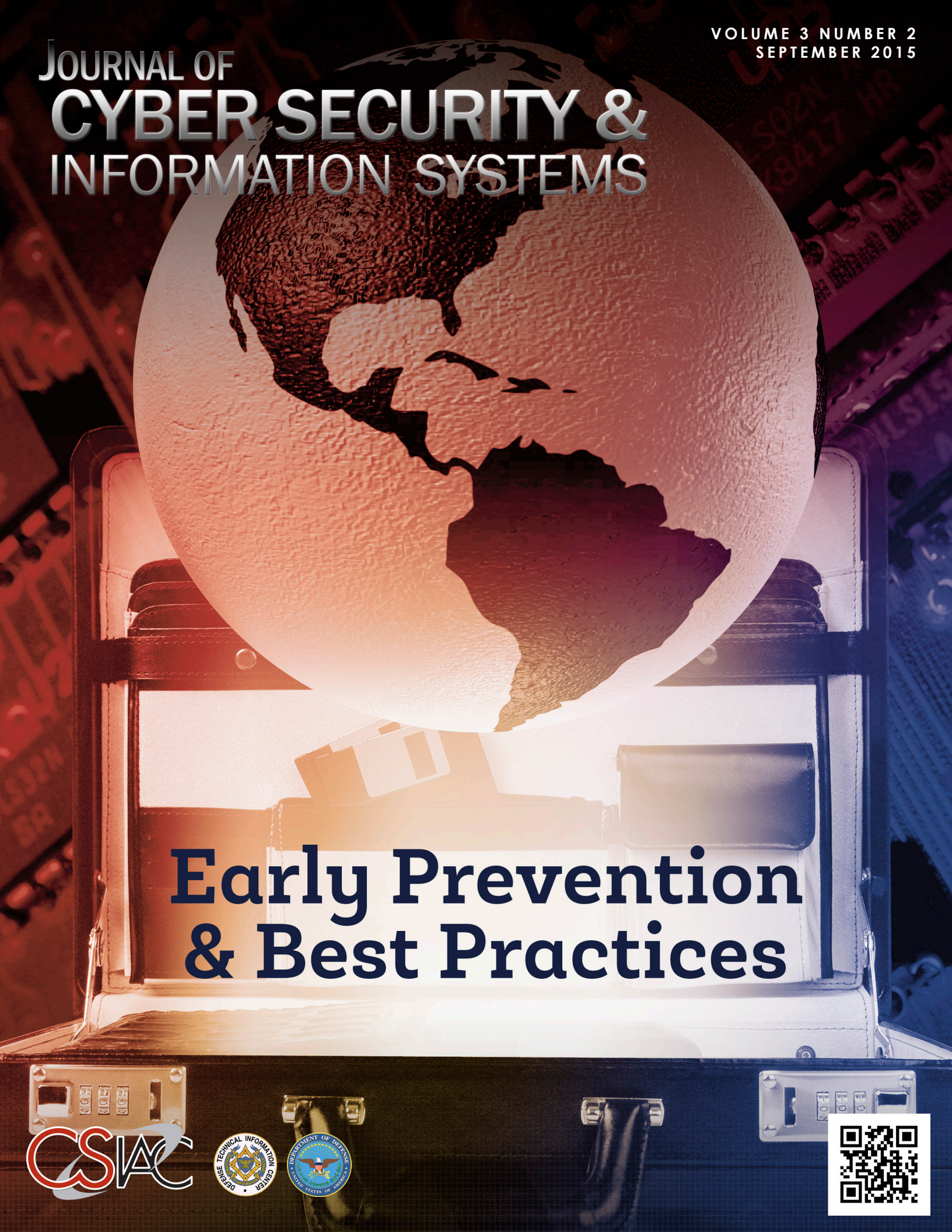
VOLUME 3 NUMBER 2
SEPTEMBER 2015

JOURNAL OF CYBER SECURITY & INFORMATION SYSTEMS



Early Prevention & Best Practices

Increasing Assurance Levels Through Early Verification with Type Safety

By Rick Murphy

Today defense, intelligence and civilian agencies are focused on cyber security challenges. An informal analysis of emerging capabilities in the DARPA Open Catalog indicates that type safety is a common requirement for increased assurance levels in these agencies. Common requirements associated with emerging capabilities are especially relevant to planning and architecture. This article is an abridged version of a longer article that describes how to increase assurance levels with a type safe technical architecture.

Early Verification with a Type-Safe Technical Architecture

This section describes how programs already familiar with the Unified Modeling Language can represent type safety in a technical architecture.

It is claimed that type safety stops programs from going wrong. So how do we stop programs from going wrong in a technical architecture? We include a feature called templates in our UML artifacts to represent the rules enforced by the type checker. UML Superstructure, Auxiliary Constructs (17) describes templates as "A parameterable element is an element that can be exposed as a formal template parameter for a template, or specified as an actual parameter in a binding of a template." Templates allow for parametric polymorphism in UML artifacts. Parametric polymorphism is a feature of type systems that enforce type safety. Templates are known alternatively as generics in mainstream programming languages like C++, Java, C# and Scala. By including templates in the artifacts of our architecture, we represent the rules verified by the type checker.

We continue with four examples of how to represent type safety with UML templates. Examples 1 & 2 provide a static view of type safe structure. Example 1 describes the type safe construction of natural numbers. Example 2 provides an overview of key functional programming abstractions. Examples 3 & 4 provide a dynamic view of type safe operations. Example 3 explains type safe operations required to lift a function into the Maybe Monad. Example 4 explains the safe application of the lifted function to a value. Recall that our purpose is to demonstrate how to represent type safety.

Example 1 - Type Safe Construction of Natural Numbers Using Templates

Figure 1 is a UML Class diagram with Templates. Nat, representing the natural numbers, is an abstract UML class. Because Class extends Classifier, Nat, like Class, is a templateable, or parametrized, element. The additional box outlined in dashed in the top-right corner of Nat class is the template. Nat is parametrized by A, a type parameter. TA is its formal parameter bound to the class A representing the type variable A. The right arrow (>) represents the binding. The sub classes Zero and Suc are data type constructors as indicated by their stereotypes. The data type constructors are also parametrized. Notice that Zero has a private, no-arg constructor restricting access so its parameter will never be used as required by the progress and preservation rules. Zero is also static and final further restricting its use. Suc takes values of type Nat as a parameter in its single arg constructor allowing recursive construction of countably infinite values. Suc too is static and final.

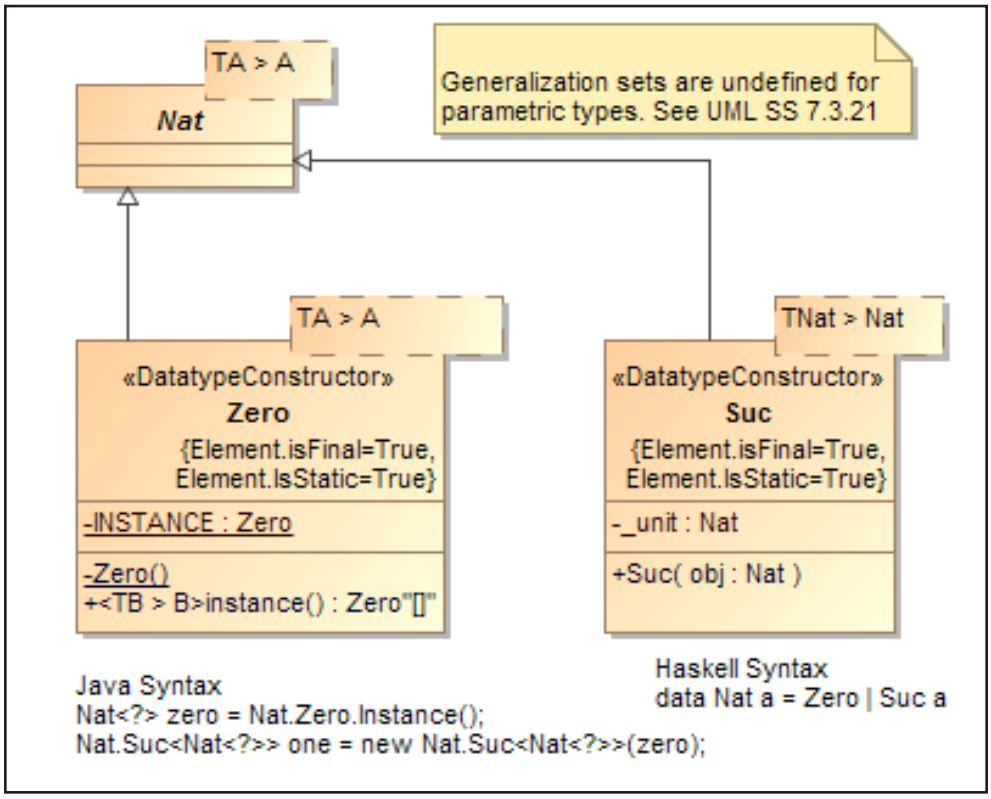


Figure 1 - Type Safe Construction of Natural Numbers Using Templates

This simple class diagram visually represents that the type checker enforces type safety in the construction of natural numbers at compile time. Enforcing type safety at compile time stops the program from going wrong. A UML architect would be expected to provide the verification rules used in a verifiable architecture milestone. The code snippets at the bottom of the diagram illustrate the syntax of two type safe languages: Haskell and Java. The section below on type safety and software verification explains compile-time constraints in source code.

