intelliware
software development

Healthcare

The HL7 Games: Catching FHIR

Healthcare Standards after v3

FHIR™

It should come as no surprise to anyone involved with HL7 v3 implementations in Canada that the v3 standard is now almost-universally considered to be a failure. As an e-Health implementer, I've certainly felt the pain of HL7 v3, but I held my nose and used it anyway because I believed in the original goals of v3: standards-based data integration and electronic health records. They're goals I still believe in.

The good news is that I don't have to hold my nose any longer. HL7's Fast Healthcare Interoperability Resources (FHIR™) has learned from the mistakes of HL7 v3, and is surprisingly delightful.

At the high level, I'd argue HL7 v3 failed for these reasons:

1. The models and messages of HL7 v3 were a much higher degree of complexity than was found in almost any other XML-based messaging standard.
2. That complexity could not be easily mitigated with easy access to clear documentation, as implementation documentation needed to be cobbled together from many different sources (e.g.: "universal" documents from the HL7 organization, pan-Canadian standards from Canada Health Infoway, and jurisdiction-specific documents from a repository implementation team).
3. The HL7 v3 standards lacked a useful extension model. The official process of "constraining" the standard for a particular jurisdiction was seldom sufficient, and essentially all of the jurisdictions ended up implementing systems that diverged from the published standards to support legitimate operational needs.
4. The defined HL7 v3 data types introduced a number of complications for implementing compliant HL7 v3 systems.
5. Finally, terminology is a surprisingly complex area.

This document will discuss some of these topics in more detail, and describe some of the ways in which the FHIR™ standard has corrected the failings of HL7 v3.

> "I believed in the original goals of v3: standards-based data integration and electronic health records. They're goals I still believe in."

> "HL7's FHIR™ standard has learned from the mistakes of HL7 v3, and is surprisingly delightful."

Complexity

That the HL7 v3 standard is exceptionally complex must, at this point, be taken as a given. Intelliware was an early Cassandra on this topic:[1] we reported this finding back in a 2007 experience report after implementing some of our first HL7 v3 point-of-service applications and, even then, we were echoing similar sentiments being reported by implementing jurisdictions at Standards Collaborative conferences. At this point, I am confident that no one who is experienced with the HL7 v3 standards doubts their complexity.

[1] HL7 Experience Report. BC Holmes, Intelliware Development. Nov. 15, 2007.
https://www.infoway-inforoute.ca/index.php/component/docman/doc_download/506-hl7-referrals-intelliware-experience-report

One way of quantifying this complexity is to look at the density of messages. The average HL7 v3 message has 5 to 10 times the number of XML nodes as average messages for other common XML standards, such as Interactive Financial eXchange (IFX) or the Amazon EC2 SOAP API. One could make the argument that the business of health care is more complex and semantically rich than the business of banking or book sales and therefore justifies that additional information density. For my part, I'm not persuaded that that's the issue: I've had the experience of upgrading a system from XML-based HL7 v2 to HL7 v3, and seeing the density increase by an order of magnitude. The tendency toward very dense messages seems to be a peculiar feature of HL7 v3. In any event the broader point remains: when implementing an HL7 v3 system, developers are suddenly confronted with a degree of complexity far beyond anything they're likely to have encountered prior to that.

> "The average HL7 v3 message has 5 to 10 times the number of XML nodes as average messages for other common XML standards."

In the following sections, we'll discuss some of the specific areas of complexity seen in the HL7 v3 standard, and some of the mitigations that FHIR™ has taken to avoid those complexities.

The Toaster Problem

There's a frequently-told joke that takes a jab at software development thinking. In this joke, a king summons two advisors, an electrical engineer and a software developer, to come up with a design for an embedded processor for a toaster. The electrical engineer rattles off a simple design, but the software developer tries to abstract the problem.

