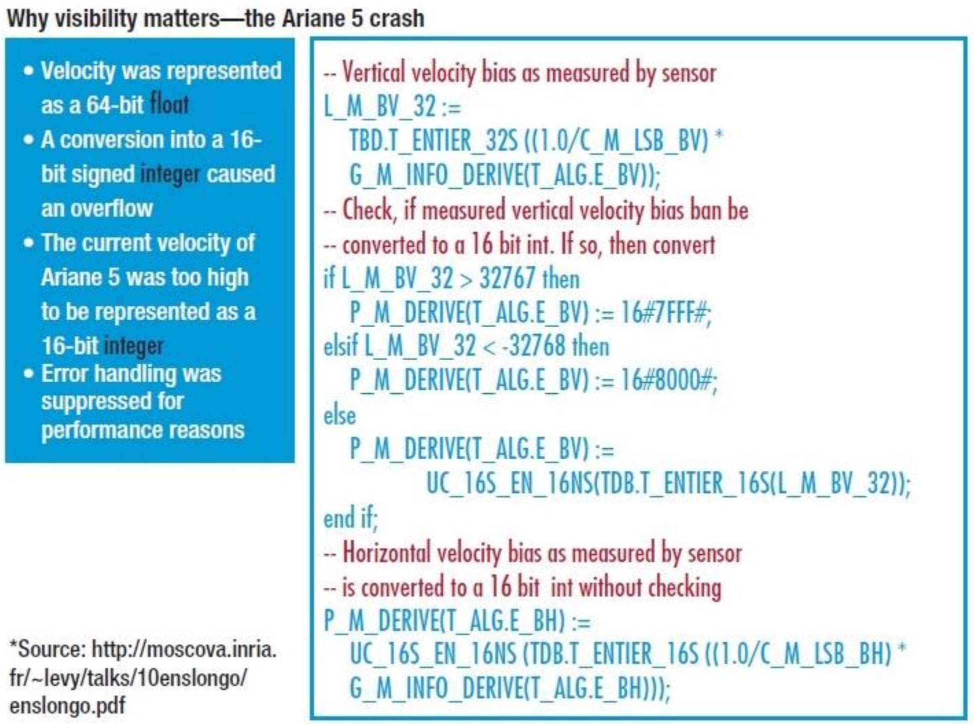
Measuring the Complexity and Adaptation of Transportation CPS

BBN Technologies

The complexity of modern military, aerospace, and other systems and systems of systems has skyrocketed in recent years. These Cyber-Physical Systems (CPSs), including transportation and aeronautical systems, comprise major physical components that can no longer function properly without integrated cyber components (such as engine control systems and communication buses) [1, 2, 3 4]. The complexity of CPSs and their increasing reliance on software control of mechanical systems has contributed to a long list of disasters [5, 6], including ones that have led to catastrophic loss of life or property [7, 8, 9, 10].

Managing the growth in complexity in CPSs while building CPSs that can adapt over a long lifetime remains a challenge for designers, developers and maintainers. Transportation systems have longer lifetimes than usual for software replacement. Frequent software patches and upgrades – necessitated by discovered bugs or vulnerabilities – are needed over vehicle lifetimes that span several years or decades. Furthermore, code (as well as mechanical parts) is often reused and retargeted from one vehicle design to another. While mechanical engineers understand mechanical interactions and software engineers understand software interactions, the interplay between software and mechanical components in CPSs is not as well understood and controlled. A famous example is the Ariane 5 rocket, which reused *inertial reference system* code from the Ariane 4 rocket without accounting for the faster initial acceleration and horizontal velocity buildup of the Ariane 5 rocket, which resulted in an integer overflow (without an exception check that was removed for performance), as shown in Figure 1 [11]. The rocket self-destructed approximately 37 seconds after launch. Nearly two decades have passed since this disaster, and (increasingly more complex) automotive and aeronautical CPSs still continue to be plagued by highly publicized and costly recalls and delays [12, 13, 14, 15, 16].