Example 2 - Overview of Functional Programming Abstractions

Figure 2 expands on our first example of type safety with an overview of functional programming abstractions. Type safety is a recurring theme in functional languages. Representing functional abstractions in UML introduces type safety to a broader audience in support technology transition. More defense, intelligence and civilian programs will be familiar with UML than the commutative diagrams of category theory. Reading Figure 2 from top left to bottom right we again see the natural numbers. Nat, Zero and Suc are elided to save space. To the right we see that Nat implements Functor. Functor is an abstraction through which it operates on its own elements using fmap, its only operation. Functor has two parameters A and B. Fmap takes a function, operates on values of its type and returns a function that lifts the values into the type of the Functor. Lifting a value into the type of Functor encapsulates the value in the type. Operations on the lifted value must now satisfy the progress and preservation properties of the Functor. Notice that the function type of fmap is also parametrized with similar binding syntax. The UML syntax of the annotation "[]" to the Function return type is incomplete because the UML modeling tool did not provide adequate type checking or type inference.

Also notice that Functor is a functional interface as reflected in its stereotype. Stereotypes can be combined into UML Profiles that are useful in extending the semantics of UML. We do not discuss profiles further in this article.

Below Functor is Function. Representing Function as a UML interface is consistent with the evolution of mainstream programming languages like Java. What were once object oriented languages now incorporate functional capabilities to strengthen type safety in the language. It too is a functional interface with two parameters A and B. Classes that implement Function must implement its only operation, apply, which takes a single argument that is a type parameter A and returns a single argument type parameter B. PlusOne and PureJust implement Function. They are also parametrized. PlusOne implements a new operation, add, which takes arguments of type Nat and returns results of type Suc. Apply is elided on PlusOne in the diagram to save space.

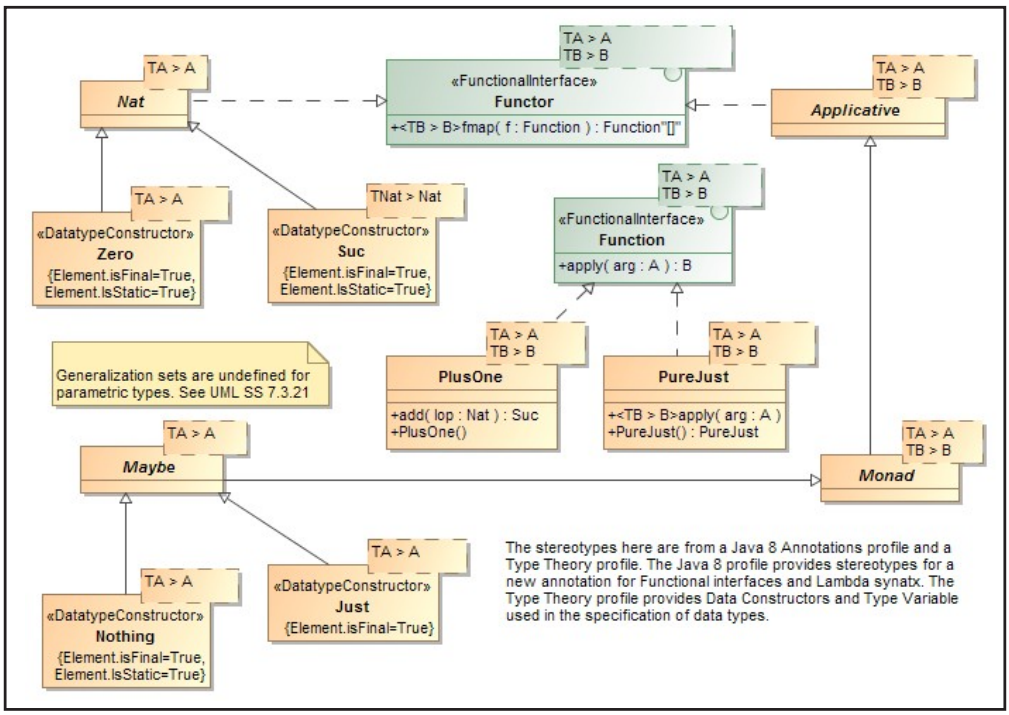


Figure 2 - Overview of Functional Programming Abstractions

To the right of Function both above and below appear the classes Applicative and Monad. Applicative and Monad are parametrized, but the attributes and operations of Applicative and Monad are elided in Figure 2, we elaborate each in detail below. Applicative encapsulates a value and allows the sequencing of computations to combine their results. Monad allows for control in sequencing the effect of a computation.

Figure 3 illustrates the Applicative class with Templates. Applicative has two: A and B bound to TA and TB respectively. Applicative and Monad are equipped with functions that operate on values of the parametrized type. The intuition behind Applicative and Monad are that their operations happen inside the type and that the type system protects the values inside the box. Applicative comes equipped with get, unit and apply. Unit puts a value inside the box. Apply takes an applicative and returns a function that takes an Applicative into Applicative. A function that takes a function into a function is called higher order. It is this function that allows the sequencing of operations. Example 4 below describes the sequencing of operations with Applicative.

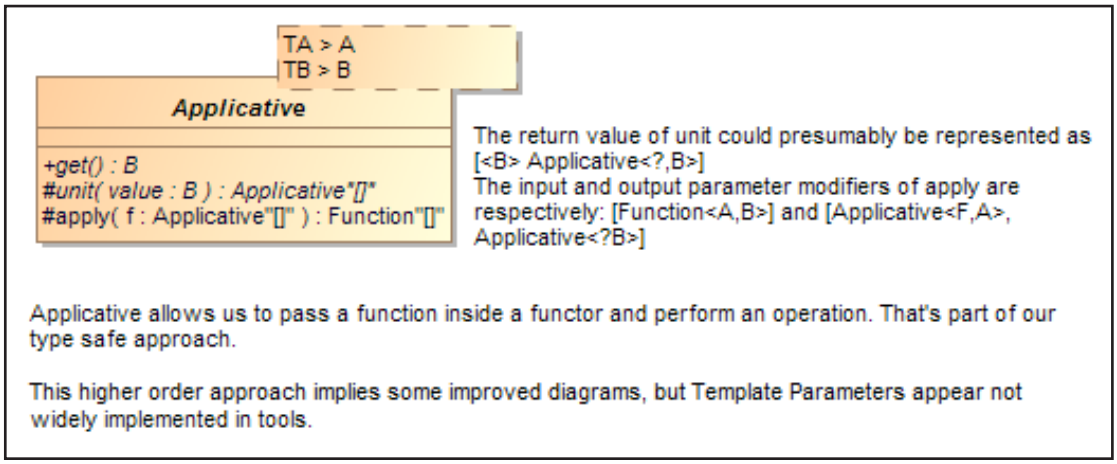


Figure 3 - Applicative Elaboration

Figure 4 elaborates the Monad class. Like Applicative, Monad has two parameters A and B bound to TA and TB respectively. Monad comes equipped with the following operations: return, bind, yield, join, fail, fmap and map. Return takes a value and encapsulates it in the monad. Bind takes an encapsulated value and a function that operates on the encapsulated value and returns the result of the function on the value. Join takes an encapsulated value and reduces the encawpsulation by one level. Yield returns an encapsulated value. Fail takes string explaining the error into the monad. Monad also includes operations inherited from Functor and Applicative.

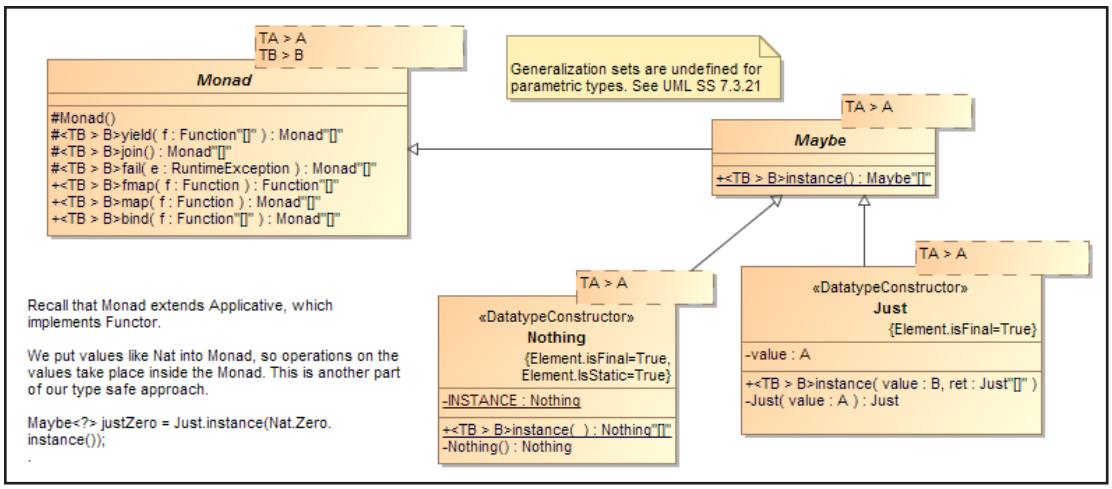


Figure 4 - Monad and Maybe Type Elaboration

Figure 4 also elaborates the Maybe class. In its elided state in Figure 3 Maybe had little resemblance to Nat. In Figure 4 we see that it is an abstract parametrized class with two final subclasses. We also see that Maybe is a Monad. It provides an abstract parametrized method, instance, that returns Maybe. The instance method is inherited by its sub classes, Nothing and Just. Nothing is static and final and an inner class. It is a singleton with an instance method inherited from Maybe. Just is final, but not static, nor is it an inner class. It has single, private constructor. Just overrides instance with the single input parameter that is a type variable and it returns Just [].