The software developer argues: "'Toasters don't just turn bread into toast, they are also used to warm frozen waffles. What you see before you is really a breakfast food cooker. As people become more sophisticated, they will demand more capabilities. They will need a breakfast food cooker that can also cook sausage, fry bacon, and make scrambled eggs. A toaster that only makes toast will soon be obsolete; we must therefore look to the future, or we will have to completely redesign the toaster in just a few years." Soon the software developer is describing a complex breakfast preparation system powered by Unix.

By the end of the joke, the king has had the software developer beheaded.

The reason IT professionals find this joke funny is that it captures a real phenomenon that happens in the software development world: a thing that we could call "the toaster problem." Developers try to consider too many cases, and try to support a lot of those cases through abstracting a problem, even though the abstraction often leads to diffused clarity of purpose, and the implementation suffers as a result. The adage "Jack of all trades, master of none" describes the same idea.

This kind of toaster problem rears its head in the HL7 v3 standards. I think that one of my favourite examples of the toaster problem relates to the Location Registry. Imagine this scenario: a doctor needs to send a patient to a blood lab for blood work. To find an appropriate lab, the doctor consults the Location Registry to find blood labs that are in convenient locations. What's convenient? Well, usually this boils down to where the location is physically located and the

Internet tends to use Latitude/Longitude coordinates to help answer that question.

The HL7 v3 standards that support these kinds of Location Registry functions include spatial coordinates. But rather than merely include a latitude/longitude value, the modellers have abstracted the problem. Presumably they imagined that some other coordinate system might someday replace latitude/longitude values. So this "basic spatial coordinate" is modelled as a position object with a coordinate type (e.g. "Latitude/Longitude"), and up to five position coordinates (because who knows if two is going to be enough?) Naturally, each of these coordinates needs a code to describe what type of coordinate "axis" it represents. One coordinate axis could be "latitude" and another could be "longitude".
To be fair, there are other coordinate systems in the world, but lat/long seems to be the most general-purpose coordinate system: other systems are generally reserved for very specialized domains. I would be highly surprised if any healthcare application used a different coordinate system in the next two decades.

As a developer trying to produce correct HL7 v3 messages, it's not enough to put the "latitude" value into the "latitude" field; now I need to figure out what the valid codes are to represent the "lat/long" coordinate system as well as the valid codes to represent each of the coordinate axes. And that's over and above any additional work I have to invest to figure out this abstract way that spatial coordinates have been represented. The abstractness of the "coordinate system" concept just seems to make my job as a developer unnecessarily more laborious. (This kind of abstractness also flies in the face of agile practices, so I have multiple reasons to dislike it).

One of the guiding principles of the FHIR™ standard is the 80/20 rule — a rule that
states that elements should only be added to the standard types if 80% of the systems use them. This principle exists precisely to avoid wishful thinking about the future that doesn't benefit current implementations.

Now, some might fear that the 80% rule would discount data elements that might be crucial to a very specific application or healthcare domain. That fear should be assuaged by the well-defined extension capabilities of FHIR™. Basically, any data elements can be added to a system as an extension. To be interoperable, other systems must elect to support the same extensions, but that's an entirely manageable process.

"One of the guiding principles of the FHIR™ standard is the 80/20 rule — a rule that states that elements should only be added to the standard types if 80% of the systems use them."

Unnecessary Cruft

HL7 v3 messages often contain a lot of extra "cruft", the most glaring example of which relates to some artifacts of the modelling exercise: these generally show up in the XML messages as "moodCode" or "classCode" attributes of XML elements. 99% of the time, these attributes provide no meaningful value in the message; they're usually constants that almost-never get inspected during any domain logic processing.

[2] Introduction to HL7 FHIR™. James Agnew, University Health Network.
https://docs.google.com/presentation/d/1MNGs1HOiHzH820OXLEJVYzG0lBpsIsneQlNsagZD-ec/edit#slide=id.p7

I'm tremendously pleased to see that one of the streamlining processes that the FHIR™ standard has gone through is the removal of these "structural" attributes.

