MTR090331

MITRE TECHNICAL REPORT

MITRE

L32 Test Plan and Test Results

September 21, 2009

This page intentionally left blank.

MTR090331

MITRE TECHNICAL REPORT

MITRE

L32 Test Plan and Test Results

Dr. Adriane P. Chapman
September 21, 2009

Sponsor: MITRE
Dept. No.: E549
Project No.: 19SPI920-FA

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

Approved By:

Mary K. Pulvermacher
Project Leader, Project 19SPI920-FA

Date

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

Abstract

Electronic health data exchange is a key element of health care reform. Current Federal mandates require that health data be exchanged electronically using recognized standards. The C32 Specification is the recognized standard for exchanging Continuity of Care information. We propose a tool called L32 for "Lightweight C32 Implementation" to accelerate and lower the cost of electronic health data exchange in accordance with the C32 Specification. L32 is intended to be a single, consistent, machine-interpretable implementation of the C32 Specification. In order to demonstrate that L32 is a viable option for representing C32 documents, we compiled a list of possible benefits and concerns of using L32 instead of the C32 Specification. We then performed five distinct experiments, containing a total of eleven tests, to determine the validity of these benefits and concerns. The results are overwhelmingly favorable for L32. We found that L32 can express the information contained within a C32 document and that L32 instances can be validated by existing validation tools. Moreover, because L32 is represented as an Extensible Markup Language (XML) schema, it is both human and machine readable, thus lowering the burden on software developers. L32 also enables the use of open source and low cost XML tools. Thus, the L32 allows for easier creation of correct C32 instances, at a lower cost to the institution by greatly reducing the amount of custom software needed.

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

Table of Contents

Robert Dingwell 10/13/09 10:36 AM
Deleted: 1

Robert Dingwell 10/13/09 10:36 AM
Deleted: 1

Robert Dingwell 10/13/09 10:36 AM
Deleted: 1

Robert Dingwell 10/13/09 10:36 AM
Deleted: 1

Robert Dingwell 10/13/09 10:36 AM
Deleted: 1

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

List of Figures

List of Tables

Robert Dingwell 10/13/09 10:36 AM
Deleted: 1

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

This page intentionally left blank.

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

1 Purpose

This document summarizes the testing on L32 (the Lightweight C32 Implementation). The L32 is intended to be a single consistent, machine-interpretable implementation of the Healthcare Information Technology Standards Panel (HITSP)[1] C32 specification.

2 Approach

This test plan outlines a series of experiments that tests the hypothesis below. In particular, we intended that these experiments will provide evidence that the benefits of using L32 stated below are true and that the concerns about the L32 stated below can be mitigated. This document includes the combined test results for Phase I and Phase 2 of the L32 effort.

3 Hypothesis

Use of the L32 (i.e., Lightweight C32 Implementation) will make it easier for software developers to generate, validate and consume C32 instances.

4 Benefits of Using L32

B1. Because the L32 includes a constrained version of C32 and provides an XML schema to validate instances against, it is easier to produce valid and correct C32 instances than using the current C32 specification.

B2. Because L32 includes an XML schema that encodes a constrained version of the C32 specification, it is possible to use free and low cost XML tools and to write simpler code to produce C32 instances than using the C32 specification.

B3. Because the C32 specification is more complex and subject to interpretation than the L32, using the L32 will make it easier to generate correct code for consumption of C32 instances.

B4. Using L32 will make it easier for software developers to use free and low cost XML tools to consume C32 instances.

The benefits may be summarized in a quadrant as shown in Table 1.

Table 1: L32 Benefits Summary

	Correctness	Cost
Production	Benefit 1	Benefit 2
Consumption	Benefit 3	Benefit 4

5 Possible Concerns with Using L32

While we explicitly test the above hypothesis, we also wish to show that the following concerns can be mitigated:

C1. L32 is not expressive enough.

C2. L32 is not an approved, released version of C32.

C3. L32 may not capture the subset of data we claim it will capture.

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

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

C4. The adoption of the L32 will require more work than going forward with current C32 version.
C5. An instance created from the L32 will not get Laika[2] or NIST[3] certification.
C6. Using L32 will make existing C32 implementations backwards incompatible.
C7. L32 is not needed because the C32 specification with Schematron Validation[4] is sufficient.

6 Terminology

The following is a description of the terminology used in this document when referring to the L32 and its parts.

C32 - The human readable specification that describes how to create a C32 document.

C32 Instance - A C32 conformant instance can be validated via NIST or Laika tools. A C32 instance is not necessarily conformant to the L32 XML Schema.