Now that we have a static view of the functional abstractions with attributes and operations of each class and interface we provide a dynamic view of their type safe operations using sequence diagrams.

Example 3 - A Dynamic View of Type Safe Operations - Lift PlusOne

We have seen that Functor, Applicative and Monad encapsulate values. The type checker ensures that operations on those values execute according to progress and preservation rules to stop the program from going wrong. Both arguments and results must satisfy progress and preservation rules. Figure 5 uses a sequence diagram to illustrate how to lift the function PlusOne into Just using fmap. Lifting the function into Just means the rules applicable to Maybe and Just are now applicable to PlusOne. Following the hierarchy in Figure 2, Just is a data constructor of Maybe. Maybe is a Monad and Monad is a Functor.

Evaluation begins in step 1 by constructing the encapsulated value Zero in the Just monad. In the next step 2 evaluation proceeds by constructing an instance of the function PlusOne. Step 3 proceeds with the evaluation of fmap which takes the function PlusOne as an argument. Notice the result type of fmap is a function which takes a Functor into a Functor. Step 4 lifts the function justPlusOne into Just. It does not apply justOne to Just Zero. The type checker now enforces progress and preservation rules on the encapsulated PlusOne. Also notice the syntax in the yellow comment box below step 4. On the left side of the equality the type checker has determined the returned function has an input type of a Nat encapsulated in Just and a result type of a successor encapsulated in Just. It makes sense because PlusOne increments Zero to its successor Suc Zero. We know that value as the number One ! Nothing else is allowed by the type checker.

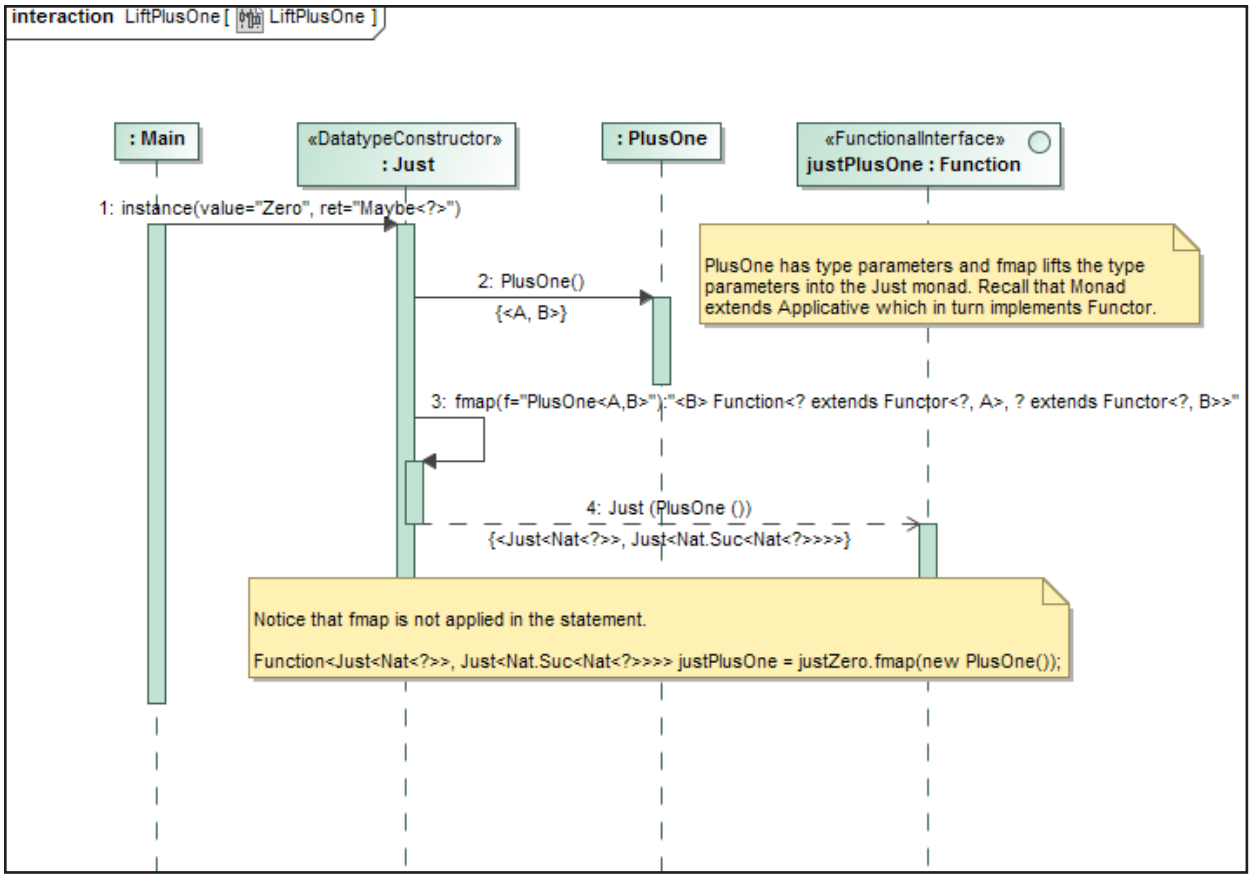


Figure 5 - A Dynamic View of Type Safe Operations - Lift PlusOne

Example 4 - A Dynamic View of Type Safe Operations - Safe Apply with Applicative

We have encapsulated Zero in Just and lifted the function PlusOne into the Just monad. The result is a function called justPlusOne. The type checker requires it to take only a Nat encapsulated in Just and return its successor. Our goal is to safely apply justPlusOne to the encapsulated value Just Zero.

Figure 6 illustrates the type safe application of justPlusOne using Applicative. Evaluation proceeds in step 1 with a call to apply on justPlusOne, the function returned in Figure 5. In step 2 the call to yield with argument PlusOne constructs an instance of Just. Notice that the type checker enforces the result type to Monad in steps 1 & 2. The call to unit in step 3 encapsulates its argument PlusOne in Just. Recall that Applicative is the superclass of Monad. In step 4, as required from the inheritance hierarchy, the compiler invokes the Applicative constructor when it constructs an instance of Just. Step 4 returns an instance of Applicative with two parameters: a function of two parameters and a type variable. Evaluation proceeds in step 5 with a call to apply in Applicative. Its argument is the encapsulated PlusOne from step 3. Its result type is a parametrized function that takes an Applicative into an Applicative. Step 6 evaluates apply in Just with argument Just Zero. Notice that step 6 contains two calls to the function get in Applicative. It is in step 7 that applicative accesses the lifted function PlusOne and in step 8 the encapsulated value Zero. Step 9 is a call to unit in Applicative. Its argument is the function PlusOne with argument Zero. In step 10 evaluation continues on apply with argument Zero in PlusOne. Step 11 is a call to the class PlusOne's add function and in step 12 the call to add creates an instance of the Suc constructor with argument Zero. In step 13 the stack unwinds with the result Just (Suc Zero). Notice that execution returns within Monad and with the reference justOne. Notice the syntax in the yellow comment just above step 13. To the left of the equals sign we see the type checker identifies justOne as a reference to the Maybe Monad with successors as input and successors as a result.

This section provided four examples of type safety in a technical architecture with UML Templates. The

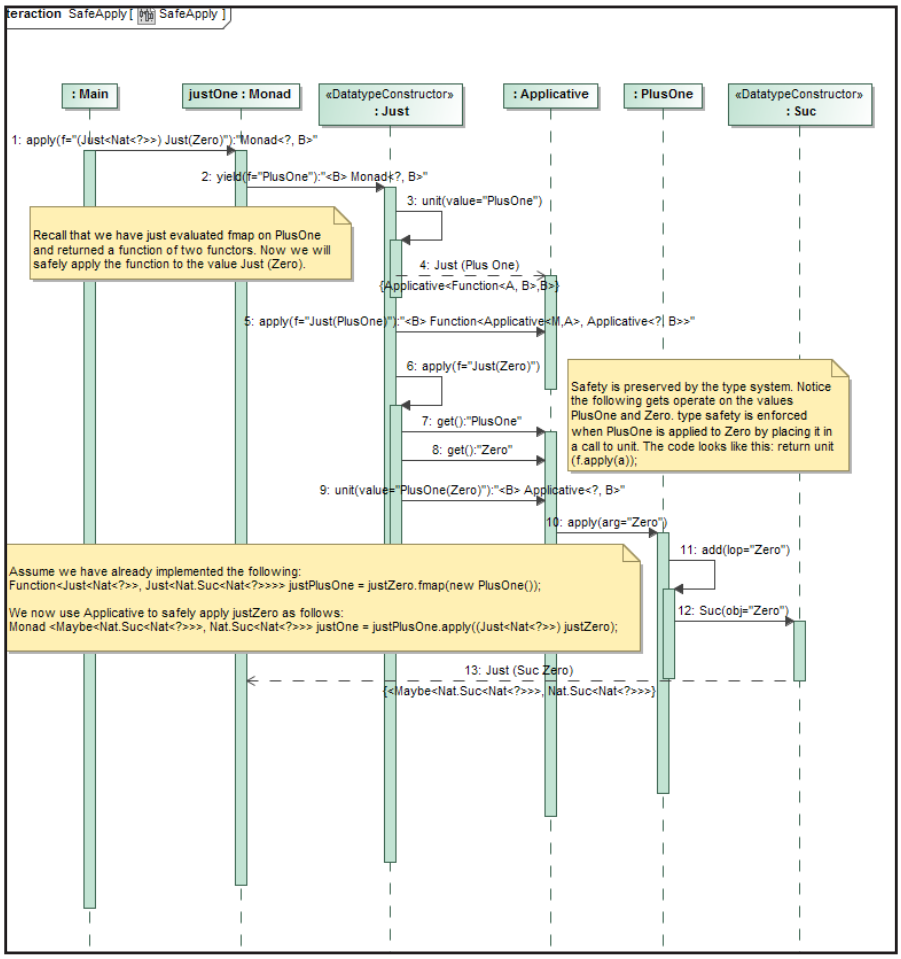


Figure 6 - Safe Apply with Applicative

unabridged version of this article elaborates on the information in this section.