Naming Problems

The University Health Network's James Agnew once commented that while HL7 v3 was intended as a modeller's paradise, it became a developer's hell[2]. To some extent, the way the HL7 v3 definition process supported the modellers lead to the "unnecessary cruft" problem, described above. A more painful issue relates directly to how things are named in HL7 v3 messages. Ever since my first exposure to HL7, I've been amazed at how badly things are named in the standard. Too many things have overly-vague names like "Assigned Entity" and, for different reasons, many things are given bad names like "Component5" or "SubjectOf3."

If you look for common healthcare terms like, say, "prescription," you're unlikely to find any message part called that. You may find some messages that bear the name "Activate prescription request" and the like, but you'd have to pay close attention to the type names in that message and recognize that the "prescription" part is actually called a "combined medication request."

While many of these unhelpful names originate with HL7 v3's tendency toward abstractness and handling all possible cases, some of them are artifacts of the diagramming tools that HL7 v3 has provided to its modellers. One of my favourite examples of this has to do with how many message parts will have an association with a name like "component6" but the type of the relationship part is "Component4". This oddity happens because the modelling tools don't really expose the name of the relationship object when dealing with relationship types (such as ActRelationships), and the modellers never see the misalignment between the association name and the relationship part name.

Thankfully, readability of the fields of a resource is a primary consideration of the FHIR™ standard — even to the extent of deprecating terms like "dob (date of birth)" in favour of more internationally-recognizable names like "birthDate."

Peeling The Onion

One of the nicest features of FHIR™ is the flatness of its message-representation. While rich data usually calls for a rich representation, the deeply nested structure of HL7 v3 data has been a pain for some time.

Consider the v3 clinical document messages. These HL7 v3 messages are intended to operate similarly to Clinical Document Architecture (CDA) documents and, in general, are ways of recording common documents that healthcare systems already produce: these documents take the form of PDF documents or, more often, images (such as for X-rays, heart-rate graphs, or even fax outputs).

> One of the nicest features of FHIR™ is the flatness of its message-representation. While rich data usually calls for a rich representation, the deeply nested structure of HL7 v3 data has been a pain for some time.

For the most part, this kind of healthcare record seems fairly easy: it's the sort of thing that many organizations might put on SharePoint or a similar low-cost document sharing tool. But its representation in HL7 v3 is actually a deeply-nested document: the image or PDF is wrapped in some additional data envelopes that attempt to provide information about any structure that can be attached to the unstructured binary, and that's wrapped in a meta-data envelope that indicates information about authorship, dates, and associated patient (and a bunch of related meta-data, like information about previous revisions).

> By contrast, FHIR™'s representation of data is amazingly lean.

And the nesting doesn't end there: that overall "document" structure is itself wrapped in some constructs that provide processing information about the document: standard wrappers called the "transport wrapper" and the "control act" which are used for routing and access control purposes. And lastly that resulting HL7 v3 message is usually wrapped in a SOAP envelope, which handles such considerations as digital signing, authentication, routing, etc. And all of that could be said to be wrapped in a mystery inside an enigma.

The end result involves XML messages where nodes are sometimes nested thirty or forty levels deep (or more!). Some of those levels of nesting arise from the complexity of the models, including cases where models are more complex than needed to support abstractness. Some levels are a result of the extra "subjectOf" parts that are a consequence of the modelling tools. Some levels of nesting are directly tied to these wrapper parts. But the end result is too much nesting.

Each layer of the onion adds a comprehension cost to the data being transmitted: that degree of nesting often contributes palpable confusion to the job of implementing. In our experience it often takes developers a few years of working with HL7 v3 before they know how to mentally make sense of the "important parts" of any particular message: that can't be in any implementation project's interest.

> In short, FHIR™ produces a lighter, more accessible data packet that developers can understand more easily. And the benefit of that to implementation is enormous.

By contrast, FHIR™'s representation of data is amazingly lean. The most extreme examples of complex data may be nested ten nodes deep, and in most cases, the essential data is within 5 levels of the root.