CDA Schema – The HL7[5] Clinical Document Architecture schema is an ANSI-approved standard to represent clinical events. The CDA is relied upon heavily by the C32 specification in the following manner. The CDA provides the most expressive possible XML representation, while the C32 adds additional requirements and restrictions.

L32 - A "Lightweight C32 Implementation" or W3C XML Schema implementation of the C32 specification.

L32 Instance - A C32 conformant instance, sometimes referred to as an "L32 Document." An L32 Instance does not contain xsi:type attributes to identify each content module, and the instance can be validated with the CDA schema, NIST, and Laika. To validate an L32 Instance with the L32 XML Schema, the instance must be translated back to a "Native L32 Instance." If the L32 Instance was created with the L32 XML Schema and associated XML style sheet, it can be easily translated back to a Native L32 Instance.

Native L32 Instance – An L32 conformant instance that includes xsi:type declarations used to make the XML schema easier to understand and better able to capture the C32 specification constraints. A Native L32 Instance can be validated with the L32 XML schema. It is not, however, a valid CDA or C32 document. Therefore, it will not validate with NIST Schematron or Laika until the appropriate XML style sheet is used to remove the xsi:type declarations.

7 Experiments

We conducted five experiments. Some experiments have multiple tests. Listed below are the five experiments and their respective test cases that are described in this document. Table 2 maps the planned experiments to the above identified benefits of using L32 and possible concerns with using L32. The only concern not addressed is Concern 2, L32 is not an approved, released version of C32. This concern is not addressed within this document. While L32 is not a new standard that will require approval, the XML schema will need to be vetted.

[2] http://www.cchit.org/laika
[3] National Institute of Standards and Technology http://www.nist.gov/index.html
[4] http://xreg2.nist.gov/cda-validation/validation.html
[5] Health Level Seven (HL7) creates standards for health care. http://www.hl7.org/

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

Experiment 1: Is L32 Sufficient?

Test 1: Manual Content Check of What Is Included
Test 2: Manual Content Check of What Is Not Included
Test 3: Use Case Data Content Check
Test 4: Existing C32 Instance Content Check

Experiment 2: Can we use a C32 Specification Stand-In?

Test 1: CDA Schema and AutoC32Schema Comparison

Experiment 3: Is L32 easier to use?

Test 1: Auto-generated Code Comparison
Test 2: Coding to API Test

Experiment 4: Are L32 instances NIST and Laika compliant?

Test 1: NIST Compliance Test
Test 2: Laika Compliance Test

Experiment 5: Can COTS tools be used?

Test 1: COTS XML Tool Usage

Table 2: Experimental Outline

Exp.	Name	Test	Name	Benefit Addressed	Concern Addressed
1	Is L32 Sufficient?	1	Manual Content Check of What Is Included	B1	C3
		2	Manual Content Check of What Is Not Included		C1, C6
		3	Use Case Data Content Check	B1	C1, C3
		4	Existing C32 Instance Content Check		C1, C6
2	Can we use a C32 Specification Stand-In?	1	CDA Schema and AutoC32Schema Comparison	N/A	N/A
3	Is L32 easier to use?	1	Auto-generated Code Comparison	B2, B3	C4
		2	Coding to API Test	B2, B4	C4
4	Are L32 instances NIST and Laika compliant?	1	NIST Compliance Test		C5, C6
		2	Laika Compliance Test		C5, C6
5	Can COTS tools be used?	1	COTS XML Tool Usage	B2, B4	C4, C6, C7

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

8 Experiments and Tests

8.1 Experiment 1: Is L32 Sufficient?

Does the L32 sufficiently capture the subset of C32 data that we are attempting to capture?

8.1.1 Test 1: Manual Content Check of What Is Included

Description: Manually check which C32 Specification contents are represented in L32.
Dataset: L32, C32 Specification
Method: Hand draw a mapping between corresponding elements and attributes in L32 and the C32 specification.

Results: The following modules from the C32 were implemented in L32 Phase 1: Person Information, Information Source, Healthcare Provider, Conditions, Medications, and Results. All other modules, except Comments were implemented in Phase 2. Table 3 contains the overall information on mapped and unmapped elements for each module. The exact mapping used is available upon request.

