Arguing With the Machine
Analysis of Auto-Generated Code

Jacob Cox, TASC

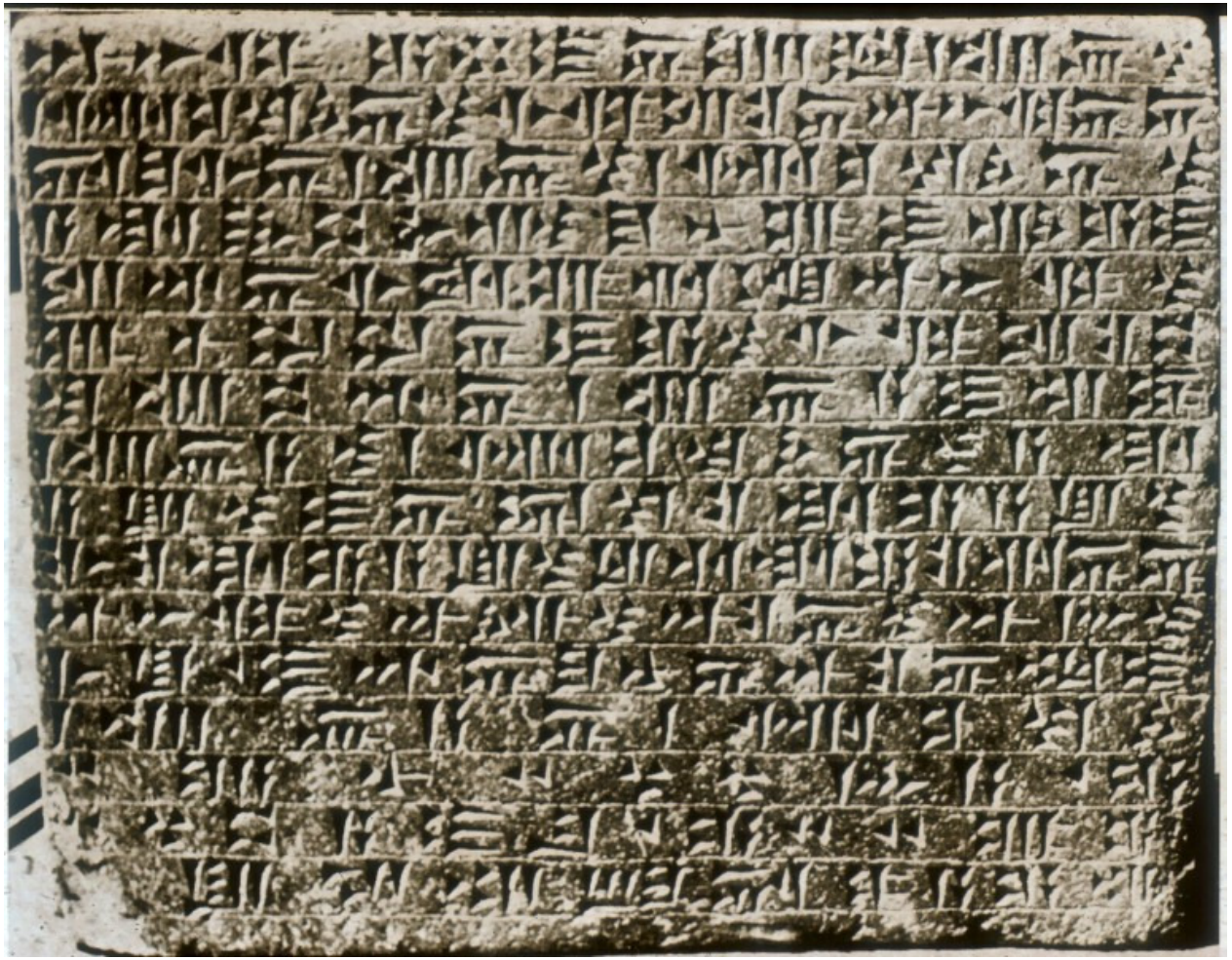
Jacob.t.cox@ivv.nasa.gov

Agenda

- Types of code verification
- General discussion of code generation
- Some experiences

