying what data can be collected by whom in order to measure those metrics and generate appropriate reports for visualizing the results. There may be existing databases or healthcare systems that can provide some or all of the data. In practice, though, it can be difficult and time consuming to extract and transform the necessary data in order to measure and report on performance. Often the analysis and understanding of goals will reveal that the necessary data are either missing or not adequately structured in the existing databases. A mobile healthcare monitoring app becomes necessary to provide more effective monitoring. As such, it is also important to identify and define adoption criteria to ensure adoption and usability of the healthcare monitoring solution.

The implementation phase consists of configuring the reporting database to store and compute the metrics and data defined and identified in the model; configuring the forms and reports for the user interface of the mobile app; and systematically mapping the clinical domain concepts needed for healthcare monitoring (demographics, diagnoses, etc.) to the labels and values seen in the forms and reports of the mobile app.

The evaluation phase consists of think-aloud sessions in which the correctness and usability of the healthcare monitoring app are evaluated by observing users and stakeholders performing their daily tasks; and by defining engineering and domain checklists that can be used to systematically quantify the extent to which the app is meeting the adoption criteria and enabling stakeholders and users to monitor performance. An analysis of the think-aloud sessions and checklists is used to understand how to improve the model and implementation in the next iteration of development.

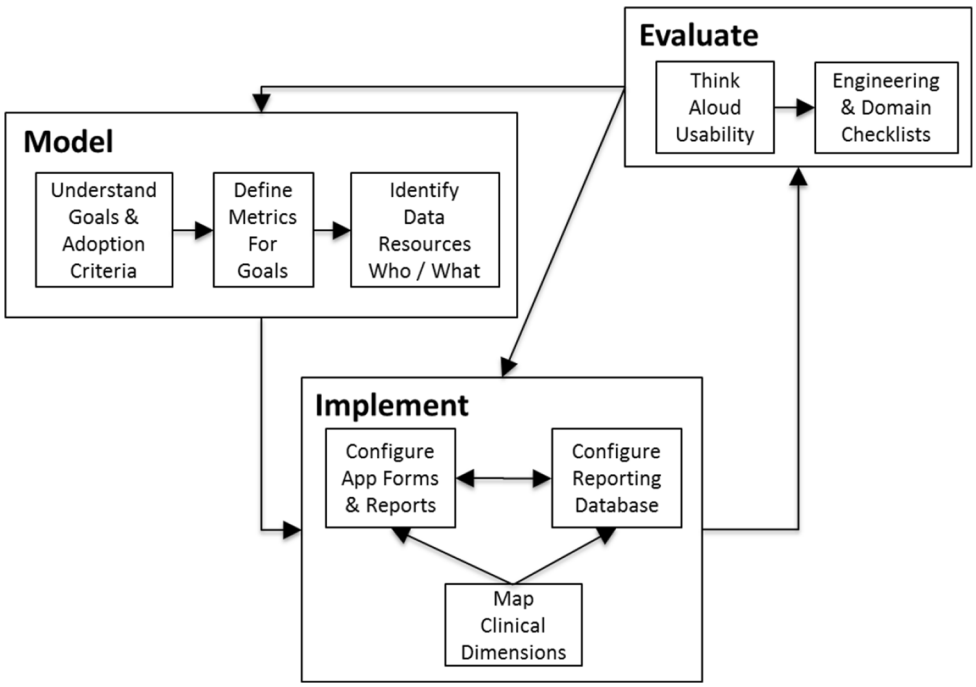


Fig. 1 Application development methodology

3.2 Application architecture

The main idea of the application architecture we are proposing is to simplify and reduce the effort needed in the implementation phase (see Fig. 1) so that the software development team can respond quickly to the feedback from users and stakeholders, and simplify maintenance of the resulting mobile healthcare monitoring app. With the right application architecture, an application should largely be a task of configuration and linking of pre-defined components. In Fig. 2, we define a three-tier architecture for mobile healthcare monitoring apps that separates the user interface (forms and reports) from the reporting database (star schema with fact table, dimension tables, and queries) in such a way that each can be configured and optimized separately but automatically linked through an application-independent middleware.

The user interface tier contains HTML, CSS, and JavaScript that configure and sculpt the particular look and feel of the mobile healthcare monitoring app in terms of forms for clinical data collection and reports for clinical monitoring. Naming conventions associate a HTML form and its fields to a fact table and its related dimension lookup tables (one for each form field) in the Reporting Database tier. There is a minimal amount of JavaScript required to interact with the JavaScript Client (AJAX) that communicates with the reporting database through the Data Access Object Service (REST). There is an extendable library of customizable HTML and CSS templates that can be used to define the look and layout of the app so that all form factors and device types can be addressed.

The application-independent middleware tier consists of a generic JavaScript Client that runs in the browser to support the user interface forms and reports and a Data Access Object Service that runs on a server to interact with the Reporting Database in response to requests from the JavaScript Client. The JavaScript Client provides a generic library of user interface controls and widgets linked to the Reporting Database that can be used to assemble powerful and easy-to-use forms and reports. The JavaScript Client also provides a set of API calls for making requests to the Database Access Object Service. For forms, the API saves form data to a fact table in the Reporting Database, and pulls data from dimension lookup tables to populate the dropdowns for form fields. For reports, the API executes application-specific queries. The JavaScript Client also manages authentication and authorization.