Table 3: Coverage of L32

Module	Number of Objects from C32 in L32	Number Objects in C32 not in L32	% Covered
Person Information	13	0	100%
Information Source	5	3	63%
Healthcare Provider	10	0	100%
Conditions	5	0	100%
Medications	43	0	100%
Results	8	0	100%
Language Spoken	2	0	100%
Support	7	0	100%
Healthcare Provider	10	0	100%
Insurance Provider	25	0	100%
Allergy/Drug Sensitivity	9	0	100%
Pregnancy	2	0	100%
Comment	2	2	0%
Advance Directive	5	0	100%
Immunization	11	0	100%
Vital Sign	8	0	100%
Encounter	6	0	100%
Procedure	2	0	100%
Total	173	5	97%

Interpretation:
Of the eighteen modules from C32, the L32 implements the majority of the fields required by C32. Only the Information Source and Comments modules were not completely implemented. The three entities not represented in Information Source are related to the representation of an external document. Discussions with medical personnel noted that these fields were not considered useful, and were intentionally not implemented in order to decrease the complexity of

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

the L32. Comments were intentionally excluded because there were too many inconsistencies in the C32 specification. The specification itself is unclear as to the use and placement of comments, and no sample documents were found containing them. As such, the L32 does not contain them. Additionally, according to the specification, comments can be added to all elements, and we wished to reduce the complexity of L32 instances. Because they were both ill-specified, and because they would increase complexity, comments were removed. However, all modules still contain the narrative component, and thus all textual comments can be included. For instance, all comments for Clinical Document can be placed in: ClinicalDocument/component/structuredBody/component/section/text.

8.1.2 Test 2: Manual Content Check of What Is Not Included

Description: Manually identify what information in the C32 Specification is not captured by L32.
Dataset: L32, C32 Specification
Method: Rigorous listing of all information in C32 specification not being included in L32

Results: Table 3 above contains the number of elements contained in C32 but not represented in the corresponding L32 module. Only 3% of the fields are missing.

Interpretation:
There are a total of three fields from Information Source, and one entire module, Comments, from the C32 specification that are missing in L32. This means that the majority of the entities required in the C32 specification exist in L32. The three fields from Information Source were determined not useful by domain experts and removed for simplicity. The Comments module from C32 is redundant given the implementation of L32 and therefore not included. Therefore, L32 contains the information and format required by the C32 specification.

8.1.3 Test 3: Use Case Data Content Check

Description: Confirm that all user data in a representative health data exchange use case can be adequately represented in an L32 instance document. Note that the patient in this representative use case is named Sam. Therefore, in this document we refer to the instance as Sam's instance and the data contained in this instance as Sam's data.
Dataset: Sam's data from Health Data Exchange Use Case[6]
Method: Create a L32 instance representing all Use Case data

Results: Sam's C32 instance was represented in an L32 compliant XML instance. Table 4 contains the types of modifications that were needed to convert a Laika generated C32 document into a valid L32.

[6] The representative Medical Data Sharing Use Case is available on MITRE intranet at http://tinyd.mitre.org/9D2 with the supporting C32 XML instance available at http://tinyd.mitre.org/9D3

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

Table 4: Changes to Sam instance to create an L32 valid instance

Change	Number
Removed Element from Laika-C32 document	11
Changed Code in field	10
Spelling or Capitalization Change	16
Added attribute to Sam's data	16
Total	53

Interpretation:
There were several elements required by the C32 that were missing from the Laika generated data. These were added when converting to L32 and shown in the "Added attribute to Sam's data" row. Sixteen changes are simple spelling or capitalization changes. For instance, the insurance provider display name was changed from "Self" to "self." Also, in order to conform more accurately to the C32 standard, ten code fields were changed to reflect the name required by the C32, instead of a shorthand version. For example, the "MaritalStatusCode" was changed to "HL7 MaritalStatusCode". Finally, eleven elements were removed from the original C32 document, such as the CDA template id at the beginning of the document. This is not required by L32. Thus, some changes must be made to create a valid L32 document from a C32 document. However, the changes are minimal and still result in a valid C32 document according to the NIST and Laika validators.

8.1.4 Test 4: Existing C32 Instance Content Check

Description: This test examines sufficiency of L32 by taking existing C32 instances and identifying if any data contained in current C32 instances is not representable in an L32 compliant instance.
Dataset: Sample Current C32 Instance received from the Department of Veterans Affairs (VA)
Method: Take a C32 instance already published, transform it into an L32 version, and then note what fields are no longer representable.

Results: A sample C32 document supplied by the Department of Veterans Affairs was transformed into an L32 version. Table 5 contains the amount of modifications needed to transform a C32 document from the Department of Veterans Affairs into a valid L32 document.

Table 5: Changes to Department of Veterans Affairs' instance to create an L32 valid instance

Change	Number
Removed Element from VA document	5
Changed Code in field	5
Spelling or Capitalization Change	6
Added attribute to VA data	17
Not in VA data so commented out	5
Total	38

Interpretation:
As with Sam's C32 document, some changes did need to be made to the underlying C32 instance information in order to be represented in L32 format. The sum of these modifications is found in Table 5. These changes included removing and adding elements in the original C32 instance as

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

well as making minor spelling changes. For example, we removed all the xsi:type=SXCM_TS declarations in the medications module since they were not required by Laika or needed in the L32. Spelling changes were all minor. Most of the added attributes were replacements of nullFlavor. Because the Department of Veterans Affairs shared a document in progress, many attributes contained nullFlavor as a place holder. Thus, as with the Sam instance some changes needed to be made to this C32 document. However, all changes were minor.