Various HL7 v3-specific wrapper specs have, instead, been replaced with modern web standards like OAuth. Even the problem of "continuation" (e.g. "get me the next 50 patients that matched my search criteria") has been implemented using an existing standard — in this case, the Atom standard.

It's additionally true that FHIR™'s flat structure helps to obviate some of the data redundancy in the messages: in HL7 v3, it was entirely possible to reproduce certain data parts multiple times in the same message — this is especially common with providers (doctors, et al) who participate in many ways in the context of a particular message, and medical interaction data, which often reproduces a lot of information about prescriptions. In the worst cases, those redundant data segments added their own deeply nested structures, and at best they just made the resulting messages that much larger.

In short, FHIR™ produces a lighter, more accessible data packet that developers can understand more easily. And the benefit of that to implementation is enormous.

Message Variance

Because work on HL7 v3 began so much earlier than so many modern XML APIs, HL7 messages have a different "feel" than many other XML messaging APIs.

For my part, my best way of expressing that different "feel" has been to describe it this way. Most XML APIs that I've experienced tend to have a relatively static structure for any particular messaging transaction. By that, I mean, if I have a transaction called "Get Account History" the resulting response message is likely to have a fairly straightforward and structurally consistent format. It might contain a collection of credits and debits; the number of records that are returned may vary; and there may be optional bits of data that appear or don't appear based on data-related rules. But there is usually a sort of consistency to the resulting message that allows the developer to easily comprehend the resulting response message and write code to support the response. I think that most people who develop these kinds of APIs tend to design messages in such a way that minimizes variability in the structure of the responses: in the cases where you need structurally different types of data, you support multiple different types of request/response pairs.

> "The single greatest thing that FHIR™ could have done (and did!) is make the FHIR™ documentation openly available on the HL7 web site."

By comparison, HL7 tends to take the opposite approach: in an HL7 system, the number of individual request/response pairs is relatively low (a typical system might support 25 to 75 request/response pairs) but the variability in the structure of the individual messages is high. For example, one of the more complex messages is the "Get Clinical Profile" message, the response for which produces a list of healthcare records. But any individual record could be a prescription, a lab report, a clinical document, an immunization record, etc. Each of these record types has a radically different structure, and implementing code to support all of these different record types is a heavy workload.

I've intentionally chosen one of the most complicated messages to make this point. The feature of HL7 message definitions that enables that "monster message" -- a feature called "choices" or "abstract message classes" -- is used in almost every message type defined by the Canadian Standards Collaborative process.
One can also have a high degree of variability, even without choices. Some messages are just designed in a way that expects very different parts of the message to be populated based on different usage scenarios.

One of the consequences of this high degree of variability has been that HL7 v3 messages have been extremely hard to manage using commercial XML mapping/transformation tools that commonly accompany enterprise service bus products or integration servers. The message variability seen in HL7 v3 messages results in a need for highly complex mapping — a degree of complexity that not many of those products are able to accommodate easily.

Documentation

Documentation has always been one of the most painful aspects of HL7 v3. In other technology domains when documentation is an issue, it's because there isn't enough.
But for HL7 v3:

1. there's a veritable mountain of documentation;
2. it's produced by a large number of different organizations;
3. not all of the documentation is easily-accessible for the average developer to access (and is often hidden from Google behind paywalls);
4. it's hard to tell what versions of documentation are still valid and applicable (and related to each other); and
5. a lot of it is badly written, especially because it describes things in "concept-speak" rather than "example-speak".

The single greatest thing that FHIR™ could have done (and did!) is make the FHIR™ documentation openly available on the HL7 web site. The documentation is additionally far more accessible, not just from a security and permissions point of view, but from a comprehensibility point of view. Possibly because one of the FHIR™ principles is to focus on actual implementations, the documentation is far more concrete and examples-based than the comparable HL7 v3 documents.

Extensibility

> "I'd argue that extensibility is going to become one of the most useful features of FHIR™ going forward."