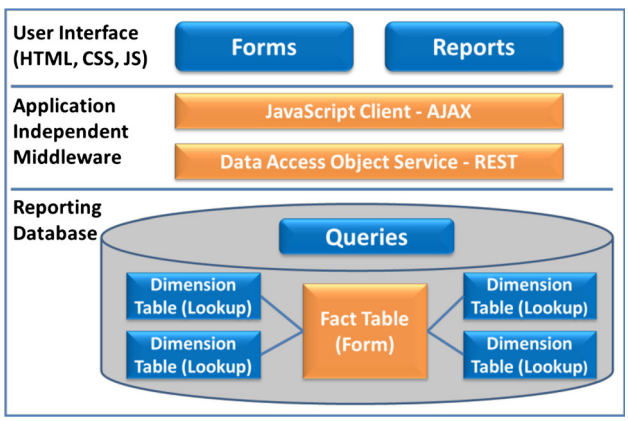


Fig. 2 Application architecture

The Data Access Object Service handles the form and report-related API calls with a series of parameterized SQL queries that provide access to the Reporting Database. The parameters consist of fact table and dimension table names as well as values used by queries to filter and select. The Data Access Object Service, like the JavaScript Client is completely generic and pre-packaged and requires no custom coding and only a minimum amount of configuration to specify the database connection information for the mobile healthcare monitoring app.

Finally, the Reporting Database is defined by a standardized generic multi-dimensional star schema optimized for reporting and compatible with standard third-party reporting tools. There is a strict naming convention for columns, tables, lookups, and facts so that the user interface can use name-based identifiers when it interacts with the JavaScript Client to make requests to the Reporting Database through the Data Access Object Service. Standardized generic templates are provided for the fact table and many types of dimension tables (data, demographics, diagnoses) as well as multi-select tables and authentication and authorization tables.

4 Resident Practice Profile case study

Our research methodology is based on design science research in which a novel or new solution to an information systems problem is developed iteratively through a process of problem identification, design, development, demonstration, and evaluation [26]. Our proposed development framework was refined and evaluated using a case study in which a mobile healthcare monitoring app was developed and deployed in collaboration with a team of healthcare researchers and doctors at the University of Ottawa to support the training of residents in family medicine. We went through four complete iterations of our application development methodology during the case study and made a significant refinement and improvement of our proposed application architecture during the implementation phase of the final iteration.

The app we developed, called the Resident Practice Profile (RPP), enables residents in the family medicine

program to self-assess how well their clinical experience, under the supervision of a physician supervisor, provides a suitable breadth of experience across the types of patients, diagnoses, and procedures that are covered in the post-graduate family medicine curriculum. We did consider generating reports off existing electronic health records (EHR) systems, but it was not practical for several reasons. First, there was no EHR common to all the clinics where residents practiced. But, even if the case study was restricted to a single clinic, EHR data were inconsistent with the family medicine curriculum; there was no mechanism for doctors to configure the EHR to collect what was actually needed, and there was no built-in support for third-party reporting tools.

We also evaluated existing practice profile apps, to see if any could be used. Just In Time Medicine and LogMD were evaluated, but they had a different purpose (assessment of medical students, self-assessment of physicians) and were not customizable to what was needed for family medicine residents at the University of Ottawa. A comparison of them and the RPP app that was built for the case study is given in Sect. 5.

The initial version of the app RPP_1 was built as a general purpose web application using a traditional J2EE application architecture and a generic software development methodology. We then went through four iterations of our application development methodology (evaluate, model, implement, and evaluate again). In the first three iterations, RPP_2 was implemented with our initial proposed application architecture (QuickForms 1.0). In the last iteration, we used a much improved version of our application architecture (QuickForms 3.0) to implement RPP_3.

The results of our proposed application development methodology are described in Sect. 4.1, while the improvements over traditional application architectures obtained from the two versions of our proposed application architecture are highlighted in Sect. 4.2. The key features of the RPP app that resulted from using our development framework are described in Sects. 4.3, 4.4, and 4.5.

4.1 Application development methodology

The initial RPP_1 app built using traditional software methodology and a generic application architecture was not well received. As we started the first iteration of our application development methodology and performed our initial evaluation, it became clear that there was a great deal of misunderstanding between the stakeholders and the application developers, and that the stakeholders were not fully aligned. There was enough feedback from a simple attempt to walk through a demonstration of the application to stakeholders that no think-aloud sessions with actual residents were attempted. The feedback from RPP_1 was used to more systematically model the healthcare monitoring app that was needed in terms of the goals, metrics and data to be collected and reported on. That analysis prompted us to design a much more specific and suitable application architecture (QuickForms 1.0). Perhaps most significantly, as we iterated, the feedback from each evaluation phase (especially think-aloud sessions) enabled us to identify the key adoption criteria for RPP:

- Residents should log their clinical experience (patient visits) in less than 30 s on a mobile device.
- Residents should see reports that map clinical experience in relation to the family medicine curriculum.
- Doctors responsible for family medicine should be able to configure drop downs for RPP.
- Third-party reporting tools must be supported for in-depth analysis to improve family medicine program.

