Developing Successful Healthcare Software: 10 Critical Lessons

Quintin Armour, Software Developer, Healthcare Solutions
Didier Thizy, Director, Healthcare Solutions

macadamian

Creating successful healthcare software — from patient management systems, to medical devices, to electronic medical records — differs substantially from traditional software. Healthcare software demands unique domain expertise, project methodology, and software architecture patterns. While many vendors and consultants would insist that good software practices and user-centered design principles are universal across domains, we believe healthcare projects are different. Healthcare domain expertise, combined with software best practices, can significantly improve your project's chances of success.

THE RISKS ARE HIGH. Unfortunately, our industry continually underestimates the effort involved in bringing a new software system or medical device to market. Passing CCHIT certification, for example, can cost upwards of $200,000 depending on the amount of rework involved, according to a comment made by Sam Bowen, president of Open Source Medical Software. Medical device verification and validation can unexpectedly account for a major portion of a project's effort – often making up 33% to 50% of overall development costs.

When the software is finally delivered, it is by no means a guaranteed success. Failed healthcare software stories are everywhere – from Cedars Sinai Medical Center scrapping an EMR system after investing $34M to Kaiser Permanente abandoning an attempt to build its own clinical system with IBM and writing off some $770M in software assets. Even among the small number of physicians in the US who do use some form of electronic medical records within their practice, more than 30% say they would not recommend the software.

Healthcare software projects have a host of industry-specific requirements. They must be well-planned and well-executed to create solutions that are highly secure, flexible, and integrated tightly with the workflow of the wide range of clinical users who will use them.

This white paper provides software R&D teams, software development managers, and product managers with insight into the most important aspects of clinical software development. Focusing on 10 critical lessons, this

paper offers concrete examples and advice gleaned from years of healthcare software development and project management experience. Topics include:

- Architecture for privacy law compliance
- Database design
- Hospital process integration
- Software validation
- Special user requirements

This paper is designed to demonstrate the value that can be gained by understanding the unique requirements of the healthcare sector and by "getting it right" the first time. Proper preparation can ultimately reduce development costs, help end-users deliver better quality healthcare, and improve efficiencies within their hospital or clinical setting.

Lesson One

Requirements Must Be Tailored to a Practitioner's Specialty

THE PROBLEM. Software teams don't always realize the extent to which requirements depend on the specialization or generality of the clinical users. A general practitioner, for example, will not have the same requirements as a physician specializing in cardiology.

When we walked the exhibit floor at a recent HIMSS conference and investigated the 100+ electronic medical record products, we noticed an overwhelming number of "general practitioner" and "one size fits all" software. These solutions will inevitably frustrate any clinical specialist who was sold on the idea that the product was designed for him.

Even if your software is designed for a particular specialty, obtaining accurate requirements in the healthcare domain can be very challenging. Often, those tasked with collecting requirements will be tempted to cut corners in order to avoid taking more time away from healthcare professionals whose main priority is to treat patients.

In cutting corners, assumptions are often made based on past experiences with other projects. This leads to software that does not meet the user's needs.

FOR EXAMPLE. Consider the different types of software that could be used by typical doctors. In a drop-in medical clinic, software records the nature of consultations that take place between doctor and patient. The software must also handle daily patient queues, but precise scheduling is not necessarily required.

On the other hand, a specialist at a teaching hospital requires software that records data from consultations and makes it available to be searched at a later date. These doctors are not only interested in treating patients — they are also interested in evaluating the outcomes for patients with a particular disease, and in detecting trends among the population.

In the drop-in clinic, the software designer can simply write software that provides human readable data. In the second case, the software must be designed to give access to the fine- grained data stored in the back end.

SOLUTION. During the requirements gathering phase, the software team needs to keep in regular contact with healthcare professionals who are specialists in the domain the software is being built. If your team feels it's taking up too much of a professional's time, spend more time synthesizing the requirements so they can be more easily understood.

Presenting healthcare professionals with use cases (not in UML, but in scenario form) that include figures, will help you confirm your understanding of their needs.

Ideally, the people interfacing with the clinical staff should be usability experts with deep healthcare knowledge so that domain terminology can be used. These experts are often the best people for the job since they are trained to ask the right questions and analyze user behavior. Ultimately, they can help to ensure a complete and accurate set of requirements is gathered in a short amount of time.

