Test verification and anomaly detection through configurable telemetry scanning

Alan S. Mazer
Instrument Flight Software Group
Instruments Division
Jet Propulsion Laboratory, California Institute of Technology

E-mail: alan@judy.jpl.nasa.gov

• Despite hundreds of hours of testing (or more), flight software still launches with undiscovered errors

• By launch, software has passed through many hands
  o Developers
  o Peer reviewers
  o Integration and test (I&T)
  o ATLO pre-launch testing

• Sometimes, if not often, anomalous behavior is captured in test data unnoticed
  o GALEX
  o MICAS camera (Deep Space 1)
Why aren’t problems found during development?

- **Time constraints**
  - Sometimes we barely have enough time to write the software

- **Software developers aren’t suited to testing**
  - Testing is tedious
  - Engineers are limited by their “creator” perspective

- **Independent testing is a thankless job**
  - Learning curve costs time and money
  - Find problems and people are upset; don’t find problems and people wonder why you’re paid
Why aren’t problems found during instrument I&T?

- Time constraints
  - System I&T is usually pressed by schedule

- Errors may present subtly
  - Small telemetry oddity may reflect larger problem

- Cost constraints
  - Expertise to recognize software errors is not always present

- Trust
  - Test teams rely on developer testing, prioritizing software checkout below other pressing issues
  - Software problems can always be fixed “later”
• “Human factors”
  o People get tired and make mistakes
  o Testers may not want to question what they’re seeing
  o People following procedures focus on following the steps rather than thinking about what they’re seeing

• Late changes
  o Without regression tests, late changes introduce risk as new requirements are implemented by developers who have already moved to other projects and forgotten the code
What can we do about this?

- Phase B/C (pre-I&T)
  - Define scriptable tests to exercise code
  - Provide visibility into software operation through (perhaps optional) telemetry
  - Verify telemetry to determine whether or not test passed

- Phase D (I&T, ATLO)
  - With system engineering, create validity rules for all telemetry points, capturing expertise and determining which anomalies are reportable
  - Verify all test telemetry against rules
Verifying telemetry is still hard

- Detailed telemetry verification is not well supported by common tools
- One approach to verifying a test is to compare test telemetry to previous runs
  - Simple
  - Works only if telemetry outputs don’t vary from run to run (e.g., due to harmless timing variations)
- Another is to use Unix expect (a selective `diff`) to verify critical outputs
  - Can ignore innocuous variations in telemetry
  - But...
    - All telemetry must be converted to ASCII
    - Repetitive goals are tedious to set up
    - Doesn’t support all-telemetry checks
• Decided to create a rule-based parser, HKCheck, based on ASCII user-authored configuration files
  o Post-processes binary data streams
  o “Protocol” spec describes packet/message format(s)
  o “Test” spec describes constraints on each telemetry point, and user goals to be satisfied by a particular test

• Supports phase B/C test verification by checking for test goals in telemetry
  o A goal might be an intended error or receipt of a particular command

• Supports phase B/C/D by scanning telemetry and calling out unexpected values

Wrote HKCheck to parse telemetry
• “Protocol” spec
  o Supports heterogeneous packet streams, matched to packet definitions at run-time based on packet contents
    • For example, engineering and science packets in a common stream
    • Packets may be variable-length
  o Provides about a dozen built-in data types
    • Integer, floating- and fixed-point values
    • Various time types, with a variety of epochs
    • Several byte orderings
  o Allows user-defined constants and data types, and arrays
  o Display formats are specific to each telemetry point
Each packet def lists a sequence of telemetry points contained in that packet type.

Each telemetry point has a data type (e.g., uint8), a display format (e.g., date, hex), and a name.
User-defined data types allow multiple telemetry points to be grouped as one.
Reduces complexity of packet definitions.
Simplifies output displays (e.g., error description is one line rather than 3).

```
datatype error = {
  uint8: errorID          errorID
  uint8: hex               details[5]
  time4s2ss: date          errorTime
}

datatype downloadCommand = {
  uint16: hex               memoryAddr
  uint16: dec               bytecount
}
```

“Error” data defines structure of single telemetry point for display.
Subpackets

Subpackets group related telemetry items for inclusion across multiple packet definitions.

```
subpacket status = {
    uint16: dec  PktCnt
    uint8: hex   FswVer
    uint8: hex   ScienceVer
    uint8: hex   SensorVer
    uint16: hex  Status
    uint8: hex   Mode
    time4s2ss: dec SCTime
    uint16: hex  CRC
    uint8: dec   Resets
    uint8: dec   TimesMiss
    uint16: dec  CmdsRcvd
    uint16: dec  CmdsExec
    uint16: dec  CmdsRejected
    mwrMessage   LastMsg
    mwrError     LastErr
    uint16: dec  ErrorCount
}
```