While developing RPP_2, there were regular group evaluation sessions of the app and its design, and three think-aloud sessions conducted with the participation of four residents, in addition to the research team. These evaluation sessions occurred between September 2012 and June 2013 before starting an official 1 year pilot of RPP with the four residents in July 2013. RPP_3, which had minor bug fixes, but which was re-implemented using our improved application architecture (QuickForms 3.0), was released in September 2013.

A total of 1607 diagnoses for 1205 different visits were logged during the pilot. Each think-aloud session uncovered significant barriers to the usefulness, usability and adoption of the RPP app that had to be carefully analyzed in order to arrive at the design breakthroughs needed to address the barriers.

First, the amount of information that needed to be collected for a single visit was cumbersome in combination with the sophisticated user interface needed in the app to select information quickly. Residents were getting lost in the app and needed to scroll up and down to remember what they had entered. The resolution to this resulted in standardized application templates and summary controls that are described in Sect. 4.3

Second, careful analysis of the family medicine curriculum was needed in order to clearly identify the dimensions needed for reporting (tracking, demographics, diagnoses, and self-assessment) and to determine the values for each dimension that would populate the drop-down controls for the various form fields. These dimensional values also directly mapped to reports. This is described in Sect. 4.4.

Finally, the sheer number of possible diagnoses ($\sim$500) relevant to a family medicine patient visit was overwhelming and had to be organized in an easy-to-use control that reflected the organization of the family medicine

curriculum and allowed for multi-selection. The generic, reusable hierarchical control that provides flexible mapping to drill-through reporting is described in Sect. 4.5.

For RPP_3, the major effort was to keep the functionality provided in RPP_2, but completely refactor the organization of the user interface and JavaScript Client tiers to have an order of magnitude simplification in the amount and complexity of code needed to configure the RPP app. This was described in Sect. 4.1. In addition, a number of routine fixes and a few new reports have been incorporated into RPP_3 based on the feedback from the RPP_2 pilot. RPP_3 will be further evaluated as it is rolled out in December 2014 for use by 75 residents in family medicine.

4.2 QuickForms application architecture

The RPP_1 app built in J2EE had 3154 lines of code and it became quite complex to manage all the custom code in the different tiers, although it was not even feature complete. Worse, it was not easy to optimize the user interface for different types of mobile devices and form factors, and it was difficult to create anything other than simple reports. The RPP_2 app, built in QuickForms 1.0, that was feature complete, had 4551 lines of code, almost all of which were written in JavaScript. QuickForms 1.0 provided the basic architecture shown at the bottom in Fig. 3, but the JavaScript Client was poorly organized with no attempt to encapsulate re-usable pre-defined components. There was a lot of custom code in the user interface that was very complex. All the desired functionality for RPP was delivered, but it was just as complex for a developer to build as the initial RPP_1 app. The final RPP_3 app built in QuickForms 3.0 has only 1273 lines of code and includes a half dozen extra reports that were not part of RPP_2. The functionality for the user is the same as RPP_2, but the JavaScript Client and user interface were completely refactored. It provides a systematic approach to encapsulating reusable controls so that developing any healthcare monitoring application is now largely a matter of configuration.

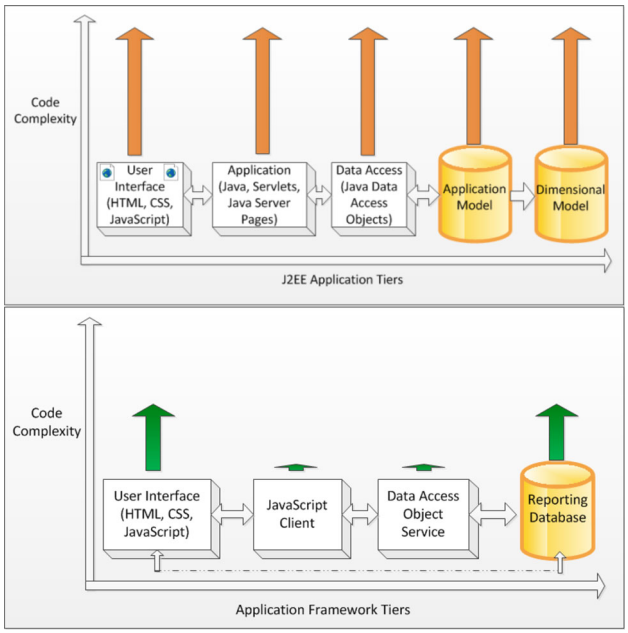


Fig. 3 Comparison of J2EE and QuickForms application architectures

Figure 3 compares the tiers of the initial J2EE application architecture with the tiers of the final QuickForms 3.0 application architecture based on code complexity, as this factor can significantly increase the level of effort required to create and maintain the application in response to the specific and evolving needs of users and stakeholders. In the J2EE tiers (on top), complex custom code is written in all tiers to define the application, its user interface, and its data access. Typically, an Application Model Database is generated from the custom-coded objects in the application and data access. It can be complex to define reports against this database since it is optimized for transactional data entry and not reporting. Often there is an extract, transform and load process [16] to move the data to a database structured around a dimensional model which is optimized for reporting and compatible with third-party reporting tools. There is no direct mapping between data elements in the user interface, application, data access, application model, or dimensional model. Maintaining consistency between them is a painstakingly manual process as any change is potentially repeated in all five places and may require custom coding by five different developers. There are no pre-defined components to optimize the user interface or Reports for a mobile healthcare monitoring app.