We have come a long way on our journey to increasing assurance levels through early verification with type safety. The next section provides code samples that stop programs from going wrong at compile time rather than defer exceptions to run-time. Increasing assurance levels in a type safe architecture is most relevant when the architecture represents a run-time implementation.

Type Safety and Software Verification

Well typed programs don't go wrong, but how do we stop them from going wrong? UML templates are artifacts in a technical architecture that represent computation. What about the computation itself?

This section illustrates how type checking at compile-time stops programs from going wrong. Programs don't go wrong because the type checker stops the compiler. The type checker enforces verification early at compile-time rather than allow exceptions at run-time.

We proceed with an example that illustrates compile-time type checking. The unabridged version of this article provides an additional example in the Agda logical framework using stronger type checking with dependent types.

Example 1 builds on the UML approach above and illustrates type checking with Java Generics. Recall that Generics in Java is a synonym for Templates in UML. We expect more agency programs are familiar with Java, so it serves our needs for technology transition. For more on UML Templates and Java Generics see Generic Architecture for Government : A Modest Proposal for Better Safety and Sharing [GAGM 2014].

Java 1.5 introduced generic types to enable future versions of the language to provide type checking. Type checking evolved in subsequent releases and support for

```
import java.util.LinkedList;
public class M {
        public static void main(String[] args) {
            // byte list
            byte zeroByte = 0;
            byte oneByte = 1;
            LinkedList xs = new LinkedList(); // raw type
            xs.add(zeroByte); xs.add(oneByte);

            // string list
            LinkedList ys = new LinkedList(); // raw type
            ys.add("zero"); ys.add("one");
            String y = (String) ys.iterator().next();

            // list string lists
            LinkedList zss = new LinkedList(); // raw type
            zss.add(ys);

            // cast Object to String
            String z = (String) ((LinkedList) zss.iterator().next()).iterator().next();

            // improper cast, string list treated as list of bytes
            Byte b = (Byte) ys.iterator().next(); // run-time exception, cannot cast string to byte

        }
}
```

Figure 7. Raw Types

```
compile:
    [mkdir] Created dir: /home/rick/java/gen4/build/classes
    [javac] Compiling 1 source file to /home/rick/java/gen4/build/classes
    [javac] /home/rick/java/gen4/src/M.java:8: warning: [unchecked] unchecked call to add(E) as a member of the raw type LinkedList
    [javac]         xs.add(zeroByte); xs.add(oneByte);
    [javac]                ^
    [javac]   where E is a type-variable:
    [javac]     E extends Object declared in class LinkedList
    [javac] /home/rick/java/gen4/src/M.java:8: warning: [unchecked] unchecked call to add(E) as a member of the raw type LinkedList
    [javac]         xs.add(zeroByte); xs.add(oneByte);
    [javac]                           ^
    [javac]   where E is a type-variable:
    [javac]     E extends Object declared in class LinkedList
    [javac] /home/rick/java/gen4/src/M.java:12: warning: [unchecked] unchecked call to add(E) as a member of the raw type LinkedList
    [javac]         ys.add("zero"); ys.add("one");
    [javac]                ^
    [javac]   where E is a type-variable:
    [javac]     E extends Object declared in class LinkedList
    [javac] /home/rick/java/gen4/src/M.java:12: warning: [unchecked] unchecked call to add(E) as a member of the raw type LinkedList
    [javac]         ys.add("zero"); ys.add("one");
    [javac]                         ^
    [javac]   where E is a type-variable:
    [javac]     E extends Object declared in class LinkedList
    [javac] /home/rick/java/gen4/src/M.java:17: warning: [unchecked] unchecked call to add(E) as a member of the raw type LinkedList
    [javac]         zss.add(ys);
    [javac]                 ^
    [javac]   where E is a type-variable:
    [javac]     E extends Object declared in class LinkedList
    [javac] 5 warnings

run:
     [java] Exception in thread "main" java.lang.ClassCastException: java.lang.String cannot be cast to java.lang.Byte
     [java]     at M.main(M.java:22)
     [java] Java Result: 1

BUILD SUCCESSFUL
Total time: 1 second
```

Figure 8. Compiler and Run-time Output

type safety strengthened recently with type inference. Example 1 describes how Java generics eliminate run-time exceptions resulting from incorrect casts. We begin with a review of the Linked List example from the Generic Java Tutorial [BOSW 1998].

Example 1 - Linked List

Figure 7 uses the Linked List raw type from the Java Collections framework. Raw types allow the developer to add objects of any type to the collection and return types of Object from the collection. This implies that when a developer retrieves an object from the collection they must both discern and cast to the correct type to avoid a run-time exception. The code below illustrates the case of an incorrect cast where the developer attempts to cast an object of type String to Byte. The exception occurs at run-time, possibly after the code has already been pushed to production.

With the addition of generics in Java 1.5, the compiler emits the warning that "Note: src/M.java uses unchecked or unsafe operations. Note: Recompile with -Xlint:unchecked for details." The warning allows compilation to complete, producing byte code targeted to the legacy JVM. The appearance of the words "unsafe" and "unchecked" should alarm any reasonable person. They are ignored and the warning often circumvented by most developers. A recompile with -Xlint: unchecked produces the output in Figure 7. In Java an unsafe operation is the evaluation of an expression whose type cannot be verified by the compiler. An unchecked exception is an exception which the language does not require the developer to catch, or handle. Run-time exceptions are unchecked exceptions. The Java Language Specification [JLS 2013] advises the developer that the operation is unsafe and imputes blame to the developer for safety of the system.

Figure 8 lists the compiler and run-time output when we adhere to the warning and implement -Xlint : unchecked. The compiler emits a warning for each call where an unchecked exception occurs. At run-time the the systems reports the ClassCastException that String cannot be cast to Byte. Again, the ClassCastException occurs at run-time, possibly after the code has been pushed to production.

Figure 9 lists the type-safe approach. Each instance of Linked List is parametrized by the type of its element: Byte, String and LinkedList respectively. In each case when an object is added to a collection, it is added according to its type. As expected, the cast from Object to String is no longer required because the elements in LinkedList are LinkedLists of String elements and we extract an element with the known type: String, not Object.

```
import java.util.LinkedList;
public class M2 {
        public static void main(String[] args) {
            // byte list
            byte zeroByte = 0;
            byte oneByte = 1;
            LinkedList<Byte> xs = new LinkedList<Byte>(); // generic type
            xs.add(zeroByte); xs.add(oneByte);

            // string list
            LinkedList<String> ys = new LinkedList<String>(); // generic type
            ys.add("zero"); ys.add("one");

            String y = (String) ys.iterator().next();

            // list string lists
            LinkedList<LinkedList<String>> zss = new LinkedList<LinkedList<String>>(); // generic type
            zss.add(ys);

            // no cast required
            String z = zss.iterator().next().iterator().next();

            // string list treated as list of bytes
            Byte b = ys.iterator().next(); // compile-time error, cannot cast string to byte

        }
}
```

Figure 9. Generic Types

```
Total time: 0 seconds
rick@metho-laptop:~/java/gen4$ ant
Buildfile: /home/rick/java/gen4/build.xml

compile:
    [mkdir] Created dir: /home/rick/java/gen4/build/classes
    [javac] Compiling 1 source file to /home/rick/java/gen4/build/classes
    [javac] /home/rick/java/gen4/src/M2.java:25: error: incompatible types: String cannot be converted to Byte
    [javac]         Byte b = ys.iterator().next(); // compile-time error, cannot cast string to byte
    [javac]                                  ^
    [javac] 1 error

BUILD FAILED
/home/rick/java/gen4/build.xml:10: Compile failed; see the compiler error output for details.

Total time: 1 second
rick@metho-laptop:~/java/gen4$
```

Figure 10. Compile-time Failure. Well typed programs don't go wrong.

In Figure 10 we have the expected result. The type checker stops the compiler with a compile-time error. The compiler emits the error "incompatible types : String cannot be converted to Byte." The code never makes it to production. We have eliminated a class of exceptions early that may not have been discovered until run-time. And run-time is just too late when your systems are under attack.