Lesson Two

Privacy Restrictions Must Be Balanced With Usability Requirements

THE PROBLEM. The development of electronic health records has made patient privacy a top concern of end-users, healthcare organizations, and governments. Governments have created laws (HIPAA, PHIPA) that impose penalties on organizations that fail to protect their electronic data from misuse. These laws are, in turn, applied to other organizations involved the electronic exchange of health information. In 2009, for example, the HITECH act mandated compliance with HIPAA for all covered entities.

From a software developer's point of view, these laws essentially require the implementation of two systems: **access control** and an **audit trail**. While these systems are simple in principle, it is important to ensure they are developed in a structured fashion that addresses privacy laws while maintaining user flexibility.

FOR EXAMPLE. Privacy laws are needed because people have a desire to keep most medical aspects of their lives private. Hospital workers, however, have been known to access and distribute patient records inappropriately. Healthcare organizations require the means to determine whether a privacy violation occurred — and to determine who is to blame.

In recent years, a number of high-profile privacy breaches have come to light and have increased awareness of the need for stricter privacy protections. These breaches include:

- Octomom's hospital records accessed, 15 workers fired
- Britney Spears: Hospital Workers Fired For Looking At Singer's Medical Records
- Sixteen Houston Hospital Workers Fired for HIPAA Violation
- 20 Hospital Workers Fired for Viewing Collier's Medical Records

SOLUTION. When developing clinical software, be sure to balance access control and audit trail mechanisms carefully. The software must respect privacy laws, but should not prevent users from efficiently accomplishing their tasks.

In the healthcare industry, a good audit trail allows for a relaxed access control mechanism. The best access control policies are those that are not overly restrictive — they should offer an override mechanism to prevent inappropriate access. Emergencies arise, and the law allows access to personal data in a life and death situation. In a practical environment such as a hospital, a healthcare professional may have a very good reason to access a record without justifying so beforehand.

Implementing access control is relatively straightforward. Access control restricts actions such as viewing, editing or deleting data elements. Typically, access control is implemented by user role so that all users of a particular role all have the same privileges.

There are two basic design approaches for role-based access control:

1. Access control policies are loaded *per page or screen*
2. Access control policies are loaded *once per user session*

The first approach requires more frequent database access but results in more dynamic access control behavior. Here, although user privileges can be revoked or granted instantly, the need for loading the access control policies can slow down the server — especially when there are many concurrent users. The second approach suffers from a lack of control over access policies, since a user can have his privileges revoked but still be allowed to continue until the end of his session. Either of these policies, however, would be sufficient to address the access control requirements of HIPAA law and even CCHIT certification in the case of electronic medical records.

A third, more advanced method of access control requires the people involved to be defined on a case-by-case basis. A hospital may decide that only medical staff involved in the care of a particular patient should be allowed to access that patient's file. In this case, only staff meeting defined criterion would be allowed to perform any actions on the patient's data. If someone new required access, this user could authenticate himself (the action would subsequently be audited) or have an administrator add him to the system. This method of access control is more precise, but it also uses more overhead than the two basic approaches detailed above.

Access control systems should always allow users to override system restrictions in emergency situations. In these cases, the system should force the user to confirm that he is overriding the controls in place. To ensure this sort of emergency behavior is tracked, a solid audit trail is required.

An audit trail serves as a record of all events that can occur through the normal use of the application. The audit trail records events such as "*user x viewed the record for patient y*." Having an audit trail in place is also of value because it alerts users to be careful when accessing data since they know their actions are being recorded.

There are three standard ways to implement an audit trail:

1. **Save to a log file.** This is normally referred to as an "audit log" as opposed to an audit trail. In this case, actions are recorded in a pre-defined format in a file rather than a database. File system privileges for the file must be restricted so that only a trusted administrator has access. This log file should also reside on storage with redundancy in place.

2. **Save to a database.** This implementation is more in line with accepted best practices, namely audit trail, and node authentication. All actions are stored in a separate database instance used only for audit records. The physical machine should be on a protected network such that only an administrator has access. The format of the audit record can take any form as long as it records the user, the patient, the date and time, and the data that was modified (if applicable).

3. **Record every database action.** It is possible to implement an application such that every SQL command along with the parameters, and the user associated with the session that invoked the command, is saved. This method provides very fine- grained detail, but the low-level nature of the data collected can make it very difficult to trace access breaches.

A good audit trail should also provide an interface through which activities can be monitored, and this interface should generate alerts when data is accessed inappropriately.

Working hand in hand, a reasonable access control approach and a solid audit trail mechanism will ensure privacy laws are respected. Since privacy is a top concern within today's healthcare organizations, these laws — and the steps needed to conform to them — must be taken into account during the software development process. Working with a software development partner with healthcare sector expertise can help you ensure the software includes proper access controls and audit trail mechanisms that do not negatively impact the product's usability.