In the QuickForms tiers (on the bottom in Fig. 3), there is no Application Model, and the JavaScript Client and Data Access Object Service tiers are completely generic and pre-defined. There is no customer code to write for them, as they are designed to automatically manage the relationship between the user interface and Reporting Database. The complexity of these two tiers for the application developer is essentially reduced to zero. A developer simply has to annotate the fields in an HTML form with the names of the dimension tables in the Reporting Database. Controls and API calls provided by the JavaScript Client automatically populate drop-down choices for fields and save collected data to the Reporting Database through communication with the Data Access Object Service. There is also a powerful set of pre-defined, customizable HTML and CSS templates to optimize the user interface (including built-in reports) for a mobile healthcare monitoring app as well as a pre-defined,

customizable fact and dimension tables for the Reporting Database.

4.3 Templates and summaries

Figure 4 shows an overview of the RPP app. On the left is the home page which uses one of the built-in home page templates with header, application tabs, and a standard login/logout. The titles, tabs, etc., are customizable in an application configuration file. There is a sophisticated, built-in grid control for displaying visits that supports filtering, searching, sorting by clicking on columns, etc. The columns used both in the grid and in the associated filters are all easily configured. On the right is the overview summary for a particular visit. The user can see at a glance a summary of what has been logged for tracking, demographics, and assessment without opening the forms for those. The form summaries are a built-in control that can be flexibly configured to determine what fields are summarized. The idea for form summaries came from one of the think-aloud sessions when residents were having difficulty seeing the complete overview of what they had filled in for a patient visit. As well, the tracking summary greatly simplified data entry. Typically, a resident would see many patients in a row at a given clinic. When the tracking information was entered once for a visit, it was automatically repeated and pre-filled for subsequent visits and the resident could see at a glance that no data entry was needed for tracking.

4.4 Simple forms and reports with lookup table management

The critical element of the QuickForms application architecture is, of course, the ability to separate and yet automatically link forms with the Reporting Database. In Fig. 5, we see the Demographics Form on the left and its associated Age/Gender report on the right. The developer links a field in the form to the dimensional table that supplies the drop-down values for that field by simply adding a reference to the table name in the HTML definition of the form. These same values are used in the report associated with the form. The Age/Gender Report uses the same values for Age and Gender that appear in the selection controls for Age and Gender.

In this case, the resident has seen a disproportionate number of females to male patients in the 21–35 year range, but this is normal for family doctors who see maternity patients. There is also a similar Special Populations/Gender report corresponding to the Care of Special Populations drop down in the form, but this is not shown here. In addition, there is a special design button on the form on the left that is available for users with administrator privileges. The design button brings up a dialog where administrators can edit the values in the lookup tables for any of the form fields, or download and upload files of the values for the dimension lookup table of a particular field.