Generics were added to Java after an installed base of the Java Virtual Machine (JVM) had already been established. To support backwards compatibility, generics were implemented with type erasure [IPW 2002]. Erasure rewrites source code from the format that enforces constraints at compile time to a format that is compatible with the JVM. As the name implies parametrized type information is removed to maintain backwards compatibility. Generic types subject to erasure are called non-reifiable types [N 2007] at run-time. Non-reifiable means only the raw type, not the generic type, can be reconstructed from its representation in the JVM. Heap pollution is an informal term that describes an undesirable coding practice when a developer assigns a variable of a generic type to a variable of a raw type.

Java, like most other programming languages, has been the subject of exploits. Early versions of its type system were the subject of type confusion or type spoofing exploits [MF 1999, S 1997]. It appears that type confusion exploits were disclosed and resolved consistent with industry best practices. Papers [OW 1997, ORW 1997] that describe development of the Pizza compiler, a precursor to Java Generics and Scala, included commentary on security related to the use of generic types. [GAS 2005] restates this commentary. The Java Generics specification and tutorial [GOSW 1998, GOSW2 1998] are silent on this issue and there is no evidence that recent exploits have been reported. Variable argument methods with non-reifiable formal parameters present a special case where improved compiler warnings and errors were introduced to account for blame in claims made that a method is safe [O 2013]. A secure system implies both memory and type safety as well as segmentation of unsafe operations. This is not a general paper on security, but a discussion of type safety implies some discussion of memory safety and security. Safety and trust are widely advertised in Java's security architecture with references to the Java sandbox and platform security models. The current Java platform security model is comprised of permissions; protection domains and security policies; and security managers and access controllers. The platform security model is considered "code-centric" in that it restricts access to operations that a class can perform in the run-time environment. The inclusion of the sun.misc.unsafe library is not widely advertised, but easy to identify through an Internet search and readily accessible to all Java developers. This library allows unsafe operations. Use of this unsafe library is often motivated by performance gains at the risk of memory safety. Oracle conducted an informal survey

of unsafe library usage and the hostile reaction to including the term "unsafe" in its name was startling [VG 2014]. There is clearly a commitment to performance over safety in the developer community. The agencies should consider sun.misc.unsafe as a candidate for static analysis. This paper does not further discuss memory safety or its relation to type safety or security. No inferences should be made that type safety provides memory safety from this or other sections of this article.

Conclusion - Type Safety and Technical Architecture

The goal of this article was to describe how to represent type safety in a technical architecture. Type safety is a common requirement for increasing assurance levels as observed in the DARPA Open Catalog. The article provides four examples from a type safe technical architecture in UML and one example of compile-time type checking. The unabridged version of this article, available on-line, explains what it means to increase assurance levels early in a life cycle. It does so by describing an early evolutionary life cycle including its iterations and associated disciplines. The unabridged version also provides an additional example using stronger compile-time type checking with dependent types.

References

[GAS 2005] Ahmed Ghoneim, Sven Apel, Gunter Saake, Evolutionary Software Life Cycle for Self Adapting Software Systems. 2005.

[BOSW 1998] Gilad Bracha, Martin Odesrsky, David Stoutamire, Phillip Wadler, Making the Future Safe for the Past : Adding Genericity to the Java Programming Language. October, 1998.

[JLS 2013] James Gosling, Bill Joy, Gilad Bracha, Alex Buckley, The Java Language Specification. February, 2013.

[IPW 2002] Atushi Igarishi, Benjamin Pierce, Philip Wadler. Featherweight Java : A Minimal Core Calculus for Java and GJ. January 2002.

[N 2007] Jamie Nino. The Cost of Erasure in Java Generics Type Systems. March 2007.

[MF 1999] Gary McGraw, Edward Felten. Securing Java. January, 1999.

[S 1997] Vijay Saraswat, Java is Not Type Safe. August, 1997.

[OW 1997] Martin Odersky, Philip Wadler. Pizza into Java: Translating Theory into Practice. January, 1997.

[ORW 1997] Martin Odersky, Enno Runne, Philip Wadler. Two Ways to Bake Your Pizza - Translating Parameterised Types into Java. November, 1997.

[GOSW 1998] Gilad Bracha, Martin Odersky, David Stoutamire, and Philip Wadler. GJ Specification. May 1998.

[GOSW2 1998] Gilad Bracha, Martin Odersky, David Stoutamire, and Philip Wadler. GJ: Extending the Java Programming Language with Type Parameters. August, 1998.

[O 2013] Oracle technical staff. Improved Compiler Warnings and Errors When Using Non-Reifiable Formal Parameters with Varargs Methods. 2013.

[VG 2014] Victor Grazi, Unsafe at any Speed; Oracle Surveys community about promoting sun.misc.Unsafe. February, 2014

[TSM 2014] Type Safe Method Plug-in. rickmurphy.org/epf-method-plugin.zip. 2014.

[GAGM 2014] Generic Architecture for Government : A Modest Proposal for Better Safety and Sharing. rickmurphy.org/gag-modest.zip, OMG 2014.

For the unabridged version of this article please visit http://www.rickmurphy.org/early.html

About the Author

Rick Murphy has over 25 years in software architecture and engineering in the private sector, public sector and academia. Rick currently works in higher order functional programming languages and logics and has a particular interest in category theory, type theory and proof theory.

You can reach Rick by email at rick@rickmurphy.org and by phone at (703) 201-9129.

Looking at M&S Education Through the Prism of the Video Game Industry

By John Lawson III

Let's discuss what DOD Modeling & Simulation (M&S) can learn about education from the commercial gaming world. There are occasions when we in DOD look at the commercial gaming industry as if we're kids with our faces pressed against a candy store's window.

In many instances, fundamental differences between DOD and the private sector prevent us from emulating their successes. For example, we sweat through the brow trying to devise persistent Live/Virtual/Constructive environments. Meanwhile, for many years, recreational gamers have had access to the PlayStation Network, the Xbox Live environment, and any number of other persistent online capabilities for PC games, tablet games, phone games, and so forth.

There are reasons why our technology sometimes seems quaint when compared to the entertainment industry's technology. Addressing these reasons probably isn't realistic, even for four-stars or undersecretaries. We in DOD must follow detailed acquisition rules. We must follow security procedures that can have life-or-death ramifications. We must coordinate across organizations, across commands, across services, and even across government. There are, in short, plenty of cases where our slow bureaucratic speed prevents fast technological progress. For all of us worker bees, these are constraints we simply have to live with.

But sometimes we can look at the commercial gaming world, see a good idea, and copy it. Education is one such example.

When the topic of M&S education arises in DOD, an obsession with checklists, processes, and credentials frequently emerges. In fact, we often don't sound like we're discussing education at all. We sound like we're discussing training.

In the commercial gaming world, discussions of education are far less rigid. In other words, their discussions about education sound more intellectual and less like Soviet-style central planning.

According to Gamasutra.com, a gaming industry website, new hires at gaming companies typically have broader majors, such as art, finance, or computer science. They tend not to have tailored majors, i.e., majors with the word "game" in the description.

DOD, on the other hand, has inclinations that go in the opposite direction. We aren't just infatuated with tailored majors and tailored curricula. We are sometimes tempted to devise courses of study in which M&S students have almost no latitude. We imagine that our M&S folks can be precisely codified, like a rifleman who shot marksman, sharpshooter, or expert. Of course, proficiency with a rifle in 2015 isn't far removed from proficiency with a rifle in 1965 or even 1915. On the other hand, M&S is a vast, rich, constantly changing realm. Even if you could pigeonhole your M&S people in 2015 (and I say you can't), your entire notion of how to pigeonhole them would have to change in a matter of months.

M&S is a dynamic area. Education should fuel it, not constrain it. The game development world is NOT madly in love with academic game development credentials. Similarly, we in DOD should NOT be madly in love with academic M&S credentials.

Over in the entertainment realm, even some of those who run game development programs at four-year institutions acknowledge that a curriculum can become dated six months after a school signs off on it. They also note that hiring top teaching talent is a challenge, because top talent usually lacks the academic credentials to land a professorship.

The Internet is full of blogs discussing whether a video gaming degree is worth the time and money, and while there is no consensus, the criticisms have a lot of intellectual traction, and the very existence of so much criticism is a warning against enshrining highly specific academic credentials. Particularly unsettling is the charge that academia moves more slowly than industry and therefore struggles to keep up with technology and related trends.

If the fast-moving commercial gaming world is hesitant to get hung up on ultra-specialized credentials, we in DOD should have even greater reservations.

Whether you're talking about measures that extend across DOD, pertain to a particular service, or confine themselves to some portion of a service (e.g., the training community), you have to recognize a basic, bureaucratic danger. It's pretty easy to imagine ourselves getting married to credentials that would be out of date by the time students left the schoolhouse for the Real World.

We all have a pretty good idea what our bureaucratic behavior looks like...

- Year 1: A study compiles information on credentials
- Year 2: Committees and working groups wrangle over credentials
- Year 3: A schoolhouse prepares to implement the courses
- Year 4: The first batch of students goes through the program
- Year 5: Graduates take what they learned into the workforce

Any process resembling that is destined to be slow, and slow is bad when you're talking technology. If you know your organization is going to be slow, allow for flexibility. Don't enshrine too many courses. Don't enshrine too many skillsets. Don't enshrine too many credentials. Emphasize education and don't slide too far into professional certification.