In general, in the HL7 v3 world, the modellers define messages that describe a particular health care interaction, and publish a specification in the form of a Model Interchange Format (MIF) file. Developers then take those MIF files and implement the messages in the healthcare system that they're currently building. Often, at that moment, a particular need is discovered — the need to pass some kind of data field that is not currently represented in the message spec — and that becomes a major issue.

In HL7 v3 implementations, in the best case, getting a change to a message definition is a days- or weeks-long exercise to re-engage the modellers and get the spec updated. In the worst case — such as with major government Medication Management systems — the implementing team is working with a specification that was published years previously, using a months-long review and publication process. Such implementations are rarely using the latest versions of the pan-Canadian HL7 v3 standards, and the likelihood that the modellers would return to such an out-of-date specification was almost zero.

There really was no well-defined extension mechanism in HL7 v3. The pan-Canadian Standards allowed for a "constraint" mechanism — one by which individual jurisdictions could increase the mandatory-ness of certain fields on messages, but it didn't really support the addition of new fields or message parts: the intention was always that the resulting message was still structurally readable by another Canadian jurisdiction.

Perhaps because of the earlier-mentioned 80/20 rule, FHIR™ has implemented a pretty good extension mechanism. The principles are fairly straight-forward:

1. Any resource can be extended. The standard definition for a patient resource contains the data fields that made the 80/20 rule cut. If additional data, like a photo of the patient, is needed, that can be included as an extension field.
2. Extensions are defined as simple key/value pairs, but the keys are URLs. The intention is that these URLs, when resolved in a browser, provide documentation about the meaning of the extension.
3. The "values" can be any standard FHIR™ data type.
4. Finally, rich data types can be created by nesting extensions.

There's even a notion of "modifier extensions" which have some large-scale implications for the resources. For example, a modifier extension could extend a prescription resource into an "anti-prescription": an instruction that the patient shouldn't take a particular medication. A prescription is a well-defined resource, but an additional "anti" flag has a fundamental change of the prescription concept and systems should process a prescription with this flag differently. To indicate this fundamental change, the "anti" extension should be recorded as a modifier extension.

I'd argue that extensibility is going to become one of the most useful features of FHIR™ going forward.

> "Until we see some experience reports from large-scale FHIR™ implementations, we can't really know if FHIR™ will do a better job of supporting the needs of true interoperability than HL7 v2."

Data Types

Some of the earliest work on the HL7 v3 standard started before most modern standards, such as XML, existed. Despite that, XML is the langua franca of HL7 v3 messaging. But because of v3's history, HL7 v3 has never aligned well with XML.

Data types are one of the key areas where this misalignment can be seen. For example, HL7 v3 defines standard primitive data types, such as String (ST) and Bool (BL) which are just different enough from the usual XML definitions of those concepts that most XML development tools end up being much harder to use.

FHIR™, conveniently, has learned from this experience: the most common primitive types (boolean, string, date, instant, base64Binary, decimal, uri, etc.) are mapped directly to comparable XML types, and more complex types (Address, Coding, Quantity, HumanName, etc.) are built up from those well-understood primitives. This take on data types gives FHIR™ a few advantages:

1. The new data types are better supported by XSDs, which can evaluate the correctness of the XML content more completely;
2. Common XML marshalling and binding tools are suddenly more viable to use;
3. XML transformation technologies, such as are common with Enterprise Service Buses and integration engines, are better able to support FHIR™ messages.

One of the side-benefits of this realignment is that it's easier to perform operations such as message validation and testing support. Creating a sufficient-

ly rich testing harness for HL7 v3 is a hard exercise (and we built one, so we think we have some insight). FHIR™ is far easier — so much so that such test beds have already appeared.