8.2 Experiment 2: Can we use a C32 Specification Stand-In?

Can we use the CDA schema or an XML Schema automatically generated from C32 instances as a "stand-in" for a C32 specification in the experiments since no C32 XML schema exists?

8.2.1 Test 1: CDA Schema and AutoC32Schema Comparison

Description: In our experimentation we wish to compare the C32 Specification to the L32. However, the C32 Specification is an English document and the L32 is an XML Schema. Therefore, the C32 may not be used with XML tools, which makes it difficult to measure and compare tools and representations. This experiment examines what to use as a "stand in" for the C32 specification in these tests. We examined two alternatives: 1) the C32 parent document of the CDA schema or 2) to automatically generate a C32 XML Schema from existing sample C32 instances.
Dataset: CDA Schema, C32 instances
Method: Create the AutoC32Schema from XML instances of C32 via Oxygen[7] from six different instance documents produced by five separate vendors at the IHE Connectathon 2009. Load the CDA schema and AutoC32Schema into the Harmony[8] schema comparison tool. Determine mappings to show where the overlaps and non-overlaps exist.

Results: For this task, we utilized an open source, free tool for schema mapping, Harmony. Harmony can perform matches on many different levels. The most basic is a simple type match. For instance, if there is a type, "POCD_MT000040.ClinicalDocument" in both documents, no matter where in the schema, and regardless of hierarchy in the document, Harmony will match them. A more complex matching also includes hierarchy. For instance, consider that "POCD_MT000040.ClinicalDocument" contains a type "POCD_MT000040.Participant1". This relationship is called parent-child. Harmony has the ability to match types while also considering relationships, called structural matching. Thus, if there exists two "POCD_MT000040.Participant1"s in one document, when matching to the second document Harmony will use the parent-child relationship to determine the best mapping.

The CDA schema is heavily recursive. In order to determine mappings between the CDA schema and the AutoC32Schema, every recursion must be instantiated. While Harmony was designed to work over large schemas, and has successfully mapped schemas many orders of magnitude larger than the CDA, the recursive nature of the CDA schema made it impossible for Harmony to efficiently process it. Because of the recursive complexity of the CDA, in which types are constantly reused in grandparent-parent-child relationships, the structural matching capabilities of Harmony could not be used. Therefore, we performed only type matching. The AutoC32Schema contained all of the types found in the CDA.

[7] www.oxygenxml.com/
[8] http://www.mitre.org/work/tech_papers/tech_papers_09/08_0260/

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

Interpretation:
Because the CDA schema is large and unwieldy, a simpler, but still representative substitute is needed. This substitute, the AutoC32Schema, was generated from a small set of actual C32 instances. It contains all of the types found in the CDA, without the full recursive possibilities, since the XML documents used to create the schema did not utilize the recursive properties of the CDA. Thus, we feel that the AutoC32Schema, while a highly simplified version of the C32 specification, may be used as a stand-in for the CDA whenever the CDA cannot be used. The type matching proved that the AutoC32Schema contains the complete set of types from the CDA, but does not contain the same structural complexity.

8.3 Experiment 3: Is L32 easier to use?

Does the use of L32 require more or less effort for a software developer to use than the C32 Specification?

8.3.1 Test 1: Auto-generated Code Comparison

Description: Many software development tools are available to automatically generate Java code stubs given an XML schema. The goal of this experiment is to take the XML schemas, automatically generate Java code, and then use standard software metrics to measure the resulting code complexity.
Dataset: L32, CDA schema, AutoC32Schema
Method: Generate code via JAXB[9]. Measure number of classes needed for CDA, AutoC32Schema and L32. Measure the resulting code with JHawk[10] to determine Hallstead Volume, McCabe's Cyclomatic Complexity, Maintainability Index, and general size counts of classes, methods and statements.

Results: As in Experiment 2, Test 1, it was impossible to use the CDA XML schema with the out of the box tools. The JAXB code creator would not create java classes from the CDA XML schema. As such, this experiment was performed only upon L32, and a schema reverse engineered from a few sample C32 instances supplied by vendors. Table 6 contains the high-level code-measuring metrics.