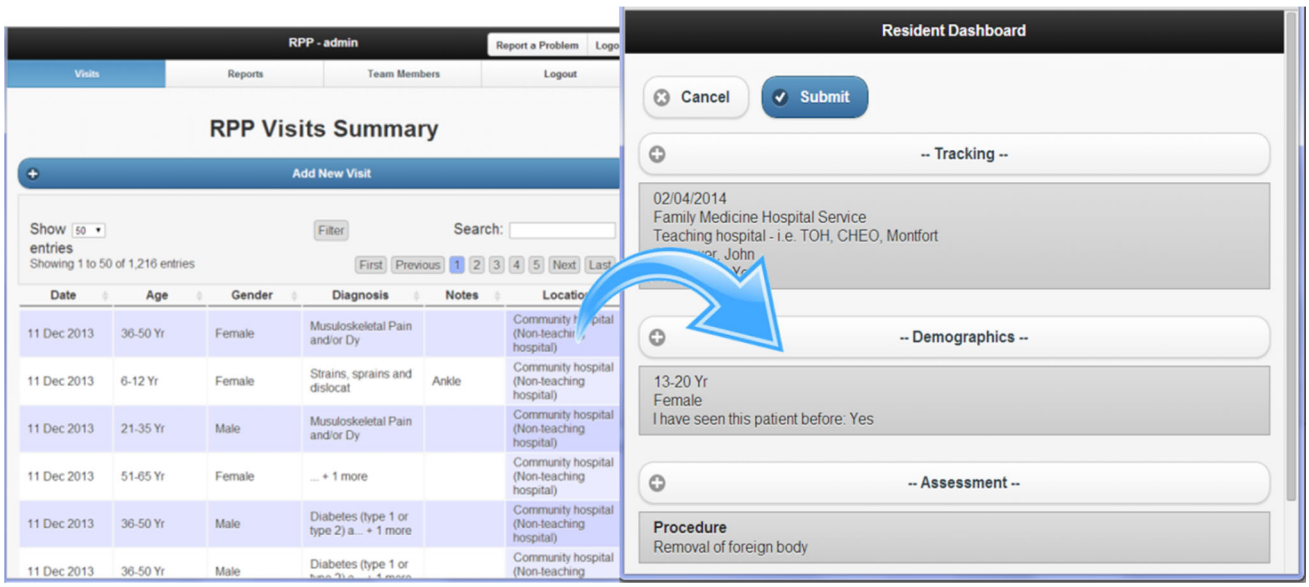


Fig. 4 RPP overview: templated with summaries

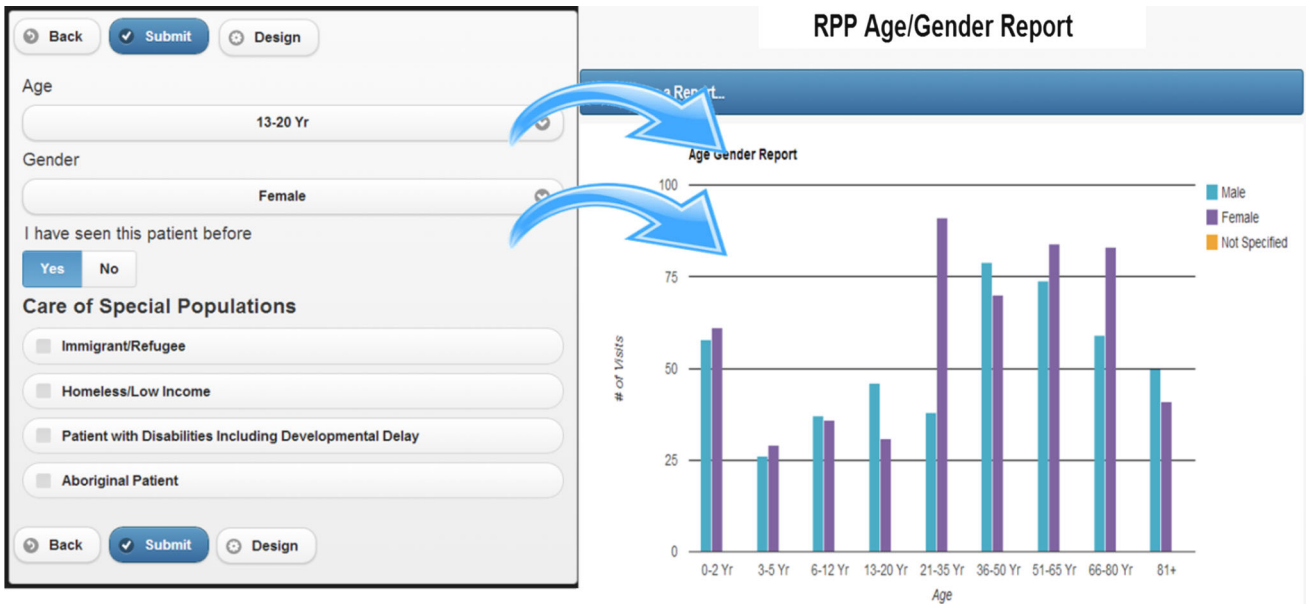
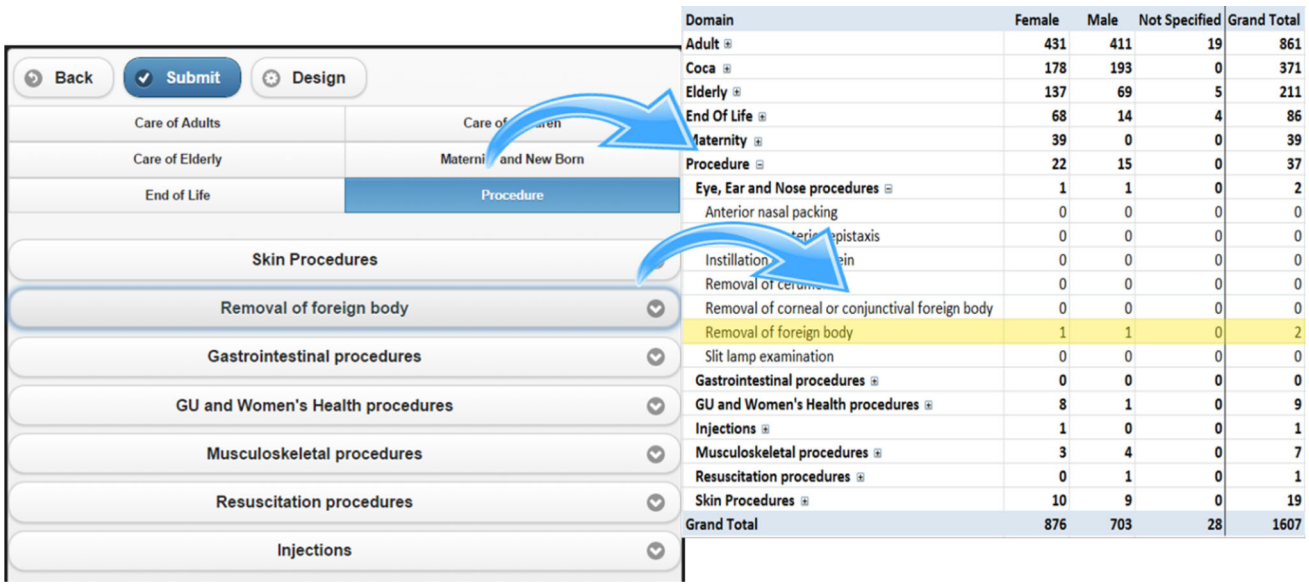