Terminology

Terminology is the representation of certain healthcare and other concepts in coded values, such as 'en' for 'English' and 'fr' for 'French.' A great deal of work in healthcare system implementations is spent making decisions about what types of codes to use. Will we represent conditions using ICD-10 or SNOMED? Will we use DINs to record prescribed medications or RxNorm? Often HL7 v3 implementations — especially implementations that concern themselves with interoperability — need to figure out what the best codes to use are, and also how to map any internally-used codes to the officially-chosen interoperability codes.

HL7 v3's focus on terminology was a direct result of v2's limitations when it came to interoperability: v2 allowed for structurally-consistent messages, but messages produced by different systems rarely spoke the same language when it came to encoding healthcare data. While wrestling with terminology is complex, it's a complexity borne out of need.

FHIR™ doesn't change much when it comes to terminology, although it does obviate some uses of codes in messages. In fact, I would argue that many of the obviated codes could be thought of as cases where the HL7 v3 creators applied the rigour of v3 terminology in places where it might have been better to choose something simpler. For example, FHIR™ removes the following types of terminology:

1. HL7-specific codes used in the transport layer and control act;
2. the codes that would have been used in "mood codes" and "class codes" throughout the message; and
3. codes that were specific to some HL7 data types, such as codes to distinguish the "zip code" part of an address from the "street name" part of an address.

I would argue that this is an example of FHIR™ keeping "the good parts" of v3 terminology, while dispensing with unnecessary complexity.

Ceci n'est pas une Pipe and Hat

When talking about the previous HL7 standard, version 2, one could successfully argue that it hasn't really flourished as an interoperable standard so much as a peer-to-peer standard. Sure, there are some HL7 v2 integration engines, and many-to-one client/-server implementations (generally inside the context of a single organization's network), but HL7 v2 messaging implementations tend to be built around the needs of a source system, and the integrating systems adapt to those needs. No one really expects these systems to produce messages that are consumable in other contexts.

Until we see some experience reports from large-scale FHIR™ implementations, we can't really know if FHIR™ will do a better job of supporting the needs of true interoperability than HL7 v2. It may be that FHIR™ becomes a standard that merely brings HL7 v2-like messaging into the modern, REST/JSON present. And if that's true, I'd argue that FHIR™ is still an advancement over both HL7 v2 and HL7 v3. Certainly mobile apps are one of the fastest-growing areas in the healthcare space: the v2 (and v3) standards look especially outdated to the average mobile developer.