Table 6: Code Complexity Results

Name	No. Classes	No. Methods	No. State-ments	Total Cyclomatic Complexity	Ave. Cyclomatic Complexity	Halstead Effort	Maintain-ability Index	Maintain-ability Index No Comment
L32	265	2029	6636	2376	1.17	93659.84	110.68	140.06
C32 Auto Schema	103	810	2675	843	1.04	14972.54	99.09	143.22

[9] http://java.sun.com/developer/technicalArticles/WebServices/jaxb/
[10] http://www.virtualmachinery.com/jhawkprod.htm

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

No. Classes	Number of classes automatically created to represent the schema in Java
No. Methods	The total number of methods in all of the created classes
No. Statements	The total number of statements in all of the created code
Total Cyclomatic Complexity	(McCabe's) The number of possible paths through code
Ave Cyclomatic Complexity	Average number of paths per method
Halstead Effort	The amount of work to re-write a piece of code
Maintainability Index (MI)	(University of Idaho) The ease of maintaining code. MI< 65 is hard to maintain 65 < MI < 85 is moderate to maintain 85 < MI is easy to maintain
MI No Comments	Same as MI but does not use comments in calculation

Interpretation:
As in Experiment 2, Test 1, it was impossible to use the CDA schema with the out of the box tools. The JAXB code creator would not create java classes from it. As such, this experiment was performed only upon L32 and the AutoC32Schema. There are two main points that should be taken away from the numbers in Table 6. First, the numbers are higher for all measurements for L32, except for the Maintainability Index. Barring the Maintainability Index, the lower the number, the better. For instance, Cyclomatic Complexity measures the number of paths through a piece of code. Thus, the lower the number, the cleaner and more easily readable the code is. Thus, it would appear that L32 does worse than the AutoC32Schema across the board, except for maintainability. However, this is an incorrect analysis. Because AutoC32Schema was created by reverse engineering a schema from five sample C32 instances, the AutoC32Schema contains only a simple representation, and does not represent the full complexity allowed by the C32 specification or CDA schema. Thus, the correct interpretation should be that L32 is within an order of magnitude from the simple and constrained AutoC32Schema for all metrics. Moreover, while L32 is slightly more complicated than the AutoC32Schema, software only has to be developed for L32 once instead of every time a new instance is seen.

8.3.2 Test 2: Coding to API Test

Description: The ability to produce correct and valid XML instances is imperative for the ability of an electronic system to function. The difficulty of producing correct and valid XML instances also affects software lifecycle costs. Comprehension of the specification or schema affects the ability to produce correct instances. The ability to write code and the length of time for code development is directly dependent upon the specification utilized. This test compares the relative difficulty of writing software to produce a conformant XML Clinical Data document using these different approaches.
Dataset: The schemas and Java APIs generated via JAXB for L32, AutoC32Schema and hData. HData[11] is another MITRE project focused on creating an easily implementable standard for exchanging electronic health records. The data for a test Laika patient in YAML and C32 formats. YAML is a data serialization standard that can be used across programming languages. In this case, it represented an institution's internal data model, containing the information that must be transformed into a C32 document. The C32 version of the information was included to allow the developer to check his transformations against a "gold standard." If his transformation matched that of the given C32, then his code was correct.

[11] http://www.projecthdata.org

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

Method: A skilled software developer was given the schemas and APIs for the three different formats: L32, AutoC32Schema , and hData. This developer was not involved in the development of either L32 or hData, and was unaware which APIs were new vs. pre-existing. He was also given the Laika test data for a sample patient in YAML format. This test data represented the base institution's internal data format. The software developer was asked to implement software to publish the given test data using the three different approaches (i.e., using the L32 API, C32 API and hData API). The software developer timed himself for the tasks shown in Figure 1. We took the developed software and measured its complexity using the same metrics as in Experiment 3, Test 1.

Results: See Figure 1 for the time it took the developer to perform specific tasks, and Table 7 for the code metrics for the produced code. Additionally, Figure 2 contains the verbatim assessment from the test developer who was not associated with the L32 or hData projects in any way.