The Simple Life

- In the beginning **people** wrote FSW in **C**

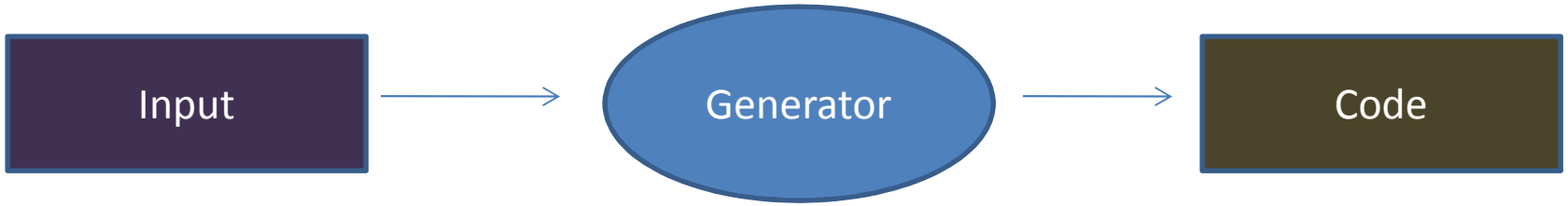


Code Analysis

- Static code analysis (Syntactic)
  - Uses industrial analyzers
  - Produces large numbers of false positives
  - Warnings need checked by hand
- Implementation verification (Semantic)
  - Tracing requirements and design elements to the software implementation
  - Manual process

Code Generators

- Take input and produce source code
- Input can be:
  - UML Models & Diagrams
  - Manual Input (drawing Object diagrams)
  - Text Files (XML)
  - Excel
  - Etc.



IV&V Approaches

- Should we analyze the input instead of the output?
  - This is the way we treat compilers
- Can you trust the generator?
  - Is the generator a COTS product?
  - Has IV&V been performed on it?
- Can you identify the generated source from the manually coded source?
- Is the generated code human readable?

Matlab/Simulink

- The code had headers stating not to edit
- Warning found in these files were ignored
  - The generator was trusted
  - The generated code was hard
    - Lack of comments
    - Difficult algorithms
    - Variables & function names that were machine created

Dictionaries

- A common place to store information used by everyone.
  - Commands
  - Telemetry
  - Faults
- The data is stored in XML files