According to an article by David Owen for ign.com, a website dedicated to commercial games, those who hire new employees for game developers are more interested in a prospect's portfolio of tangible work, rather than a transcript or a resume.

We should resist our bureaucratic habit of checking into the comfort zone that's laden with checklists and credentials. We should take a page out of the commercial gaming world's book and examine the portfolios, i.e., the productivity, of our M&S folks. People who have been educated in a meaningful way should be able to harness their intellectual horsepower and produce something. Let's think a little less specifically about what happens inside the black box of educational institutions, and let's think a little bit more about outputs.

Education ought to be synonymous with an open mind. If we're thinking there are only a few, narrowly defined ways to educate our M&S personnel, we're probably pursuing our goals in the wrong way. Or, at a minimum, we're probably talking about training rather than education.

Our quasi-colleagues in the commercial gaming industry have a variety of academic backgrounds. Computer science is common, but there are plenty with backgrounds in math, physics, English, graphic design, etc.

And even when the commercial gaming industry does hire students who went to school with an eye toward specialization, the industry is open to a wide array of specialized students. There are at least several dozen colleges and universities with impressive curricula for game developers. However, a look at the diversity of these schools and the diversity of their academic approaches ought to deter DOD from following a cookbook mentality about M&S education. If we fixate on a few credentialing recipes, we'll struggle to keep up technologically.

Nobody ever accused the gaming industry of struggling to keep up technologically.

About the Author

John Lawson III is a contractor who supports the Marine Corps M&S Management Office. Before that, he was a contractor serving several Air Force entities as an analyst. He has spent more than a decade monitoring commercial game technology and its potential utility for military analysis and military M&S. In the 1990s, he was a newspaper reporter, mainly for The Tampa Tribune; he is also the author of Tom Landry and Bill Walsh: How two coaching legends took championship football from the Packer Sweep to Brady vs. Manning. Lawson served in the Marine Corps Reserve for nine years and reached the rank of staff sergeant. He has a B.S. in mechanical engineering from the University of Maryland; a B.A. from Washington & Lee University for a double major in history and English; and an M.A. from the University of Florida in mass communications.

Best Practices

Over the past year the Software Engineering Institute (SEI), in coordination with the Assistant Secretary of Defense for Research and Engineering for Acquisition, Technology, and Logistics (ASD(R&E) AT&L), together with CSIAC produced five software best/recommended practices:

Agile at Scale. Agile practices have been used for well over a decade and have enjoyed much success and broad adoption in the commercial sector. But business and mission goals are larger than a single development team, and applying agile at scale is challenging along several dimensions. These recommended practices, orchestrated together, will help enable agility at scale.

Managing Operational Resilience. Organizations have invested a tremendous amount of resources in cybersecurity, yet cyber attackers continue to penetrate systems. An organization should pursue a strategic approach that balances actions that protect assets with actions that sustain services and operations. Managing operational resilience includes all the practices of planning, integrating, executing, and governing these activities.

Managing Intellectual Property in the Acquisition of Software-Intensive Systems. Department of Defense regulations now require that programs develop an intellectual property (IP) strategy as part of the acquisition strategy. These recommended practices focus on managing IP for acquisitions, with emphasis on noncommercial software. They include planning and consideration of data rights and licenses throughout the life cycle of the acquisition.

Monitoring Software Intensive System Acquisition. Effective program management requires maintaining an accurate understanding of a program's status, quickly identifying issues that threaten program objectives, and dealing with them efficiently. These recommended practices implement an approach called the "program dashboard" that helps the program manager and contractor come to a mutual understanding of a program's progress and the significance of deviations from expectations.

Safety-Critical (SC) Systems. For safety-critical systems, failure may cause serious injury to people, damage to equipment, or environmental harm. As the needs for real-time and fail-safe performance become more stringent, it becomes harder to develop and evolve such systems. These recommended practices help an organization successfully develop and sustain safety-critical systems.

The CSIAC Journal is pleased to present these Best Practices over the next several issues, beginning with ***Agile at Scale*** and ***Managing Operational Resilience*** in this issue.

Agile at Scale (AAS)

By Robert L. Nord, Ipek Ozkaya - Software Engineering Institute

There are four parts to our discussion of Agile at scale. First, we set the context by providing an answer to the question, "Why is AAS challenging?" The ten AAS primary technical best practices follow. We then briefly address how an organization can prepare for and achieve effective results from these best practices. We conclude with a listing of selected resources to help you learn more. Also, we've added links to various sources to help amplify a point—be mindful that such sources may occasionally include material that might differ from some of the recommendations below.

Every organization is different; judgment is required to implement these practices in a way that provides benefit to your organization. In particular, be mindful of your mission, goals, existing processes, and culture. All practices have limitations—there is no "one size fits all." To gain the most benefit, you need to evaluate each practice for its appropriateness and decide how to adapt it, striving for an implementation in which the practices reinforce each other. Also, consider additional best practice collections (such as the one from the GAO that is referenced at the end of this article). Monitor your adoption and use of these practices and adjust as appropriate.

These practices are certainly not complete—they are a work in progress. For example, as future additions we plan to include webpages addressing management and acquisition best practices for AAS.

Why is AAS Challenging?

Agile practices, derived from a set of foundational principles, have been used for well over a decade and have enjoyed much success and broad adoption in the commercial sector with the net result that development teams have gotten better at building software. Reasons include: increased visibility into a project and the emerging product, increased empowerment of development teams, the ability for customers and end users to interact early with executable code, and the direct engagement of the customer or product owner in the project to provide a greater sense of shared responsibility.

But business and mission goals are larger than a single development team and thus applying AAS is challenging along these dimensions:

1. **Team size**. What happens when Agile practices are used in a 100-person (or larger) development team? What happens when the development team needs to interact with the rest of the business such as quality assurance, system integration, project management, and marketing to get input into product development and collaborate on the end-to-end delivery of the product? Scrum and Agile methods such as extreme programming (XP) are typically used by small teams of at most 7-10 people. Larger teams require orchestration of both multiple (sub)teams and cross-functional roles beyond development.

2. **Complexity**. Large-scale systems are often large in scope in terms of the number of features, the amount of new technology being introduced, the number of independent systems being integrated, the number and types of users to be accommodated, and the number of external systems with which the system communicates. Does the system have stringent quality-of-service requirements (e.g., strict real-time, high-reliability, and security requirements)? Are there multiple external stakeholders and interfaces? Typically, such systems must go through rigorous verification and validation (V&V), which makes the frequent-deployment practices used in Agile development challenging.

3. **Duration**. How long will the system be in development? How long in operations and sustainment? Larger systems need to be in development and operation for a longer period of time than products to which agile development is typically applied, requiring attention to future changes, possible redesigns as well as maintaining several delivered versions. This is a focus that some Agile teams would consider antithetical to the Agile principles. Answers to these questions affect the choice of quality attributes supporting system maintenance and evolution goals that are key to system success over the long term.

AAS Best Practices:

1. **Use Scrum of Scrums carefully when coordinating multiple teams**. Scrum is the most often used Agile method in today's environment, and primarily involves team management practices. In its simplest instantiation, a Scrum development environment consist of a single Scrum team with the skills, authority and knowledge required to specify requirements, architect, design, code, and test the system. As systems grow in size and complexity, the single team mode may no longer meet development demands. If a project has already decided to use a Scrum-like project-management technique, the Scrum approach can be extended to managing multiple teams with a "Scrum of Scrums," a special coordination team whose role is to (1) define what information will flow between and among development teams (addressing inter-team dependencies and communication) and (2) identify, analyze, and resolve coordination issues and risks that have potentially broader consequences (e.g., for the project as a whole). A Scrum of Scrums typically consists of members from each team chosen to address end-to-end functionality or cross-cutting concerns such as user interface design, architecture, integration testing, and deployment. Creating a special team responsible for inter-team coordination helps ensure that the right information, including measurements, issues, and risks, is communicated between and among teams. But care needs to be taken when the Scrum of Scrums team itself gets large to not overwhelm the team. This can be accomplished by organizing teams—and the Scrum of Scrums team itself—along feature and service affinities. We further discuss this approach to organizing teams in our Feature-Based Development and System Decomposition practice. Such orchestration is essential to managing larger teams to success, including Agile teams.

2. **Use an architectural runway to manage technical complexity**. Stringent safety or mission-critical requirements increase technical complexity and risk. Technical complexity arises when the work takes longer than a single iteration or release cycle and cannot be easily partitioned and allocated to different technical competencies (or teams) to independently and concurrently develop their part of a solution. Successful approaches to managing technical complexity include having the most-urgent system or software architecture features well defined early (or even pre-defined at the organizational level, e.g., as infrastructure platforms or software product lines).

   The Agile term for such pre-staging of architectural features that can be leveraged by development teams is "architectural runway." The architectural runway has the goal of providing the degree of stability required to support future iterations of development. This stability is particularly important to the successful operation of multiple teams. A system or software architect decides which architectural features must be developed first by identifying the architecturally significant requirements for the system. By initially defining (and continuously extending) the architectural runway, development teams are able to iteratively develop customer-desired features that leverage that runway and benefit from the quality attributes they confer (e.g., security).