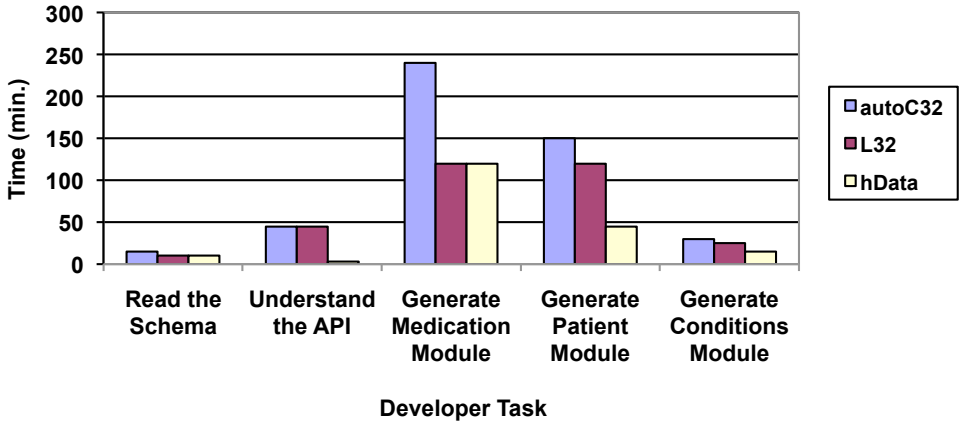

Figure 1: Time to Generate Code Using L32 vs. C32 in minutes

Table 7: Code Complexity Results – Using the Schemas

Name	No. Classes	No. Methods	No. State-ments	Total Cyclomatic Complexity	Ave. Cyclomatic Complexity	Halstead Effort	Maintain-ability Index	Maintain-ability Index No Comment
L32	1	5	108	6	1.20	15195.94	76.34	94.21
C32 Auto Sch.	1	6	116	7	1.17	17808.46	116.53	95.92
hData	1	5	73	6	1.20	11567.34	122.33	101.71

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

No. Classes	Number of classes automatically created to represent the schema in Java
No. Methods	The total number of methods in all of the created classes
No. Statements	The total number of statements in all of the created code
Total Cyclomatic Complexity	(McCabe's) The number of possible paths through code
Ave Cyclomatic Complexity	Average number of paths per method
Halstead Effort	The amount of work to re-write a piece of code
Maintainability Index (MI)	(University of Idaho) The ease of maintaining code. MI< 65 is hard to maintain 65 < MI < 85 is moderate to maintain 85 < MI is easy to maintain
MI No Comments	Same as MI but does not use comments in calculation

GOLD	hData – This API was definitely the simplest to match up and interface with. When I completed connecting to this API, I was fairly confident I had the right connections into the API. And I could make these connections rather fast.
SILVER	L32 – This API was much better than C32, but at times I still struggled with matching up the data to the API. I questioned if I was supposed to use template IDs and code system declarations or not. I'm fairly sure I made a fairly decent document, but I think there might be errors within.
DISQUALIFIED	C32 – What can I say about this API. It is obvious that this API is not being used for its original intent. The C32 guide is helpful in pointing out how to store the data, but the fact that the guide knows exactly what I want to match yet the standard does not have simplified fields for doing so indicates that this standard was repurposed for storing this data. It's grueling to go through the guide, the long template IDs seem arbitrary and easy to mess up, and I really question how close to perfection I got.

Figure 2: Developer Comparison of Technologies (verbatim)

Interpretation:
The time numbers in Figure 1 show that L32 is easier to write code for than C32, but hData is the easiest. L32 is easier than C32 due to the reduced flexibility of information allowed in L32 instances, as well as the existence of a written, as opposed to auto-generated, schema. The developer performed the tasks in the order they are shown in Figure 1. In other words, he generated the Medications Module for C32, L32 and hData before generating the Patient module for all of them. Thus, over time, the developer does get faster for all three technologies. However, even as his confidence and speed increase by the last task, programming to L32 is still faster than C32. The new hData specification yields even better developer performance.

Looking at Table 7, it becomes apparent that not only does the developer understand the schema and code more quickly for L32 than C32, but the code produced is of a higher quality and is less complicated. The Halstead Effort and McCabe's Cyclomatic Complexity are lower for the L32 code when compared with C32. With respect to the Maintainability Index, the higher the number, the better. Additionally, because this code was written purely as a one-off test, the

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

developer did not include any comments. As such, the only valid reading is the Maintainability Index with No Comments. While the Maintainability Index is better for the C32 code, the L32 code is very close. Again, the new hData specification is superior overall to both the L32 code and the C32 code.

Finally, Figure 2 contains the actual words, and scores the test developer gave to the autoC32 API, the L32 API and the hData API. Recall that all three APIs were automatically generated using the same package, JAXB. Thus, any differences are a result of the underlying schema and data encoding semantics. The developer really appreciated the simplicity and understandability of hData. In comparison, the codes and naming convention of C32, reflected in L32, were difficult. However, the developer thought the C32 package, without the benefit of the L32 XML schema, was incredibly complex.

8.4 Experiment 4: Are L32 instances NIST and Laika compliant?

Does a set of instances generated via L32 get NIST and Laika compliance?