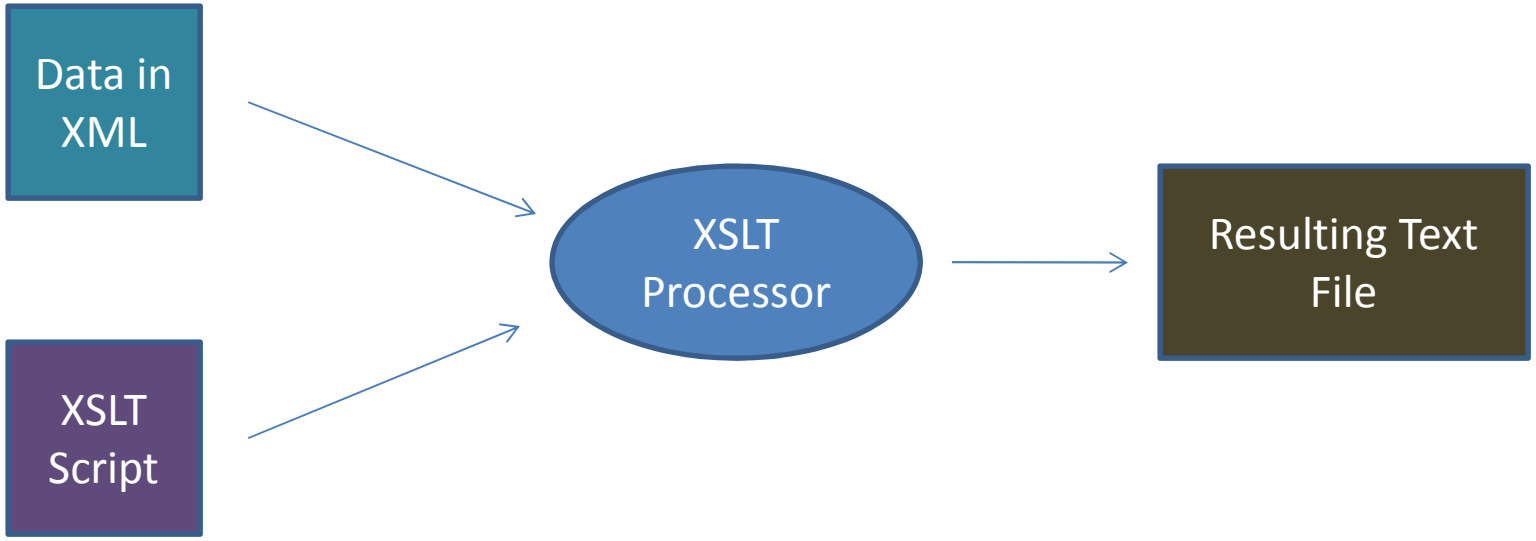
XML

- Text file
- Hierarchical data store
- Allows defining schemas
- Human readable

```
<Dictionary>
  <Command>
    <Name>AddStarToCatalog<Name/>
    <Params/>
    <Param>
      <Name>RightAscentionHour<Name/>
      <Type>UnsignedByte<Type/>
      <MinVal>0<MinVal/>
      <MaxVal>23<MaxVal/>
    <Param/>
  …
  <Command/>
…
<Dictionary/>
```