Lesson Three

Model Hospital Processes Before You Write a Line of Code

THE PROBLEM. Healthcare software needs to map properly to the existing business processes in place, and must not negatively affect or significantly change these processes. Hospital processes are organic in nature and if software places additional demands on the existing process, users will choose to revert to the "old way" of doing things. As a result, it is vital that all processes be understood and documented.

FOR EXAMPLE. Think of hospital processes as a set of conveyor belts. A patient could go through the admission process, the diagnosis process, the treatment process, and the discharge process on the same visit or on multiple visits. The patient will be on one of these process-conveyor belts at any moment in time. Within a hospital there are *many* such processes — each with its own intricacies.

SOLUTION. Each hospital process must be fully understood before any software can be designed to support it. A tried and true method for understanding processes is to do the following:

1. **Model** your understanding of the existing process
2. **Repeat** (as many times as possible):
    a. Present the model to all those involved in the process
    b. Update the model according to the feedback received
3. **Pass the model back** to the software designers

In software engineering, these steps are taken during the early requirements phase and different tools can be used to do the modeling. One that is easy to use is called jUCMNav. It models processes by way of use case maps. You can see an example of this for a patient registration process in the figure below. These maps are translated into requirements, which are then used to design the software.

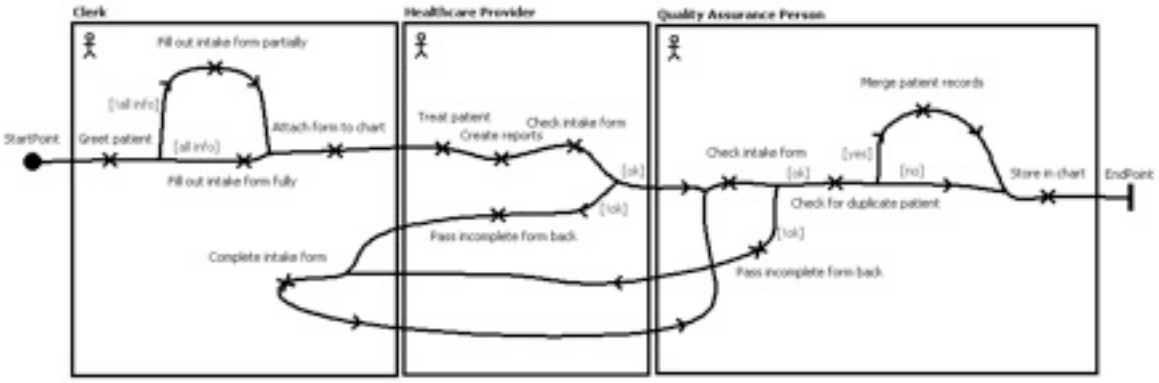


Fig. 1: A jUCMNav example of patient registration

To understand all of the existing processes, it is important to involve a range of end users including someone who understands the processes from a high level, such as a departmental manager. These people can comment on the accuracy of your business process model and critique it.

Many of these users may have never visualized the processes before, and you may even be able to suggest certain optimizations or predict how the processes might change in the future. This can help you gain user trust and become viewed as a valuable partner in the organization's operations.

Typically the business processes manifest themselves within the software as user workflows, where each user must complete a set of tasks before passing the case over to the next user. Therefore, it's important to fully map out user workflows and understand who is doing what — and when.

In our experience, the success or failure of a healthcare application is often tied to the accuracy of the business process models, and on how well the software is able to adapt as processes change. When mapping out healthcare processes, be sure to have a usability expert on hand who is adept at gaining end-user involvement. This expert can help ensure your team gains a solid understanding of the processes that will be touched by your software.

Lesson Four

Doctors Have No Patience For Software That Doesn't Save Them Time

THE PROBLEM. Software developers often create systems without considering the time restrictions healthcare practitioners face. In today's hospitals, healthcare workers are dealing with growing patient waiting lists and need to be able to get their jobs done as efficiently as possible. They can't afford to wait for software and will be unwilling to do so.

In the example of the Cedars Sinai Medical Center given in the introduction, physicians found that it took much, much longer to use the new computer system than it did to do the same ordering with pen and paper. With physicians already working *80 hours per week*, there just wasn't the extra time available.

In some cases, users can knowingly (or unknowingly) shortcut the slow aspects of an application to reduce the time they must wait. This behavior, however, can exacerbate design flaws and lead to even bigger problems.