8.4.1 Test 1: NIST Compliance Test

Description: The NIST validator[12] is a valuable tool for all institutions exchanging medical data electronically. It is essential that all C32 instances can pass the validation step performed by NIST, whether they are generated by L32 or not.
Dataset: Sam's data from the Health Data Sharing Use Case represented in an L32 compliant instance, the Department of Veterans Affairs' C32 instance, and the Department of Veterans Affairs' C32 instance represented in an L32 compliant instance
Method: Run each instance through NIST validation tools.

Results: See Table 8 for the results from NIST. The NIST validator has three major subcomponents that it checks all documents against: CDA, CCD and HITSP/C32v2.1. In order to be validated by NIST, a document must have no errors for all three subcomponents.

Table 8: NIST Validation of the L32 version of a C32 instance

Dataset	Subcomponent	Number Errors	Number Warnings
Sam-L32	CDA	0	0
	CCD	0	2
	HITSP/C32v2.1	0	4
VA-L32	CDA	0	0
	CCD	0	4
	HITSP/C32v2.1	0	9
Original VA-C32	CDA	0	0
	CCD	0	0
	HITSP/C32v2.1	5	0

[12] http://xreg2.nist.gov/cda-validation/index.html

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

Interpretation:
The L32 version of both the Laika generated and Department of Veterans Affairs' instance data can be validated via standard tools such as NIST. While warnings do exist in the L32 documents, they are mostly warnings about the ordering of subsections within the document, and are not critical for using or validating the instances.

8.4.2 Test 2: Laika Compliance Test

Description: This test also checks for compliance to the C32 Specification but using the Laika[13] Electronic Health Record testing tool. It checks for Laika compliance using a C32 instance that was transformed into a L32 compliant instance.
Dataset: Sam's C32 Instance
Method: Take Sam's C32 instance, certify that it is Laika compliant, and transform it into an L32 version. Run the L32 version through Laika to check for validation to C32.

Results: Table 9 contains the results of the Laika Compliance Test for Sam's data.

Table 9: Laika Validation of the L32 version of a C32 instance

Dataset	Subcomponent	Results
Sam-L32	Schema Validation	Pass
	Schematron Rules	Pass
	UMLS code tests	Pass
	Content Check	35 Errors
Original Sam-C32	Schema Validation	Pass
	Schematron Rules	Pass
	UMLS code tests	Pass
	Content Check	17 Errors

Interpretation:
As seen in Table 9, the L32 version of Sam's data passes the schema validation, the Schematron rules, and the Unified Medical Language System (UMLS) code tests. Only errors in the content check occur. Laika content check errors are merely warnings that expected content was not found, but if all other tests pass, then the document passes Laika validation and is marked for CCHIT auditor review. Of the errors that occur in Sam's document, all are content errors, and many of the content errors for the L32 instance are identical to the errors for the Laika produced C32. For seventeen of the errors, Laika expected "nil" and instead got a value; the L32 actually encodes more data than expected. Additionally, some of the "errors" are a result of L32 using valid OID formats, when Laika does not. Given that the document passes schema validation, Schematron rules and UMLS code tests, the Sam L32 passes Laika validation.

8.5 Experiment 5: Can COTS tools be used?

Is it easier to use commercial off the shelf (COTS) tools with the C32 specification or L32?

[13] http://projectlaika.org

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

8.5.1 Test 1: COTS XML Tool Usage

Description: The ability to use COTS tools means that development time will be reduced, and fewer bugs will be introduced into the system due to custom code. This test examines the COTS tools used within this project and their performance upon L32 and the CDA schema.
Dataset: C32 specification, CDA schema, L32.
Method: For every tool utilized by the MITRE team to manipulate the CDA schema, C32 specification or L32, note whether it was usable and identify any errors that occurred. Note that the C32 specification is a document and therefore unusable by XML tools. Thus, we substitute use of the CDA schema since it is the parent schema utilized by the C32 specification.

Results: See Table 10 for the results.

Table 10: XML Tool Usage Summary

Tool	C32 Specification	CDA	L32
Oxygen - XML Editor	Unusable	OK	OK
JAXB (Java Architecture for XML Binding) – Tool to generate Java classes from a given XML schema	Unusable	Unusable. Required 14 person-hours and was still unusable, with cryptic error messages. Several attempts were made to massage the schema, from flattening it, to removing all Schematron references. None were successful.	OK
Harmony – MITRE developed, open source schema matching tool	Unusable	Minimally usable. Required 8 person-hours to successfully load the CDA and match using types only. Was still unable to show recursive CDA structure or match based on structure.	OK
LiquidXML – XML Editor	Unusable	OK	OK
OpenII – MITRE developed, open source schema integration tool	Unusable	Required massaging.	OK

Interpretation:
Of the sampling of tools required to deal with XML and schemas, the majority were either painful to use with the CDA schema or did not work at all despite large numbers of hours debugging the tool and massaging the CDA schema. As such, out of the box tools do not function as expected or save development time when used with the CDA. Of the tools tried, only Oxygen was completely successful at working with the CDA.