Fig. 5 Demographics form and age/gender report



Domain	Female	Male	Not Specified	Grand Total
Adult	431	411	19	861
Coca	178	193	0	371
Elderly	137	69	5	211
End Of Life	68	14	4	86
Maternity	39	0	0	39
Procedure	22	15	0	37
Eye, Ear and Nose procedures	1	1	0	2
Anterior nasal packing	0	0	0	0
Epistaxis	0	0	0	0
Instillation	0	0	0	0
Removal of	0	0	0	0
Removal of corneal or conjunctival foreign body	0	0	0	0
Removal of foreign body	1	1	0	2
Slit lamp examination	0	0	0	0
Gastrointestinal procedures	0	0	0	0
GU and Women's Health procedures	8	1	0	9
Injections	1	0	0	1
Musculoskeletal procedures	3	4	0	7
Resuscitation procedures	0	1	0	1
Skin Procedures	10	9	0	19
Grand Total	876	703	28	1607

Fig. 6 Multi-level selection control for diagnoses and drillable report

4.5 Multi-level selection control and drillable reporting

The form for assessment is more complex. In Fig. 6 on the left, we see a tabbed multi-level selection control provided by QuickForms that a resident can use to select any and all diagnoses and procedures that apply to a patient. There are roughly 500 diagnoses and procedures covered by the family medicine curriculum and they are organized in the multi-selection control in a similar, though simplified, fashion to their classification in the curriculum. At the top level, there are six tabs corresponding to the major curriculum domains: Care of Adults, Care of Children and Adolescents (CoCA), Care of Elderly, Maternity, End of Life, and Procedures. Within each tab, there is another list of groupings, and within each grouping there are the selections to be made. In this case, the resident is selecting "Removal of Foreign Body" from the list of Eye, Ear, and Nose procedures under the Procedures tab.

The organization of the tabs and groupings and selections is automatic based on the lookup table for the assessment field in the database. All the developer has to

do is specify the column name in the fact table that links to the lookup table. The organization is driven by the classification columns (one for tabs, one for groupings), and the sequence number column to specify the order of values. As well the design button facilitates maintenance of diagnoses and procedures by launching a dialog that allows separate files to be uploaded and downloaded for each tab.

The drillable report on the right in Fig. 6 leverages the same classification columns from the assessment lookup table to organize the report of how many assessments and procedures a resident has seen. One can see at a glance that, in addition to Eye, Ear and Nose procedures, the resident has done very few gastrointestinal, injection, or resuscitation procedures. In addition, most of their diagnoses have been for the Care of Adults.

5 Results and discussion

In this section, we evaluate our proposed development framework. It should be emphasized that these are preliminary results that are only intended to establish the potential for our approach to improve adoption and usability of mobile healthcare monitoring apps, while reducing the effort and complexity required to build and deploy such apps.

In terms of adoption, we found that standard web application development methodology is not sufficient to address adoption barriers in clinical practice. It was not possible to recognize and understand adoption criteria such as recording data in less than 30 s, and ensure value to residents in terms of the reporting using a methodology that left the software development team disconnected from the context of integrating a mobile app into day-to-day activities and thought processes of the residents who would use the app.

Table 1 shows innovations resulting from evaluation at each iteration to address user adoption. The RPP_1 app in J2EE had major disconnects between clinician needs and what was delivered. Making changes to address those flaws was difficult. RPP_2a, 2b, and 2c correspond to the three iterations in which QuickForms 1.0 was used. QuickForms 3.0 was used for RPP_3.

Using our development methodology, major innovations, reflecting a deep understanding of user and stakeholder needs, resulted from the evaluation and model-based analysis that was done in each iteration. In the first iteration, insight into the need for mapping from forms to database tables to reports based on a goal model, resulted in a much improved application template and appropriate application architecture. In the second iteration, insight into the need to eliminate scrolling on the mobile device and target less than 30 s for data entry resulted in sub dialogs, multi-level selection, and summary controls in the app.

By the third and fourth iterations, the development team was in synch with clinical users and stakeholders so the focus was on minor refinements like customizable filters and drillable reports using third-party reporting tools. We also refined our application architecture (QuickForms 3.0) to reduce even more the time and effort required to respond to user needs. Our development methodology that leveraged think-aloud sessions and expert checklists was critical to understand and recognize barriers to adoption.

In terms of usability, Table 2 compares RPP with similar practice profiling applications: Just In Time Medicine (JIT) and LogMD which were reviewed by the entire team of software engineering and healthcare researchers, and doctors.