   Having a defined architectural runway enables technical risks to be uncovered earlier, thereby helping to manage system complexity (no late surprises). The consequence of uncovering underlying architectural concerns such as security, performance, or availability late—that is, after several iterations have passed—often is a significant rework rate and schedule delay. Delivering functionality is more predictable when the infrastructure for the new features is in place so it is important to maintain a continual focus on the architecturally significant requirements and estimation of when the development teams will depend on having code that implements an architectural solution.

3. **Align Feature-Based Development and System Decomposition.** A common approach in Agile teams is to implement a feature (or user story) in all the components of the system. This gives the team the ability to focus on something that has stakeholder value. The team controls every piece of implementation for that feature and therefore they do not have to wait until someone else outside the team has finished some required work. We call this vertical alignment because every component of the system required for realizing the feature is implemented only to the degree required by the team.

   However, system decomposition could also be horizontal, based on the architectural needs of the system, focusing on common services and variability mechanisms promoting reuse.

   The goal of creating a feature-based development and system decomposition approach is to provide flexibility in aligning teams horizontally, vertically, or in combination, while minimizing coupling to ensure progress. Although organizations create products in very different domains (embedded systems to enterprise systems) similar architecture patterns and strategies emerge when a need to balance rapid progress and agile stability is desired. The teams create a platform containing commonly used services and development environments either as frameworks or platform plug-ins to enable fast feature-based development.

4. **Use quality-attribute scenarios to clarify architecturally significant requirements.** Scrum emphasizes customer-facing requirements—features that end users dwell on—and indeed these are important to success. But when the focus on end-user functionality becomes exclusive, the underlying architecturally significant requirements can go unnoticed.

   Superior practice is to elicit, document, communicate, and validate underlying quality-attribute scenarios during development of the architectural runway. This becomes even more important at scale when projects often have significant longevity and sustainability needs. Early in the project, evaluate the quality-attribute scenarios to determine which architecturally significant requirements need to be addressed in early development increments (see architectural runway practice above) or whether strategic shortcuts can be taken to deliver end-user capability more quickly.

   For example, will the system really have to scale up to a million users immediately, or is this actually a trial product? There are different considerations depending on the domain; for example, IT systems use existing frameworks, so understanding the quality-attribute scenarios can help developers understand which architecturally significant requirements might already be addressed adequately within existing frameworks (including open-source systems) or existing legacy systems that can be leveraged during software development. Similarly, such systems have to deal with changing requirements in security and deployment environments that necessitates architecturally significant requirements to be top priority when dealing with scale.

5. **Use test-driven development for early and continuous focus on verification**. This practice can be summarized as "write your test before you write the system." When there is an exclusive focus on "sunny-day" scenarios (a typical developer's mindset), the project becomes overly reliant on extensive testing at the end of the project to identify overlooked scenarios and interactions. Therefore, be sure to focus on rainy-day scenarios (e.g., consider different system failure modes) as well as sunny-day scenarios. The practice of writing tests first, especially at the business or system level (which is known as acceptance test-driven development) reinforces the other practices that identify the more challenging aspects and properties of the system, especially quality attributes and architectural concerns (see architectural runway and quality-attribute scenarios practices above).

6. **Use end-to-end testing for early insight into emerging system properties**. To successfully derive the full benefit from test-driven development at scale, consider early and continuous end-to-end testing of system scenarios. When teams test only the features for which they are responsible, they lose insight into overall system behavior (and how their efforts contribute to achieving it). Each small team could be successful against its own backlog, but someone needs to be looking after broader or emergent system properties

and implications. For example, who is responsible for the fault tolerance of the system as a whole? Answering such questions requires careful orchestration of development with verification activities early and throughout development. When testing end to end, take into account different operational contexts, environments, and system modes.

At scale, understanding end-to-end functionality requires its elicitation and documentation. This can be achieved through use of agile requirements management techniques such as stories as well as use of architecturally significant requirements. However, if there is a need to orchestrate multiple systems, a more deliberate elicitation of end-to-end functionality as mission/business threads should provide a better result.

7. **Use continuous integration for consistent attention to integration issues**. This basic Agile practice becomes even more important at scale, given the increased number of subsystems that must work together and whose development must be orchestrated. One implication is that the underlying infrastructure that developers will use day to day must be able to support continuous integration. Another is that developers focus on integration earlier, identifying the subsystems and existing frameworks that will need to integrate. This identification has implications for the architectural runway, quality-attribute scenarios, and orchestration of development and verification activities. Useful measures for managing continuous integration include rework rate and scrap rate. It is also important to start early in the project to identify issues that can arise during integration. What this means more broadly is that both integration and the ability to integrate must be managed in the Agile environment.

8. **Consider technical debt management as an approach to strategically manage system development**. The concept of technical debt arose naturally from use of Agile methods, where the emphasis on getting features out quickly often creates a need for rework later. At scale, there may be multiple opportunities for shortcuts, and understanding technical debt and its implications becomes a means for strategically managing the development of the system. For example, there might be cases, where to accelerate delivery, certain architectural selections are made that have long-term consequences. Such tradeoffs must be understood and managed based on both qualitative and quantitative measurements of the system. Qualitatively, architecture evaluations can be used as part of the product demos or retrospectives that Agile advocates. Quantitative measures are harder but can arise from understanding productivity, system uncertainty, and measures of rework (e.g., when uncertainty is greater, you might be more willing to take on more rework later). Several larger organizations have started to look into technical-debt management practices organizationally.

9. **Use prototyping to rapidly evaluate and resolve significant technical risks**. To address significant technical issues, teams employing Agile methods will sometimes perform what in Scrum is referred to as a technical spike, in which a team branches out from the rest of the project to investigate the specific technical issue, develop one or more prototypes to evaluate possible solutions, and bring back what was learned to the project so that it can proceed with greater likelihood of success. A technical spike may extend over multiple sprints, depending on the seriousness of the issue and how much time it takes to investigate the issue and bring back information that the project can use.

   At scale, technical risks having severe consequences are typically more numerous, and so prototyping (and other approaches to evaluating candidate solutions such as simulation and demonstration) can be an essential early planning but also recurring activity. A goal of Agile methods is increased early visibility. From that perspective, prototyping is a valuable means of achieving visibility more quickly for technical risks and their mitigations. The Scrum of Scrums practice mentioned earlier has a role here, too, for helping to orchestrate bringing back what was learned from prototyping to the overall system.

10. **Use architectural evaluations to ensure that architecturally significant requirements are being addressed.** While not considered part of mainstream Agile practice, architecture evaluations have much in common with Agile methods in seeking to bring a project's stakeholders together to increase their

visibility into and commitment to the project, and to identify overlooked risks. At scale, architectural issues become even more important, and architecture evaluations thus have a critical role on the project. Architecture evaluation can be formal, as in the Software Engineering Institute's Architecture Tradeoff Analysis Method, which can be performed, for example, early in the Agile project lifecycle before the project's development teams are launched, or recurrently. There is also an important role for lighter weight evaluations in project retrospectives to evaluate progress against architecturally significant requirements.

Under what conditions will organizations derive the most benefit from the AAS best practices?

None of these practices in isolation will enable agility at scale. They are meant to be orchestrated together. Improving visibility and understanding into high priority concerns for the system under development and understanding the technical challenges hindering their development early on and continuously is what enabled agile development practices to succeed in its initial context. Carrying that to scale means making sure the technical barriers and enablers are clearly communicated through not only team practices but through the working system as well. When an organization neglects the following factors, the effectiveness of AAS practices, and of Agile more generally, may be severely limited:

1. **A technical infrastructure that empowers the teams to collaborate**. An infrastructure that supports such things as configuration management; issue and defect tracking; and team measurement and analysis are extremely important for Agile and AAS practices. For example, a large Agile project with distributed teams may lack something as simple as a standard virtual-meeting capability to support daily standup meetings.

2. **A management culture that empowers and trusts team decisions**. Agile practices assume empowerment of development teams. Technical decisions made at the development level should be trusted and propagated to other teams and management that might be affected. More generally, communication barriers must be removed, and management must create a culture that removes silos, particularly around interdependent work.

   One key is ensuring that team members have the training and mentoring they need to make sound technical judgments. Teams must be empowered and encouraged to define their own work processes, define the measurements they will collect and analyze, and regularly evaluate the quality of their work and gauge the progress made.

   Strongly hierarchical decision-making organizations may experience significant challenges as they try to transition to such a culture: development teams may be used to being told what to do and may experience unease taking the initiative, and their management may remain uneasy in granting teams that initiative.

3. **Visibility**. Agile is all about achieving visibility early and continuously and recognizing and addressing risks in a timely way. The challenge with knowledge work is that though work processes may be "proven" across a range of circumstances, they nevertheless represent theories of how the work should proceed (theories that can improve with time); thus, team processes should be measured, monitored, and adjusted as needed.

   One key to greater visibility and understanding is to make all team artifacts that contribute to the development of the system broadly accessible to everyone in the project. Many open-source efforts now employ social coding environments—such as GitHub—that provide full transparency into each developer's work. More generally, it is not possible to fully anticipate who needs to know about team progress and issues, now or in the future, and thus the environment should make working code, team and project backlogs, and quality-attribute priorities visible to all.

Learn More

For more information about Agile at scale, please see:

Leffingwell, Dean. *Scaling Software Agility: Best Practices for Large Enterprises.* Addison-Wesley, 2007.