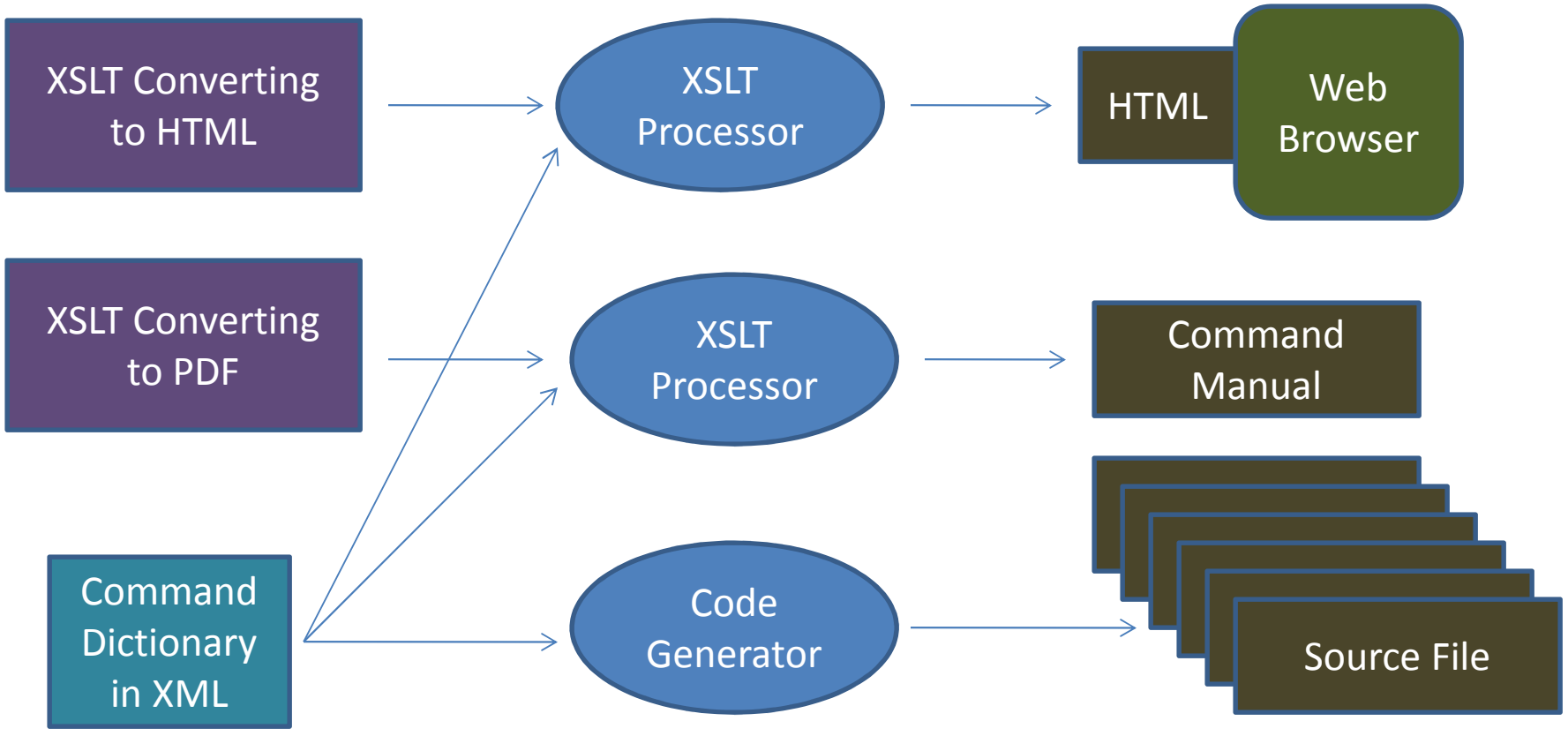
XSLT

- Is an XML transformation language
- Is XML
- Is naturally recursive



Using the Dictionaries

All products are consistent since they come from the same source



Results of the process

- Clearly readable code
- Stubs to place processing of Command Handlers
- Very efficient in code production
- Very in-efficient with respect to SLOC

How Did the IV&V Project React

- Black box approach on validating the generators
  - Compared input XML patterns to resulting code
- Static (syntactic) code analysis on the resulting code
- Traced appropriate requirements to the resulting code (semantic analysis)

Klocwork Raw Results

Warnings from Klocwork are sorted by Autocode generator, or Handcoded

When warnings are put into a pivot table, the grouping of error types by Autocoder are apparent.

Warning Type					HandCoded					Total
ABR					462					462
ABV.STACK					12					12
ASSIGCOND.GEN					2					2
FUNCRET.GEN					4	3				7
F_CYCLE	2				11	2		5		20
F_DEF_IN_HEADER_DECL					148					148
F_DEF_IN_HEADER_EX					183			1		184
F_DUPL_HEADER					1					1
F_MISS_DECL	612		80		1242	21	14	343		2312
F_MULTI_KIND	65	11	19	761	1926					2804
F_ONLY_DECL	139				97					236
NC_CONTEXT	97	11	11		337	1		31		488
NC_EXTRA	153	5	4	4	427	1	9	111		734
NCONSISTENT.LABEL					4					4
PAR					13					13
NPD.CHECK.CALL.MIGHT					14					14
NPD.CHECK.MIGHT					269		61			330
NPD.CHECK.MUST					1113			34		1147
NPD.FUNC.MIGHT					1					1
NPD.FUNC.MUST					12					12
NPD.GEN.MIGHT					1					1
NPD.GEN.MUST					2					2
PRECISION.LOSS					504		1			505
RETVOID.GEN					2					2
RNPD.CALL					10					10
RNPD.DEREF					13					13
SV.FMT_STR.BAD_SCAN_FORMAT					1					1
SV.FMTSTR.GENERIC					3					3
SV.INCORRECT_RESOURCE_HANDLING					1					1
SV.STR_PAR.UNDESIRED_STRING_PARAMETER					1					1
SV.STRBO.BOUND_COPY					108			68		176
SV.STRBO.BOUND_SPRINTF					4					4
SV.STRBO.UNBOUND_COPY					2					2
SV.STRBO.UNBOUND_SPRINTF					2					2
SV.TAINTED.INDEX_ACCESS					1					1
SV.TAINTED.LOOP_BOUND					1					1
SV.TOCTOU.FILE_ACCESS								17		17
UNINIT.CTOR.MUST					1					1
UNINIT.STACK.MIGHT					13					13
UNINIT.STACK.MUST					260			34		294
UNREACH.BREAK					10		96			106
UNREACH.GEN	442				235		55	26	63	822
UNREACH.RETURN					1					1
UNREACH.RETURNS					40					40
VA_UNUSED.GEN	569				160	19	20	33		801
VA_UNUSED.INIT					251			443		694
VA_UNUSED.INITCONST	50	16			227		46	111		450
Total	2156	63	123	766	9115	47	302	1342	63	13979