“Status” subpacket groups status items which appear in both science and engineering packet formats.
Protocol for GPS/IMU Data Stream

- Set byteorder=msb4thin8

Subpacket header = {
  uint16:hex  syncWord
  uint16:dec  msgID
  uint16:dec  wordCount
  uint16:hex  flags
  uint16:hex  checksum
}

Packet timemarkPacket = {
  subpacket
  fixed<1+20+43>:hms gpsSecs
  fixed<1+17+46>:hms utcSecs
  uint16:none pad[5]
  uint16:dec day
  uint16:dec month
  uint16:dec year
  uint16:hex data[wordCount-15]
}

If (msgID == 3623)

Packet allOthersPacket = {
  subpacket
  fixed<1+20+43>:hms gpsSecs
  uint16:hex data[wordCount-3]
}

Middle-endian byte ordering specified

All packet formats include common header

Format of GPS TimeMark packet

Format of other packets

Protocol for GPS/IMU Data Stream
Real-life Protocols Are Large

- Typical protocols for flight instruments run to hundreds of lines
  - User-defined data types and constants
  - Subpacket definitions
  - Multiple packet definitions
“Test” defines conditions for each telemetry point to be performed on each applicable packet

- Allows each telemetry point to be verified against user-defined conditions and/or conditionally displayed

- Error and display conditions...
  - Use C-like syntax
  - Can reference the current, previous, and last-different values
  - Can reference the age (in packets) of the current value
For this example, want to...
  o Verify packet numbers are sequential
  o Verify that S/C time in each science packet is later than previous S/C time, but not by more than 5 seconds
  o Display the contents of each non-empty dump packet

Nomenclature:
  o $ refers to current value; _$ is last value
  o “template”, “check”, and “show if” are keywords

```
template mytest = {
  PacketNumber  check $ == _$+1
  SpacecraftTime check $ > _$ && $ <= _$+5
  DumpLength    show if $ != 0
  DumpData[0..254] show if DumpLength != 0
}
```
“Test” files may specify sequential goals to be met
  o Can be used to verify that a test completed successfully as reflected in telemetry
  o Goals are simply conditions using same syntax as used for checks
• For this example, want to...
  o Verify that first packet in stream is science packet
  o Verify that we have at least one non-empty dump packet

• Nomenclature:
  o “goal” is a keyword

  goal “First packet is science packet”
    (PacketNumber == 1 && PacketType == NOMINAL)

  goal “Found dump”
    (PacketType == DUMP && DumpLength != 0)
HKCheck takes the protocol and test file(s), along with the binary telemetry input, and generates a report

Reports show
- Rules violated ("check")
- Conditionally-displayed values ("show if")
- Goals met and unmet ("goal")
- Summary notes ("startnote" and "endnote")
In this portion of a run on flight telemetry from Mars Climate Sounder, HKCheck found an odd time increment (nominal is 2-3 seconds)

Nomenclature:
- “start” is a keyword which evaluates true the first time a packet type appears in the stream

SCTim has an error value: 887581376 (was 887581375)

Requirement:
```
start || Resets == _Resets+1 ||
($ >= _$+2 && $ <= _$+3)
```
LastCmd UPLOAD XRAM 0xcee7 138 0x80 0x75 0x2d

LastCmd UPLOAD XRAM 0xdd46 8 0x02 0xc6 0x77

LastCmd UPLOAD XRAM 0xde84 8 0x02 0xc6 0x30

LastCmd EQX 0 250
Met goal: "CRC check"

Met goal: "Pos-error resync #1"
Met goal: "Pos-error resync #2"
Met goal: "Pos-error resync #3"

... Status has an error value: 0x42 (was 0x02)
  Requirement:
    $ == 0x00 || $ == 0x02 || $ == 0x40

Met goal: "Pos-error resync #4"
EOF
All goals met
Failed -- found one or more errors
• Useful for ASCII-fying telemetry through “show” statements as a test record
• Optionally generates spreadsheets as .csv files, or native Excel (with commercial add-on package)
• Enables rapid, repeatable testing during development
• Post-launch telemetry can be scanned...
  o to confirm instrument health
  o postmortem, to look for odd conditions prior to a failure
• Allows expertise to be encoded in rules, reviewed, and carried through the life of the project
• Used for flight software regression testing or telemetry scanning on
  o Mars Climate Sounder (MRO), Diviner (LRO), Microwave Radiometer (Juno), Phoenix MECA, GALEX, and various airborne missions
• Open-source release pending