Despite improved tools for the design and development of the mechanical, electronic, and software components of modern CPSs, we continue to build cyber physical systems of systems without an adequate understanding of the final complexity or how difficult it will be to modify the system for future needs. A key reason for this lack of understanding is that there are few ways to measure the complexity or adaptability of CPSs. As a result, CPSs today are in-



Why visibility matters—the Ariane 5 crash

- Velocity was represented as a 64-bit float
- A conversion into a 16-bit signed integer caused an overflow
- The current velocity of Ariane 5 was too high to be represented as a 16-bit integer
- Error handling was suppressed for performance reasons

*Source: http://moscova.inria.fr/~levy/talks/10enslongo/enslongo.pdf

```
-- Vertical velocity bias as measured by sensor
L_M_BV_32 :=
   TBD.T_ENTIER_32S ((1.0/C_M_LSB_BV) *
   G_M_INFO_DERIVE(T_ALG.E_BV));
-- Check, if measured vertical velocity bias ban be
-- converted to a 16 bit int. If so, then convert
if L_M_BV_32 > 32767 then
   P_M_DERIVE(T_ALG.E_BV) := 16#7FFF#;
elsif L_M_BV_32 < -32768 then
   P_M_DERIVE(T_ALG.E_BV) := 16#8000#;
else
   P_M_DERIVE(T_ALG.E_BV) :=
            UC_16S_EN_16NS(TDB.T_ENTIER_16S(L_M_BV_32));
end if;
-- Horizontal velocity bias as measured by sensor
-- is converted to a 16 bit  int without checking
P_M_DERIVE(T_ALG.E_BH) :=
   UC_16S_EN_16NS (TDB.T_ENTIER_16S ((1.0/C_M_LSB_BH) *
   G_M_INFO_DERIVE(T_ALG.E_BH)));
```

Figure 1. A software error due to code reuse led to physical destruction of the Ariane 5 rocket.

Joseph Loyall, Shane Clark, Partha Pal, Rick Schantz **{jloyall, sclark, ppal, schantz}@bbn.com**

creasingly complex and lack adaptability, negatively impacting the effectiveness, lifecycle cost, and maintainability of the systems. These shortcomings are particularly alarming in the transportation sector where software bugs can lead almost directly to human injury or death.

Some complexity metrics do exist for software only, including McCabe's metrics (e.g., Cyclomatic Complexity), Halstead's metrics (e.g., source lines of code), and coupling and cohesion. Similarly, metrics exist for the physical aspects, such as Size, Weight, and Power (SWAP). However, the community is lacking in complexity-centric and adaptability-centric metrics that encompass and capture the interconnections, dependencies, and combined aspects of the cyber and the physical components throughout the system's lifecycle. The lack of appropriate metrics means there is little support for system designers and developers to compare competing designs, safely reuse cyber components in new physical systems, modify CPSs, and make design tradeoffs in critical areas affecting system success and acceptability. This shortcoming motivates new research into appropriate metrics integrated into design and maintenance environments and processes. We suggest that there are two classes of metrics that require research and development to support the design, development, and maintenance of transportation CPSs, namely *complexity* and *adaptability* metrics.

Complexity metrics – The main purpose of these metrics is to assess the extent to which unforeseen situations can arise in a CPS. The more complex a system is, the more likely that some condition that might arise during its lifecycle has not been thoroughly tested, vetted, controlled, or compensated for. The key for complexity metrics is that they are *comparative within a single CPS instance*. That is, while it is not realistic to hope for a complexity metric that is an absolute judge of the worth of a specific system (e.g., a system might be less complex than another because it is much less feature rich or capable), it is worthwhile to determine whether a change to a CPS or a particular design or implementation choice makes the system more or less complex, and therefore more or less maintainable over its lifecycle.

Important factors that could be incorporated into useful and general complexity metrics include, but are not limited to, the following:

- The degree to which subsystems are interdependent, indicating the likelihood that changes or failures in one can affect others.
- The degree to which the logic and mechanics of a subsystem are well organized, understood, documented, and tested.
- The degree of resource contention between system components, where resources can include the traditional cyber resources of CPU, memory, and bandwidth, but also physical resources such as gross weight, power, space, and thermal dissipation, and lifecycle resources, such as maintenance dollars (e.g., for replacement parts), hours (e.g., time to upgrade, fix, and replace), and manpower (e.g., amount of manual labor involved in maintenance).

Adaptability metrics – The main purpose of these metrics is to assess the ability of a CPS to adapt to future changes. A CPS could be low complexity, but very rigid, requiring complete redesign and redevelopment for each new feature or new version. An adaptable CPS, in contrast, would accommodate upgrades and maintenance without major cost or effort, facilitate reconfiguration of components for reuse across platforms, and reduce obsolescence by enabling individual component upgrades. Likewise, systems that are runtime adaptable handle wider ranges of operating conditions, new or unforeseen interactions and interconnections, failure cases inclusive of malicious intent, extremes of environmental conditions, changes in their patterns of use, and many other aspects of an unknown future. Improved adaptability leads to less downtime, longer lifecycles, more efficient usage, and safer, streamlined upgrades.