All three apps are focused on reporting and ease of use for data entry. JIT has a rigid and more detailed data entry to ensure the same steps are followed for all students to ensure consistency in how students were assessed for a single clinical experience. RPP was designed for faster data entry, because it was important for a resident to log all patient encounters so they could self-assess their entire clinical experience. RPP has the most sophisticated support for capturing diagnoses, because the full spectrum was needed to document the clinical experience. RPP is arguably easier to use for integrated data collection and reporting because of its support for form/report linkage, templated summaries, and drill-through reporting.

It is also important to note that our application architecture support for third-party reporting and application configuration addresses three of the four key criteria from our case study (described in Sect. 4.1) better than JIT and LogMD.

Table 1 Evaluation—major innovations

	RPP_1	RPP_2a	RPP_2b	RPP_2c	RPP_3
Architecture	J2EE	Quick Forms1.0	Quick Forms1.0	Quick Forms1.0	Quick Forms3.0
Major innovations	Major disconnect and hard to fix	App template and simple forms mapped to tables mapped to reports	Sub dialogs, summary controls, multi-level selection, <30 s, no scroll	Drillable reports, customizable filters	Minor bug fixes, 6 extra reports

Table 2 Evaluation of RPP versus JIT and Log MD

Criteria	JIT	LogMD	RPP
Purpose	Student assessment	Physician self-assessment	Resident self-assessment
Form factors	All form factors	All form factors	All form factors
Data entry	2–5 min rigid	<2 min	<30 s
Diagnosis selection	One level	Two levels (procedures)	Two levels (all)
Form/report linkage	No	Yes	Yes
Templated summaries	No	No	Yes
Drill-through reporting	No	No	Yes
Third-party reporting	No. Hard-coded reports	No. Hard-coded reports	Yes. Dimensional model
Application configuration	None	None	Data tables. UI controls

- Residents should see reports that map clinical experience in relation to the family medicine curriculum.
- Doctors responsible for family medicine should be able configure drop downs for RPP.
- Third-party reporting tools must be supported for in-depth analysis to improve family medicine program.

In fact, it was precisely because these criteria were not met, that RPP was built using the QuickForms application architecture. This was an issue for all family medicine EHR systems that were reviewed as well as for JIT and LogMD. The general multi-level selection control described in Sect. 4.5 is a reusable control that can map the diagnoses in any healthcare domain, not just family medicine.

Finally, the effort and complexity required to implement a mobile healthcare monitoring app are evaluated in Table 3 by comparing the three different application architectures that were used for RPP. We can see that RPP_2 achieved the main application requirements in terms of features, adoption, and usability, while RPP_3 greatly reduced the complexity of the code through better encapsulation and packaging of pre-defined components. This improved the ability to configure the application and greatly reduced development effort.

It is, of course, premature to use a single case study to claim that this approach works for all healthcare monitoring apps. However, the application development methodology is designed to reduce disconnects between developers and users for any healthcare monitoring app and in the application architecture, the star schema and library of user interface (UI) controls is completely generic and is also intended to be reusable for any healthcare monitoring. There is enough demonstrated potential here to justify further study.

6 Conclusions and future work

Our proposed development framework has established the potential to improve adoption and usability of mobile healthcare monitoring apps, while reducing the effort and complexity required to implement such apps. It significantly reduced the disconnects between software developers and users and guided the resolution of adoption criteria in our case study. As well, the application architecture significantly reduced the effort in building RPP, and it significantly improved the configurability and user experience for both data collection and reporting compared to a traditional web application development framework.

More work is needed, of course, before we can make any definitive claims about how well the development framework would work for other teams and other apps. It should be noted that there is a number of simplifying assumption in our work:

- The storage needs for such apps are not compromised by restricting to a star schema.

Table 3 Evaluation of different versions of RPP

Criteria	RPP_1 (J2EE)	RPP_2 (QuickForms 1.0)	RPP_3 (QuickForms 3.0)
Form factors	PC browser optimized	All form factors	All form factors
Form/report linkage	No	Yes	Yes
Third-party reporting	No. Hard-coded reports	Yes. Star schema database	Yes. Star schema database
Application configuration	None	Data tables	Data tables. UI controls
Encapsulation	No. Mixed layers	No. Complex JavaScript	Yes. QuickForms library
Lines of code	3154	4551	1273

- No rich content or content management (Voice, Pictures, Attachments).
- Multi-locale issues are ignored.
- Concurrency issues in which two or more users might edit the same form or edit the same lookup table at the same time are ignored. The nature of our apps so far allows us to assume only one user at a time.
- Communication, service, and/or application integration issues. Apps are strictly standalone.
- Security and fault tolerance issues except for obvious issues are ignored. The simplicity and benign nature of our apps in a research setting have allowed this so far.

We believe our approach should be useful for most form-based monitoring and logging apps where data are collected for analysis and/or performance management. It addresses the development of apps in complex environments (i.e., healthcare) with evolving information needs and where access to existing data sources constitutes a challenge. By reducing the code complexity of the apps and reducing the disconnects between clinicians and the development team, applications are more likely to be adopted as it could be easily customized, deployed, and maintained. The QuickForms 3.0 application architecture used in our approach was published as an open source project [27] in April, 2014.