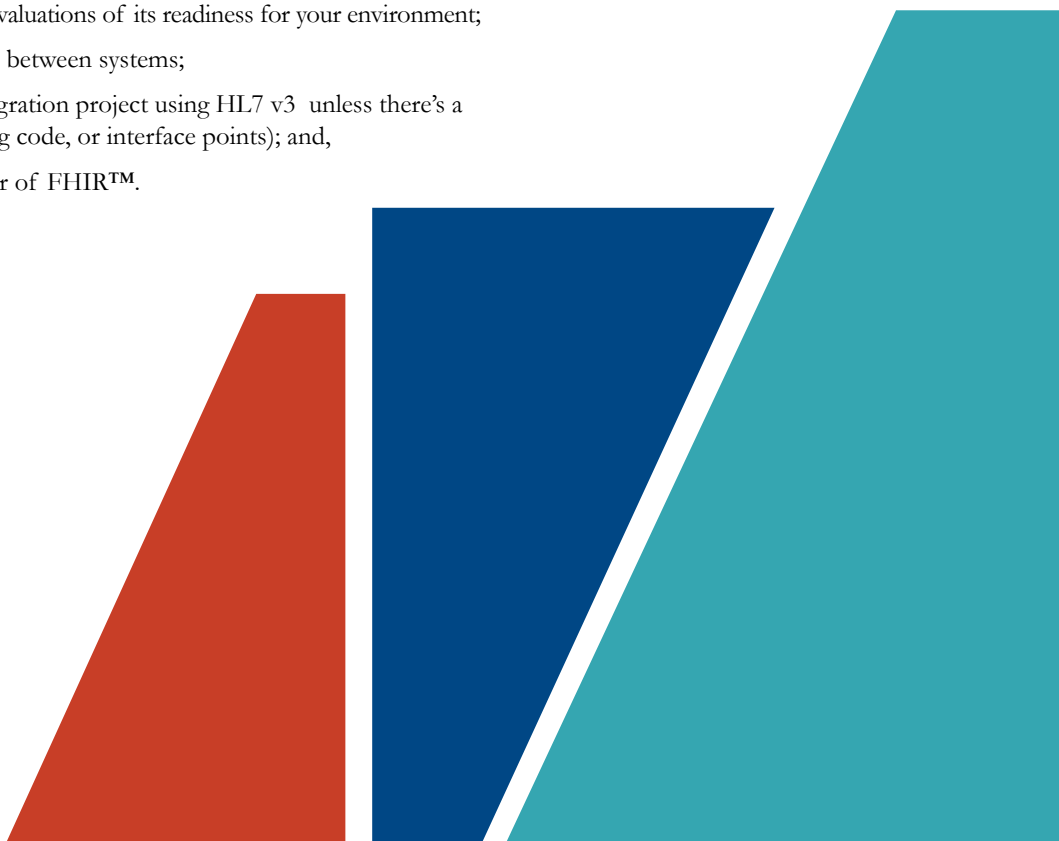
Conclusions

Even for its newness, FHIR™ already seems like a more attractive alternative to HL7 v3. It has learned from the mistakes of its predecessor and has been developed around some excellent guiding principles. Further, it has been built upon some excellent modern standards, such as REST, JSON, OAuth, and even (oddly) Atom.

While there are still some outstanding questions about how easy it'll be to support big, jurisdictional registries and database systems using FHIR™, I think there's every reason to believe that FHIR™ is a great starting place for small-scale healthcare integration projects, such as mobile apps for in-hospital data or communication between systems within a healthcare organization.

We recommend that organizations:

1. Evaluate FHIR™'s applicability to your systems — a small proof-of-concept project might be one of the most meaningful evaluations of its readiness for your environment;
2. Favour FHIR™ over any one-off API between systems;
3. Avoid starting any new healthcare integration project using HL7 v3 unless there's a compelling reason (e.g.: already existing code, or interface points); and,
4. Consider deprecating HL7 v2 in favour of FHIR™.



Author's Bio

BC Holmes Chief Technologist
BC has over 20 years of software development experience. She has extensive experience leading solution architecture and delivery of health informatics systems integration and product development. She led the development of the Infoway Message Builder API, an HL7 v3 library used by multiple vendors across Canada to exchange Healthcare data.

BC has a Bachelor of Math Degree from the University of Waterloo.

About Intelliware Development Inc.

Intelliware is a custom software, mobile solutions and product development company headquartered in Toronto, Canada. Intelliware is a leader in Agile software development practices which ensure the delivery of timely, high quality solutions for clients. Intelliware is engaged as a technical partner by a wide range of national and global organizations in sectors that span Financial Services, Healthcare, ICT, Retail, Manufacturing, and Government.

Intelliware placed among the Top 5 Mobile Technologies Companies in the 2012 Branham300 report, the definitive listing of Canada's Information and Communication Technology (ICT) industry leaders, as ranked by revenues.

About Intelliware's Healthcare Practice

Intelliware's Healthcare practice specializes in systems integration, product development and outsourced custom software development. We work with healthcare organizations of all sizes, ranging from startups to large enterprises. Our clients include: HealthPRO, Canada Health Infoway, SickKids, the University Health Network, and the International Health Terminology Standards Development Organisation (IHTSDO).

To get in touch with our Healthcare team, please contact Ken Stevens,
Vice President, Healthcare, at 416.576.4096.

200 Adelaide Street West, Suite 100
Toronto, Ontario M5H 1W7, Canada
(416) 762-0032

www.intelliware.com

/company/intelliware-development-inc-
/intelliware.inc
/intelliware_inc
/GooglePlusIntelliware