FOR EXAMPLE. Doctors order diagnostic tests every day. These tests produce reports that are then sent back to the referring doctor. The tests themselves, however, generate data that can be needed by other downstream systems within the hospital, such as billing and inventory management systems.

In a real-life example we have seen, the diagnostic test was a nuclear imaging scan. When the doctor marked the case status as "completed and closed," a report was generated. The data was then passed to the billing system so that the case would be billed for, and to the hospital management system so that inventory counts would be adjusted.

To reduce the load on the diagnostic system that collects the data, the data was transferred to the reporting system, which then acted as the resource for the other dependent systems.

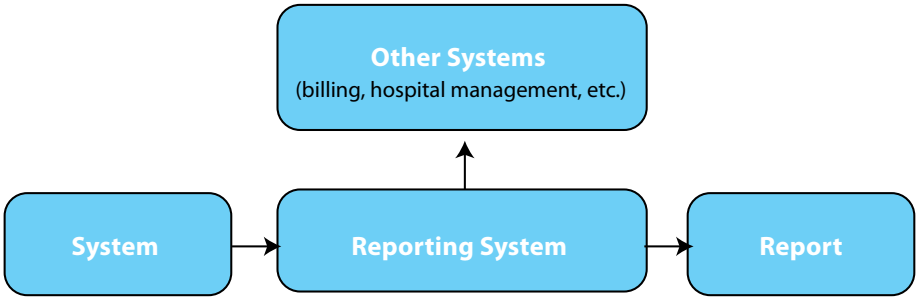


Fig. 2: Example flow of diagnostic test information

A doctor could order the test, walk out of the room, and complete the report based on the information he remembered from the case. When he completed the report, he would close the case.

The problem was that data transfer from the data collection system to the reporting system **stopped** as soon as a case was closed, so in this case, the doctor's actions would accidentally cause all downstream systems to only have access to *partial data*, resulting in inaccurate billing and inventory management.

SOLUTION. To avoid the example scenario, designers of a new system would have needed to be aware of the doctor's desire to get clinical paperwork out of the way as quickly as possible. The system should not have halted the data transfer when the clinical case was closed.

More generally, software systems in healthcare domains need to be very responsive and inform the user about what the application is doing.

In a system with long-running tasks or data transfers, make sure to ask yourself the following questions:

- **Are blocking operations as short as possible?** Short blocking times will ensure users are not kept waiting.
- **Are progress indicators provided and are they accurate?** If users do need to wait, progress indicators will allow them to know how long it will be until they can regain control.
- **Are network data transfers atomic and able to recover from failure?** When large amounts of data have to be transferred, they should be atomic — in other words, they should either happen in their entirety or not happen at all. When dealing with a patient case or a hospital visit, a non-atomic transfer could leave partial data in a remote database. If there is a network error, or the receiving application stops the transmission, it should be possible to resume the transfer without duplicating or overwriting data.
- **Are users aware of impacting operations that run in the background?** When an application features jobs that run in the background, these activities can slow down the normal speed of the program. The user should

be made aware such jobs are running so they do not attribute a lag in the application's performance to an unknown or unresolved problem.

- **Are there any alternate designs to speed up operations?** Sometimes a design may work but not be the most optimal configuration available. In many cases, a simple "rethink" in how things are done can improve operations significantly. Caching interface components, for example, can offer performance gains by eliminating the need to access databases or other networked resources.

By thinking about each of these questions, your team will be able to properly address delays or dependencies in your application and provide the user with clear indications of when he needs to wait and when he can proceed with his tasks.

Lesson Five

Validation Is Far More Important Than Verification

THE PROBLEM. Developers often think application crashes are the most serious problem a healthcare program can exhibit. While a crash is obviously not an ideal outcome, hospital processes are well established to deal with software crashes. Of far greater inconvenience is an application that makes a clinical worker's job more difficult while behaving as designed.

Due to the emphasis on application stability, software teams with little healthcare experience often put too much emphasis on verification and not enough on **software validation**. Software can always be made to do what you want it to do (verification), but it is noticeably harder to be sure you are building the right software for the job (validation). In the healthcare arena, software validation is critically important and while it is easy to allocate time for software validation, the actual validation process requires experience with a thorough validation process, as well as strong domain expertise.

FOR EXAMPLE. Many healthcare applications can be fully tested and agree with requirements, yet be rejected in practice. Such problems are worse than crashes because their root cause is a miscommunication between the client and the development team. Sometimes these issues can be fixed quickly but more often, they require a substantial change, or complete re-think of the problem.