These complexity and adaptability metrics need to be incorporated into design, development, and maintenance environments and processes, so that design decisions, implementation tradeoffs, and maintenance choices can be evaluated with respect to their ability to reduce (or at least not increase) the complexity of the system and maintain or improve the adaptability of the system. In order to achieve this vision, the metrics must have the following attributes:

- *Usable* within existing or emerging CPS tools and processes. That is, the metrics should encompass relevant characteristics of CPSs to drive design, development, and maintenance decisions and support integration into the systems and tools used by designers, developers, and maintainers.
- *Observable* and *computable*, i.e., it should be possible to calculate or estimate the metrics for a wide range of systems at various stages of development while the system is being conceived, designed, developed, and maintained.
- *Validatable* on data from historical and current real CPSs, i.e., it should be possible to calculate the metrics using information from prior or current existing systems, such as those referenced above, toward experiments validating a correlation between lower complexity and higher adaptability with longer lifespans, lower lifecycle costs, and fewer documented problems.

The definition, design, and adoption of such metrics is a challenging, high-payoff area of research. The current state of the art for evaluating complexity and adaptability typically focuses on the cyber (i.e., software) aspects or physical aspects, but not both and their interplay. Many existing metrics are easily computable, but overly simplistic and consider only a limited set of system characteristics. Others are more comprehensive, but not computable in general. The depth of this problem is underscored by the fact that there are still ongoing debates over the meaning and scope of complexity and adaptability in cyber-physical systems today. Meaningful debates about which factors have the greatest impact on complexity and adaptability, or which metrics are appropriate in different contexts have yet to begin in earnest.

References

1. Mark V. Arena, Obaid Younossi, et al., Why Has the Cost of Fixed-Wing Aircraft Risen? Report No. MG696, RAND Corporation (2008).
2. Paul G. Kaminski et al., Pre-Milestone A and Early-Phase Systems Engineering, National Research Council (2008).
3. Lee, E.A.; , "Cyber Physical Systems: Design Challenges," Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on , vol., no., pp.363-369, 5-7 May 2008
4. Kurt Rohloff, Partha Pal, Michael Atighetchi, Richard Schantz, Kishor Trivedi and Christos Cassandras. “Approaches to Modeling and Simulation for Dynamic, Distributed Cyber-Physical Systems.” Workshop on Grand Challenges in Modeling, Simulation, and Analysis for Homeland Security (MSAHS-2010), Mar 2010.
5. Glass, R. Software Runaways: Monumental Software Disasters, Prentice Hall PTR, 1997.
6. Neumann, P. “First Six Months of the Forum in Retrospect; Updated Disaster List,” The Risks Digest, 2(1), Feb 1986.
7. Leveson, N, Turner, C. “An Investigation of the Therac-25 Accidents,” *IEEE Computer,* 26(7), Jul 1993, pp. 18-41.
8. US Government Accountability Office, Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia, GAO/IMTEC-92-26, Feb 4, 1992.
9. Gleick, J. “A Bug and a Crash: Sometimes a Bug Is More Than a Nuisance,” The NYTimes Magazine, Dec 1, 1996.
10. ARIANE 5, Flight 501 Failure, Report by the Inquiry Board, 1996, http://www.ima.umn.edu/~arnold/disasters/ariane5rep.html.
11. Buschmann, F., “The Pragmatic Architect – To Boldly Go Where No One Has Gone Before,” *IEEE Software*, 29(1), Jan/Feb 2012, pp. 23-25.
12. Ostrower, J., Pasztor, A., “Dreamliner’s Other Issues Draw Attention,” *Wall Street Journal*, May 20, 2013.
13. Charette, R., “Honda Recalls 936,000 More Vehicles for Electrical and Software Fixes,” *IEEE Spectrum*, Sep 7, 2011.
14. “Tech. View: Cars and Software Bugs,” Babbage Science and Technology Blog, *The Economist*, May 16, 2010.
15. Manjoo, F., “I’m Sorry, Dave, I’m Afraid I Can’t Make a U-Turn, Should We Be Worried That Our Cars Are Controlled By Software?” *Slate,* Feb 16, 2010.
16. Hyde, J., “Key Car Quality Study Ranks Software Bugs As Most Common Complaint, Knocks Ford Again,” http://news.yahoo.com/blogs/motoramic/key-car-quality-study-ranks-software-bugs-most-170243393.html, Jun 20, 2012.

Joseph Loyall, Shane Clark, Partha Pal, Rick Schantz {jloyall, sclark, ppal, schantz}@bbn.com