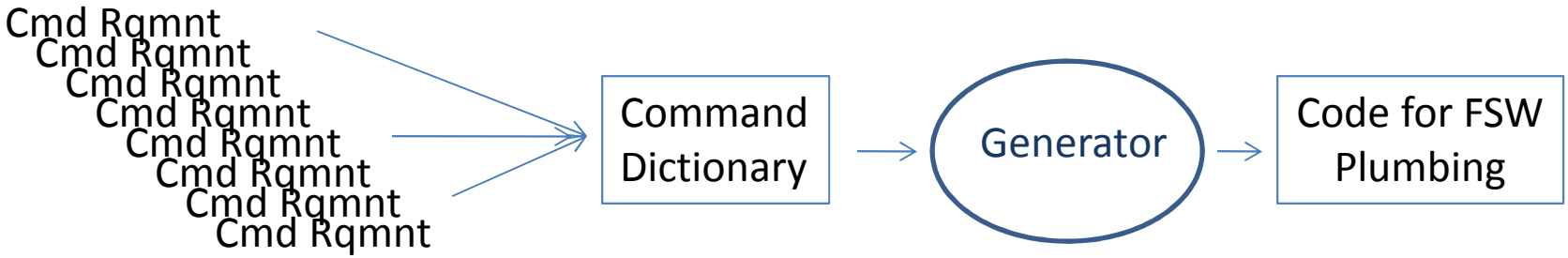
Examples

- Checking parameters to insure they are not greater than maximum value but not checking they were less then minimum.
- Pointer increments at the end of repeated blocks of code.

Generally speaking, static code analysis on auto-generated code did not produce issues and the warnings fell into easily identified patterns.

Code, Dictionaries and Requirements

Project A

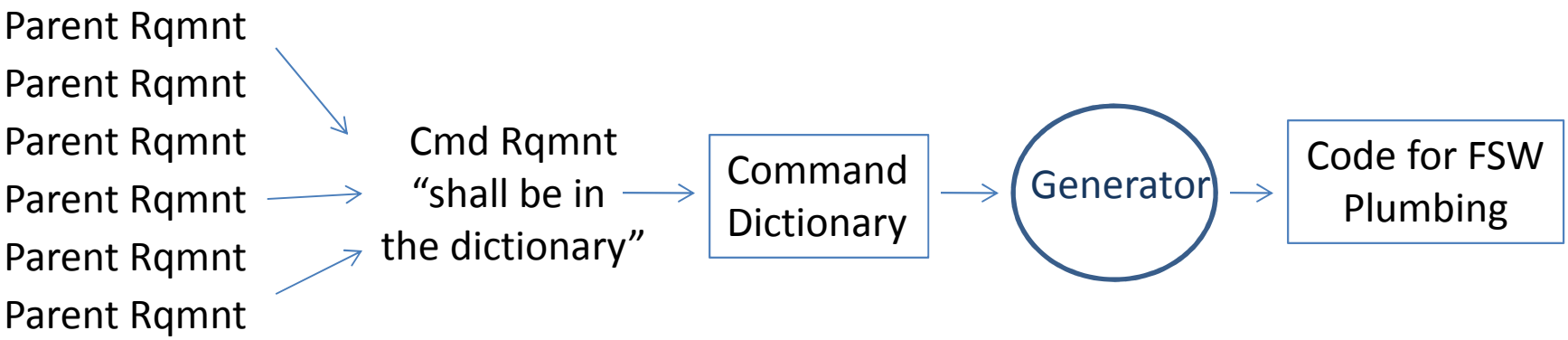


There is a transitive relationship between the requirements, the dictionary entries and the code implementation. Checking the requirement in code also verified the dictionary (at least the parts that generated code).

Note: Semantic analysis of a command requirement found that mode restrictions Had disappeared from the dictionary.

Code, Dictionaries and Requirements Ctd.

Project B



There is no relationship between the requirements and the dictionary entries or code implementation. A solution could be a reference to the parent requirement in the dictionary entry.

Model Generated Code

- Fully model generated code which generates all the code using
  - Class diagrams
  - Sequence diagrams
  - Activity diagrams, etc.
- Partially generated code where the generator creates headers and stubs that are filled in by hand
  - Class diagrams

Analysis should be on the Inputs to the generator

Really just well structured Hand written code requiring the same analysis as regular Hand written code.

Model Generated Code, Ctd.

- The Hybrid where the diagrams generate the code BUT there is hand written code embedded in the diagrams that is incorporated into the generated code.
  - Semantic Analysis can be performed on the snippets
  - Does IV&V perform static analysis on the handwritten parts?
  - Would it be performed on the snippets only and how to identify them in the generated output?

Questions?