Some validation errors are basic in nature, such as an input box not accepting the value being entered because of a misunderstanding of bounds. A developer, for example, might assume that all blood pressures are numerical and that all blood pressures have both a systolic and diastolic component. In practice however, it is sometimes difficult for a physician to measure the diastolic blood pressure component. In these cases, the practitioner would normally want to enter non-numerical values such as "125/?" or "125/P" for "125 over pulse."

In more serious examples, we've seen software rejected because it lengthened the amount of time a doctor had to spend per patient as a result of it poorly

mapping to the organization's business processes, or because the development team had overlooked the fact that their software had to integrate with a major external system, like an external billing system.

SOLUTION. Problems like those described above can be avoided if they are identified through a rigorous software validation process. This validation period must also be used to test all exceptional and rare (from a healthcare viewpoint) cases so the application can be sure to integrate into the environment in even the most unusual circumstances.

A good software validation process will catch these problems before too much development work has been completed. Moreover, an **agile software development process** can be appropriate for certain types of healthcare application development, such as patient management and EMR software development. With an agile process, the application can be tested and deployed at various milestones of the development cycle. An agile process allows for more opportunities to perform software validation and ensures it is conducted early in the project lifecycle.

Software validation allows users to report on any problems they have seen, and gives users confidence that the application will fit into their workflow and do everything it is needed to do.

While a crash in any application is a serious error, a program that crashes from time to time can still be used if it solves the business need it was built to solve. In practice, these applications are simply restarted or corrected by some kind of workaround.

Software developers need to remember that a program won't crash if no one uses it. Development teams should therefore ensure they have the required healthcare software validation process and expertise on hand whenever they tackle a healthcare project.

Lesson Six

Input Validation Should Not Block Users

THE PROBLEM. Many healthcare applications deal with data entry, and the methods through which data is validated are important software development considerations. In short, treatment cannot be withheld because of the requirements of a software application.

FOR EXAMPLE. A data entry form, such as a form that records patient demographics, typically has a set of required fields. Normally, the patient identifier, first name, last name, gender, phone number, address line 1, province, city, and postal code are required. It is possible, however, that a patient arrives by ambulance at the emergency department alone and is unable to speak. How can the patient be entered into the system if the software stubbornly requires fields not immediately necessary?

In practice, false information is usually entered and corrected once is the true identity of the patient is known. The solution is sloppy, but it gets the job done. In other situations, information cannot be corrected at a later date.

Consider worksheets in which questions are dependent on each other with rules such as "if Question 2 is answered, then Question 3 must be answered too." If the situation arises in which the user answers Question 2, but is unable to answer Question 3, the application and the user can be indefinitely blocked. In the case where Question 3 is a radio button list or a drop down list, any false answer provided will conflict with legitimate answers. Once data like this enters the database, analysis becomes difficult since the two types of answers — the false, and the legitimate — will be indistinguishable.

SOLUTION. Good healthcare applications keep the number of required fields to a minimum, ensure only the most important information is required, or give users the option to skip the input field. When a patient is undergoing a test or is sitting in a consultation room, information may not be known and the delivery of care must be allowed to proceed.

In other cases, a patient may *prefer* to not provide certain information when they do not feel comfortable proceeding with a test or measurement. Since a patient can conceivably reject any portion of a test, every field should be optional in this case. Even if the patient does not stay for a test, the technician or physician must still record the fact that the patient arrived and refused to do the test.

Since other parts of an application or form may depend on fields that are left blank, software developers must ensure the application can still function without error, or alert the user that the data is missing. There must also be provisions in place to manage the incomplete data so it can be entered once it is known. In the case of the patient who cannot speak upon emergency room admission, the information can be obtained from the patient's family, or from the patient once he can speak. If the patient refuses a portion of a test, any automated statistics or reports generated from the data collected from the visit must either indicate the refusal to participate, or be adjusted so that the application can still function properly.

To summarize, effective input validation in a healthcare environment should follow these input validation best practices:

- Input validation should not block users from submitting a form.
- Keep the number of required fields to a minimum. A field should only be required if its absence obviates the form in its entirety.
- Provide defaults or other mechanisms to handle when data cannot be provided instead of requiring the data to be entered.
- Users should be able to skip questions and complete them later, however dependence between questions should be functional so there's no way for one to be answered without the other being answered first.
- Data quality checks should account for incomplete or 'fake' placeholder data and advise users to complete the data entry once the values are known.
- Downstream data consumers must continue to function and raise alerts in case they are fed with incomplete data.