8.5.2 COTS XML Tool Usage Discussion

As seen in Experiment 5, Test 1, having an XML schema allows the use of standard XML tools. Because the use of commercial and open source XML tools can significantly decrease the amount of custom software needed to produce, validate, and consume a data instance, this topic warrants further discussion.

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

Table 11 is a comparison of the steps needed to produce an XML instance in conformance with the C32 Specification starting with the C32 Specification versus using an XML schema for the C32 (i.e., the L32). The table compares both the difficulty levels for a software developer as well as the XML tools available for use. While the difficulty levels noted are admittedly subjective, clearly using an XML schema allows the use of many more standard XML tools. Use of standard XML tools means much less custom software needs to be written. Less custom software means lower development and maintenance costs.

A similar comparison could be made for writing software to consume a C32 instance as well. However, with C32 instance consumption there may be additional costs. Relying upon human interpretation of the C32 Specification document may result in different interpretations of this specification. This can result in differing implementations. Different implementations mean a failure of interoperability. An XML schema will constrain the XML instance so that its format and content are predictable, thereby allowing standard software, generated by XML tools, to be used.

Table 11: Steps, Developer Effort and Tools to produce C32 and L32 instances

Produce C32 Instance		Difficulty for Software Developers		XML Tools Available	
Step	Description	With C32 Specification	With L32 Schema	With C32 Specification	With L32 Schema
1	Understand C32 instance structure and content	Hard: Read 100s of pages across three documents	Moderate to Easy: Understand XML schema	No	XML Editors
2	Create C32 instance interface software	Hard: Custom Code	Easy: XML schema to API tools	No	API Generation Tool
3	Create software to map C32 to local institution data model	Moderate: Custom Code	Moderate: Custom Code	No	XML Tools Emerging
4	Validate XML instance	Moderate: Use supplied validator after developed	Easy: Use COTS & supplied validator	Schematron Rules	XML Editors & Schematron Rules

It is also interesting to note the differences in using Schematron rules versus standard XML editors for validating XML instances. While current methods rely upon Schematron to ensure the quality of the documents being produced and consumed, this is a false security. Schematron functions by firing rules in a particular order and then checking constraints when those rules are satisfied. For example, consider the requirement that all red cars have grey interiors. First Schematron would check the color of the car. If the color was listed as "red", then the rule fires and content of the interior is checked. If the content matches "grey", then the car passes inspection. However, if the car color is listed as "brick red", then the "red" rule never fires, the interior is never checked, and any color interior will pass inspection. Utilizing a schema, such as L32, adds an additional safety check to all instances. Because an XML schema constrains the form and contents of a document, and can be automatically validated using any XML display

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

tool, even a web browser, there is an additional built in surety that the instance conforms to the required regulations.

Robert Dingwell 10/13/09 10:36 AM
Deleted: 9/28/2009 9:30:00 PM

9 Conclusions

L32 was evaluated in terms of completeness, correctness and cost savings. These tests on L32 show that:

1. The CDA is complex and it is difficult to use COTS tools with it. In contrast, the L32 works well with available XML tools.
2. The L32 encodes a valid C32 instance. This instance can represent all data in a standard C32 document except for four fields which were removed for simplicity.
3. L32 is easier for software developers to use than using the C32 specification.

Therefore, our results indicate that the L32 XML schema is easier to use for software developers than the CDA schema or the C32 Specification. By this we mean that it is easier to generate, validate, and consume valid C32 instances using L32 (the Lightweight C32 XML Schema) than the C32 Specification. Our results also indicate that the L32 sufficiently represents C32 Continuity of Care content for the given test instances.

Additionally, the lifecycle of an L32 instance vs. a C32 instance through creation, validation and ingestion by an external system requires less work from the underlying institutions producing and consuming it. Moreover, utilizing the L32 schema allows developers to write better code more quickly through the use of free or low cost XML tools. Table 12 shows a summary of this information.

Table 12: Overview of Benefits of L32 vs. C32 specification

	Correctness	Cost
Production	L32 produces valid C32 instance documents L32 sufficiently represents C32 content	Developers can write code to produce L32 instances faster than by using C32 Developers can write higher quality code with L32 than by using C32
Consumption	With use of an XML schema, L32 instances are consistent and can be consumed from multiple sources	L32 has been successfully used by all COTS tools examined, which indicates that its adoption will require much less custom code