Acknowledgments This work was supported by an AIME grant from the School of Medicine at the University of Ottawa, the Mitacs Accelerate program, IBM and an NSERC Discovery Grant. We would like to thank the residents who participated in this case study, especially Alexandre Labelle. We would also like to thank Dr. Susan Humphrey-Murto and Dr. Eric Wooltorton for their participation and extremely useful feedback throughout this research.

References

1. Kafeza E, Chiu D, Cheung S, Kafeza M (2004) Alerts in mobile healthcare applications: requirements and pilot study. IEEE Trans Inf Technol Biomed 8(2):173–181
2. Kho A, Henderson L, Dressler D, Kripalani S (2006) Use of handheld computers in medical education. J Gen Intern Med 21(5):531–537
3. Baarah A, Mouttham A, Peyton L (2012) Architecture of an event processing application for monitoring cardiac patient wait times. Int J Inf Technol Web Eng 7(1):1–16
4. Ferenchick G, Solomon D (2013) Using cloud-based mobile technology for assessment of competencies among medical students. PeerJ Inc 16(5–6):407–411
5. Mouttham A, Kuziemsky C, Langayan D, Ling Y, Peyton L, Pereira J (2012) Interoperable support for collaborative, mobile, and accessible health care. J Inform Syst Front 14(1):73–85
6. Novak L, Brooks J, Gadd C, Anders S, Lorenzi N (2012) Mediating the intersections of organizational routines during the introduction of a health IT system. Eur J Inf Syst 21(5):552–569
7. Johnson R (2005) J2EE development frameworks. Computer 38(1):107–110
8. Simitsis A, Vassiliadis P, Sellis T (2005) Optimizing ETL processes in data warehouses. In: Proceedings 21st international conference on data engineering, pp 564–575
9. Chamney A, Mata P, Viner G, Archibald D, Peyton L (2014) Development of a resident practice profile in a business intelligence application framework. In: 4th international conference on current and future trends of information and communication technologies in healthcare, Halifax, Canada. http://www.sciencedirect.com/science/article/pii/S1877050914010059
10. Leggat S, Bartam T, Stanton P (2012) High performance work systems: the gap between policy and practice in health care. J Health Organ Manag 25(3):281–297
11. Waterson P (2014) Health information technology and sociotechnical systems: a progress report on recent developments within the UK National Health Service (NHS). Appl Ergon 45(2):150–161
12. Rowley P, Gough R, Doylend N, Thirkill A Leicester P (2013) From smart homes to smart communities: advanced data acquisition and analysis for improved sustainability and decision making. In: International conference on Information Society, pp 263–268
13. Schmidt DC, Gokhale A, Natarajan B (2004) Leveraging application frameworks. ACM Queue 2(5):66–75
14. Jing-Mei L, Guang-Sheng M, Gang F, Yu-Qing M (2006) Research on Web application of struts framework based on MVC pattern. In: International workshop on advanced web and network technologies, and applications. Springer, Harbin, pp 1029–1032
15. Matthijssen N, Zaidman A, Storey M, Bull I, Van Deursen A (2010) Connecting traces: understanding client–server interactions in Ajax Applications. In: IEEE 18th international conference, pp 216–225
16. Kimball R, Ross M (2013) The data warehouse toolkit: the complete guide to dimensional modeling, 3rd edn. Wiley, New York
17. Gangadharan G, Sundaravalli S (2004) Business intelligence systems: design and implementation strategies. Inform Technol Interf 139–144
18. Vredenburg K, Mao JY, Smith PW, Carey T (2002) A survey of user-centered design practice. In: Proceedings of the SIGCHI conference on human factors in computing systems, pp 471–478
19. Ammenwerth E, Iller C, Mahler C (2006) IT adoption and the interaction of task, technology and individuals: a fit framework and a case study. BMC Med Inform Decis Mak 6(1):3. doi:10.1186/1472-6947-6-3
20. Heeks R (2005) Health information systems: failure, success and improvisation. Int J Med Inform 75(2):125–137
21. Fonteyn M, Kuipers B, Grobe S (1993) A description of think aloud method and protocol analysis. Qual Health Res 3:430–441
22. Epstein RM, Siegel DJ, Silberman J (2008) Self-monitoring in clinical practice: a challenge for medical educators. J Contin Educ Health Prof 28(1):5–13
23. Iglar K, Polsky J, Glazier R (2011) Using a Web-based system to monitor practice profiles in primary care residency training. Can Fam Phys 57:1030–1037
24. Lyman J, Schorling J, Nadkarni M, May N, Scully K, Voss J (2008) Development of a web-based resident profiling tool to support training in practice-based learning and improvement. J Gen Intern Med 23(4):485–488
25. LogMD. http://www.logmd.com. Accessed Apr 2015
26. Peffers K, Tuunanen T, Gengler C, Rossi M, Hui W, Virtanen V, Bragge J (2006) The design science research process: a model for producing and presenting information systems research. In: Proceedings of the first international conference on design science research in information systems and technology, pp 83–106
27. QuickForms3. https://github.com/uoForms/quickforms3. Accessed Apr 2015