By understanding the range of possible validation errors and the risks associated with them, your team can ensure medical care is never delayed because of a software requirement.

Lesson Seven

Database Designs Must Be Flexible as Workplace Requirements Are Likely To Change

THE PROBLEM. In a clinical setting such as a hospital, workplace processes and requirements are continually changing. New information may need to be collected from patients for legal or medical reasons, new treatment rules may be implemented, and discharge policies could change without notice. IT resources, however, are scarce and healthcare organizations may not have the resources to purchase new software or systems to accommodate these changes. When developing and deploying software, you need to keep these constraints in mind, and architect systems that are able to handle changing requirements.

With the right data model, you can ease these challenges and allow for continual updates as requirements are refined and features are added.

FOR EXAMPLE. If you are asked to create a data model that provides for the storage of patient risk factors, you would naturally create a table to store these values. If additional risk factors were added, the table would be extended, or new tables would be created to accommodate them.

Let's say you need to store the following hypothetical risk factors:

- current smoker (true/false)
- date of bypass surgery (date)
- number of family members with diabetes (number)

With the approach discussed above, the resulting structure would be something similar to the following figure:

TRADE-OFFS

The downside of a more flexible and dynamic data model is decreased performance in large systems, as the database queries are more complex and there are fewer opportunities for indexing. One way to have the best of both worlds is to start with a dynamic database design early-on, when there is a higher degree of uncertainty and the database schema is prone to frequent changes. When the software has matured within the hospital or clinic and the data schema is more stable, you can refactor the data model to boost performance.

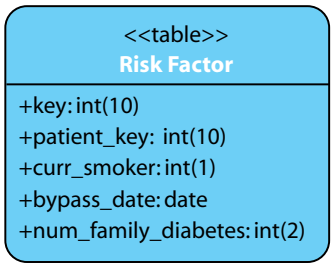


Fig. 3: Basic data model

If you were asked to add a fourth risk factor, you might create an additional column in the table, and so forth.

This approach, however, requires that all programmed models be updated to reflect the newly updated table. This can require significant development re-work, and often thesechanges end up being undone at a later stage, resulting in even more re-work.

Clearly risk factors, and any other set of data fields that may need to be modified frequently require a more adaptable data model.

SOLUTION. As an alternative to the simple data model listed above, create a set of tables to hold a more generic structure of key/value data. Such a structure would look like the model shown below.

This data model stores data as:

- QuestionField: the question text
- AnswerType: the type of answer expected
- QuestionAnswer: all possible answers to the questions
- AnswerValue: a mapping from the patient record to the actual answer values

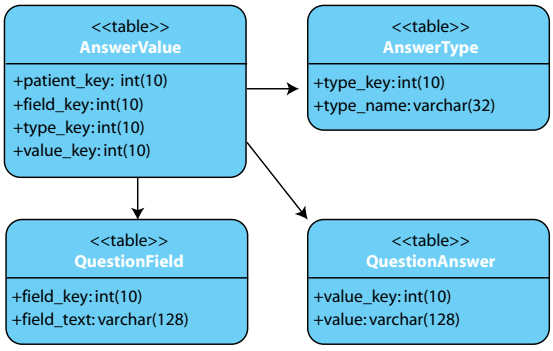


Fig. 4: Generic data model for flexibility

With this model, it now becomes programmatically possible to add fields to an interface of risk factors without having to modify the database schema. With the right data model in place, you will be better positioned to refine the software and handle new requirements as they arise.

Lesson Eight

Hospital Processes Are Highly Collaborative and Asynchronous

THE PROBLEM. Hospitals and clinics contain a large number of autonomous individuals all working to deliver healthcare. There is opportunity for collaboration, but work almost always takes place between caregiver and patient. When a difficult case presents itself, multiple staff members may participate to gain experience, but for common, routine cases work can normally be done without discussion.

For routine cases such as diagnostic tests, communication between team members is normally not necessary. Each member can be given a list of the patients scheduled for that day, and each can work through his or her workflow in the most efficient manner.

If there is no way for a team member to check on the progress of another, however, the caregiver will need to physically ask his colleague if he is done his part of the workflow. If not, he will need to keep checking until the case is ready.

FOR EXAMPLE. Consider the example of the workflow of a cardiac exercise test:

- Patients are scheduled by a clerk who creates the appointments.
- Patients present themselves to the front desk and are directed to an examination Room.
- A technician finds the patient record associated with the patient who is in the room and completes the test.
- A doctor pulls the cases for which the test is complete, interprets the results and completes a report for each.
- An assistant forwards the results of the test to the patient's family doctor and anyone else involved in the case.

Here, each team member is working through his or her own work log. When an employee is finished a case, it then becomes someone else's work, which then gets passed onto someone else until the case is complete. If the system is not able to identify the progress of the case for each of the interested participants, participants will not know when to contribute.

SOLUTION. Traditionally, a piece of paper has been used to synchronize team members. If you had the piece of paper, it meant you were the one who had to work. However, since it would take a while before the paper arrived on your desk, getting the paper implied you had to do a lot of work, and as soon as possible. It is much more efficient to let people work when they want, and do as much work as they want.

For this reason, healthcare software should support a workflow process where each member is empowered to have autonomy over tasks, but is able to get a sense of how many tasks need to be completed.

Each user may have drastically different work patterns. In the example above, clerks and technicians work on a daily schedule, whereas the physicians may work on a weekly or an "every couple of days" schedule. Physicians are always getting interrupted and distracted, and so their schedules are usually more erratic.

An employee who is not able to do all of her work at once, for example, can do as much as she has time for and leave the rest for later. Even if doctors can only interpret results twice a week, this does not block an assistant from forwarding the test results for the cases already interpreted. The assistant can do other work and then receive a notification from the system when the doctor has completed more cases.

Communication between healthcare professionals is largely asynchronous. This asynchronous user behavior needs be understood by developers so that the resulting application maps to the existing workflow process.

Lesson Nine

Supporting Interoperability Standards Is Tougher Than It Looks

THE PROBLEM. Most healthcare applications need to interface with external systems. Since these interfaces are usually complex, standards have been developed to simplify the process.

Implementing code that adheres to the standards requires a lot of time, and is often a source of overruns and rework. It is crucial to ensure the design and implementation of interoperability standards is carefully planned and estimated.

FOR EXAMPLE. There are many standards for coding, formatting and exchanging healthcare data across healthcare IT applications. Electronic Health Records alone use the following array of standards:

- HL7 v2.x, 3
- X12 4010, 5010
- CCR, CCD
- ICD9/10, SNOMED
- RxNorm
- UNII
- UCUM
- NCPDP script 10.x
- CVX

For example, a patient could visit a hypertension clinic that records medications in RxNorm format, records active diagnoses in ICD10, and produces a CCD record from the visit. The document is then transferred via HL7 interface to the patient's family doctor and to the hospital's EMR system.

Different systems interoperate using different standards, so applications often need to support many different standards. An EMR may need to exchange data with a patient registry that expects patients' records in CCR format, a physicians' portal which expects a CCD format, and a billing system which has its own custom format. What's more, those standards are continually changing and evolving.

SOLUTION. The first step to ensure the interoperability standards code is well-planned and well-estimated is to account for all the possible tasks that will be required of the software team during implementation. These can include:

- Design planning to ensure that past, current, and future versions of each standard are supported.
- Documenting test cases for full range of possibilities.
- Design handling of no-match cases so that these can be identified and fixed (if a received ICD-10 code does not match any real code, for example).
- Writing unit tests to ensure continued compliance with standards.
- Mocking remote end incoming messages and responses.
- Implementing logging of all exchanged data.
- Handling message replay needs in case of error.
- Spending time with the remote endpoints to conduct end-to-end testing.

Once you have very thorough, granular task breakdown and estimates, you have solved half the problem. The other half is avoiding the high risk of rework as the software evolves.

Without an automated mechanism to test the code, and without proper separation from other parts of the application, maintaining compliance with the standard can quickly become unmanageable. Code regressions can be introduced and quality assurance personnel will have to continually test the application to make sure the interfaces to the standards still work.

The best software teams make sure the program code is properly encapsulated and create automated unit tests so that the code that deals with the standard can remain in place until the next version of the standard is released. This also ensures unrelated features that are added to the software over time do not introduce regressions in the standards- specific code.

Interoperability is one of the biggest challenges in healthcare IT today. A plethora of standards have emerged to address this issue, but there is a challenge in implementing and managing the code that implements these standards. By understanding the full range of implementation tasks, designing for encapsulation, and developing automated unit tests, you can avoid cost overruns upfront and over the course of the software lifecycle.

Lesson 10

Not All Clinical Users Are Doctors And Nurses

THE PROBLEM. Developers often focus on the needs of doctors and nurses since, traditionally; these are the most difficult groups to please. Doctors and nurses are most pressed for time, and their actions most directly affect the quality of patient care. There are, however, other groups of users who will directly or indirectly use a healthcare application. These include:

- Hospital administrators and executives
- Middle management (each department/clinic has a manager)
- Physician-run office workers
- Physician or independent researchers

Often, physicians are the ones who request the development of an application and are the only ones contributing to the requirements. When the software is released, however, it can fail to deliver an improvement in the hospital's process because it does not deliver what is needed to all those who need it.

FOR EXAMPLE. An application that is used by doctors and nurses, but has greater benefits for managers and administrators, is a departmental clinical triage system. This application receivhzzes referral requests, and implements policies to prioritize, and schedule patient visits.

Clerks and nurse managers use the application to publish and organize patient lists for any given day. When the cases are completed, the user marks them as closed, and enters any relevant remarks.

Physicians, on the other hand just use the system to consult on patient cases and patient details.

Since all cases go through this system, it can be a useful tool for reporting statistics to managers and administrators. Some examples of relevant statistics include:

- Patient volumes
- Case types
- Wait time statistics
- Number of cancellations

If the solution was designed with only the doctors and nurses in mind, it would be unlikely to include statistics-reporting functionality. Without this ability, the software could be quickly shelved because it would not satisfy the needs of all the users.

SOLUTION. Software teams need to be sure to not only satisfy the requirements of healthcare providers, but also of users who may not use the application directly but who still depend on it to get their work done.

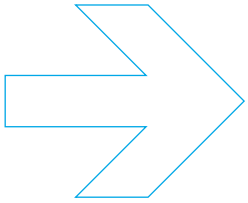
It is important to consider those who can benefit "downstream" from the data collected by the application. Be sure to have a usability expert with healthcare experience on hand who can help you ask the right questions, and obtain the right requirements from all the people affected by hospital processes. This process is known by usability experts as "stakeholder identification." A usability expert can help you model the people, groups, and institutions that contribute to — or will be affected by — a change to hospital processes. These experts can also identify the array of user personas within an organization to help you solidify your understanding of who, exactly, will use the application.

With a solid understanding of all potential users and their needs, you will be better positioned to deliver relevant, useful software that solves the problems of the healthcare organization as a whole. Not only will this build your reputation as a trusted software development partner, it can help the organization justify its software development costs and ensure value is being delivered across the organization.

Conclusion

DEVELOPING CLINICAL PRODUCTS can be complex and fraught with risks. The healthcare industry presents a set of unique software development challenges that require domain expertise, foresight and specialized knowledge.

By understanding — and preparing for — the most commonly encountered issues in clinical software development, you can avoid typical mistakes, reduce schedule, and cost overruns, and help healthcare practitioners deliver better care. By developing flexible, software that is adaptable to clinical processes, you can ensure your clinical end-users will be able to leverage well-designed systems that meet both their current and future needs.



macadamian

About Macadamian

MACADAMIAN IS A GLOBAL LEADER in software product creation providing a complete range of product strategy, user experience design and software engineering services to clients around the world, including healthcare industry leaders like Cardinal Health, Elsevier and Telus Health. For established and emerging technologies and platforms, Macadamian has a track record of helping clients create successful products on iPhone, iPad, Android, Web 2.0 & RIA, Windows 8 and Mac OS X.

Macadamian's medical software development team understands the unique requirements of the healthcare industry, its standards and its regulatory requirements. Our experts are well versed in FDA, HIPAA, ARRA and CCHIT requirements and the environments in which healthcare devices are used.

We also know that usable, well designed software can differentiate you from your competition and drive the adoption of your medical software product. Over the past twelve years, our engineering, usability, and visual design teams have worked with clinicians, doctors, nurses, and domain experts to develop a range of healthcare products that integrate with their workflow and addresses specific clinical needs.

At Macadamian, we follow proven processes to provide the skilled, speedy and high-quality development you need to deliver on your commitments. Our healthcare experts can help you tackle your clinical software development project and ensure your product will be embraced by users when it hits the market.

Additional information can be found at www.macadamian.com.

Contact Us

FOR QUESTIONS OR COMMENTS about this whitepaper, or for more information about our healthcare software design, and development services, please contact:

Didier Thizy, Director, Healthcare Solutions
165 Rue Wellington
Gatineau, QC, Canada
J8X 2J3
didier@macadamian.com
+ 1 613 739 5976 x136
www.macadamian.com