Government Accountability Office. *Software Development: Effective Practices and Federal Challenges in Applying Agile Methods*. Report GAO-12-681. July 2012. http://www.gao.gov/products/GAO-12-681

Larman, Craig and Vodde, Bas, *Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum*

Stephany Bellomo, Robert L. Nord, Ipek Ozkaya: *A Study of Enabling Factors for Rapid Fielding: Combined Practices to Balance Speed and Stability.* ICSE 2013: 982-991

For more information about architectural tactics and Agile, please see:

Royce, W. *Measuring Agility and Architectural Integrity*, Int'l J. Software and Informatics, vol. 5, no. 3, 2011, pp 415-433.

For more information on Agile for the enterprise and teams, please see

Leffingwell, Dean. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise.* Addison-Wesley, 2011.

To learn more about the interplay of Agile at scale best practices, see:

Integrate End to End Early and Often, IEEE Software July/. August 2013 issue, Felix Bachmann et al

Government Accountability Office. *Software Development: Effective Practices and Federal Challenges in Applying Agile Methods*. Report GAO-12-681. July 2012.

For more information about quality attribute scenarios, please see:

Ipek Ozkaya, Len Bass, Raghvinder Sangwan and Robert Nord. *Making Practical Use of Quality Attribute Information*, in IEEE Software Volume 25 Issue 2 March-April 2008, Page(s): 25-33.

Leffingwell, Dean. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise.* Addison-Wesley, 2011.

To learn more about test-driven development, see:

Whittaker, James A., Jason Arbon and Jeff Carollo: *How Google Tests Software* (Apr 2, 2012)

Beck, Kent: *Test Driven Development by Example*

Learn more about continuous integration by seeing:

Continuous Integration: Improving Software Quality and Reducing Risk. Paul M. Duvall; Steve Matyas; Andrew Glover; Addison-Wesley Professional, 2007

To view information about technical debt, please visit

Philippe Kruchten, Robert L. Nord, Ipek Ozkaya. *Technical debt: from metaphor to theory and practice*. IEEE Software Special Issue on Technical Debt (Nov/Dec 2012).

To learn more about prototyping, see:

Stephany Bellomo, Robert L. Nord, Ipek Ozkaya. *Elaboration on an Integrated Architecture and Requirement Practice: Prototyping with Quality Attribute Focus*. Second International Workshop on the Twin Peaks of Requirements and Architecture. International Conference on Software Engineering (ICSE) 2013, May 18-26, 2013 in San Francisco, CA, USA.

About the Author(s)

Robert L. Nord is a senior member of the technical staff at the Carnegie Mellon Software Engineering Institute (SEI). He is engaged in activities focusing on agile architecting, architectural technical debt, and effective methods and practices for software architecture. He is coauthor of the practitioner-oriented books *Applied Software Architecture* and *Documenting Software Architectures*: Views and Beyond and lectures on architecture-centric approaches.

Ipek Ozkaya works to develop, apply, and communicate effective methods for software architecture and agile and iteratire development to improve software development efficiency. At the SEI she is the deputy lead for the Architecture Practices (AP) initiative and the technical lead for the the Value-driven Incremental Development research project.

Managing Operational Resilience

By Julia H. Allen, Pamela Curtis, Nader Mehravari - Software Engineering Institute

A search at your favorite news aggregator for keywords such as "malware," "computer virus," or "data breach" will return results in the tens of thousands. For most organizations it's not a question of ***if*** a cyber attack will occur, but ***when***. And when an attack happens, the tempo of response must be fast, so an organization must already have practices in place covering how to respond. These practices should reflect a strategic approach that balances actions that protect assets such as customer data and intellectual property with actions that sustain services and operations.

A recommended approach to address both protection and sustainment is the application of resilience management practices. *Operational resilience* is the ability of an entity to prevent disruptions to its mission from occurring, continue to meet its mission if a disruption or incident does occur, and return to normalcy when the disruption is eliminated. The concept of operational resilience applies to entities such as organizations, systems, networks, supply chains, critical infrastructure, cyberspace, Armed Forces, and even nations.

Operational resilience management includes all the practices of planning, integrating, executing, and governing activities to ensure that an entity can

- identify and mitigate operational risks that could lead to service disruptions before they occur
- prepare for and respond to disruptive events (realized risks) in a manner that demonstrates command and control of incident response and service continuity
- recover and restore mission-critical services and operations following an incident within acceptable time frames

Operational resilience management draws from several complex and evolving disciplines, including risk management, business continuity, disaster recovery, information security, incident and emergency management, information technology (IT), service delivery, workforce management, and supply-chain management, each with its own terminology, principles, and solutions. The practices described here reflect the convergence of these distinct, often siloed disciplines. As resilience management becomes an increasingly relevant and critical attribute of their missions, organizations should strive for a deeper coordination and integration of its constituent activities.

Our discussion of operational resilience management has four parts. First, we set the context by providing an answer to the question "Why is operational resilience management challenging?" A set of recommended practices for operational resilience management follows. We then briefly address how an organization can achieve effective results by following these practices. We conclude with a list of selected resources to help you learn more about operational resilience management. Also, we've added links to various sources to help amplify some points.

Every organization is different; judgment is required to implement these practices in a way that benefits your organization. In particular, be mindful of your mission, goals, existing processes, and culture. All practices have limitations. Some of these practices will be more relevant to your situation than others, and their applicability will depend on the context in which you apply them. To gain the most benefit, you need to evaluate each practice for its appropriateness and decide how to adapt it, striving for an implementation in which the practices meet your business objectives. Monitor your adoption and use of these practices, and adjust as appropriate.

Why is managing operational resilience challenging?

Over the past 10 years, organizations have invested a tremendous amount of resources in cybersecurity. Nevertheless, regardless of how much has been spent on protection, cyber attackers continue to penetrate systems. We have reached a point in the battle for information and cybersecurity where we should change the focus of security investment from a narrow focus on planning how to avoid cyber attacks to a more balanced focus on avoidance *and* planning how to recover from cyber attacks.

Operational resilience management has two sides—protect and sustain—and both are equally important. An organization must learn about the threat environment, maintain situational awareness of the context in which it operates, and create a risk-management plan that is as thorough and reliable as possible. But when an attack occurs, can the organization sustain its critical services and operations? Can it adequately recover its systems and get them back online as quickly as possible? Can it restore and recover service within a prescribed recovery time and according to its recovery-point objectives? An organization must ask, where can we not afford to have something bad happen, and where can we afford to have something bad happen and bounce back as quickly as we can? The need for organizations to achieve a balance between protect and sustain is why operational resilience management is so important.

Operational resilience management is challenging for several reasons:

1. **Making a long-term commitment:** Operational resilience is an emergent property. An *emergent property* is not something an organization can buy and put in place or assemble by buying its parts. For a property to emerge within an organization, the organization must execute a certain set of activities in a coordinated manner and do so with consistent discipline. Our own health makes a good analogy: we would all like to have good health, but we cannot buy it at any store. To become healthy, we must do certain good things, such as eat well, exercise, sleep enough, and get checkups. And we must do these things in a disciplined manner for a long time. Achieving operational resilience requires an organization to make a similar long-term commitment to perform certain activities with consistency. The activities involved in operational resilience management must become part of the organization's daily habits across the enterprise.

2. **Understanding the big picture:** To be operationally resilient, organizations must address operational risk on many dimensions simultaneously, including people, technology, information, facilities, supply-chain, management, cyber, and physical dimensions. This requires careful planning, coordination, and training across many interdependent domains, as well as understanding how the organization's capabilities along these dimensions contribute to mission success.

3. **Overcoming organizational hurdles:** An organization may encounter these barriers to operational resilience management:

    - the vague and abstract nature of operational risk management
    - compartmentalization of operational risk-management activities, such as segmenting responsibilities for information security and business continuity/disaster recovery
    - focusing on technology instead of on all the dimensions listed in Challenge 2
    - the proliferation of practices for operational resilience management
    - insufficient funding and staff
    - insufficient success stories and measurements
    - (over)reliance on people
    - regulatory climate
    - existing policies
    - the tendency to ignore current information to avoid a painful reality and the need to act
    - competitive pressures or short-term goals

Recommended Practices for Managing Operational Resilience in Organizations

1. **Governance and program management.** Organizations must oversee and manage the execution of resilience activities. Resilient organizations ensure that all such activities derive their purpose and focus from strategic objectives and critical success factors for operational resilience. The governance and program-

management practice ensures that the investment in operational resilience, cybersecurity, service continuity, and other domains is consistent with the organization's business objectives. This practice entails regular planning, definition of roles and responsibilities, adequate funding, appropriate resource allocations, oversight in executing the plan, and corrections as necessary. In addition, governance and program management involves measuring, analyzing, and reporting the effectiveness of resilience-management practices and implementing improvements. These are all standard business practices for successful, mature organizations, but they are often overlooked when managing operational resilience.

2. **Staff preparation and deployment**. Organizations must be prepared when a disruptive event occurs. That means making sure that staff at all levels of the organization are trained in how to perform their assigned roles when disruptions occur. Everyone must know his or her role, receive training, and rehearse plans and contingencies. Skill gaps and deficiencies should be identified and training provided to address them.

   Training can be designed to help meet the goals of resilience management as well as other goals of the organization that depend on interdisciplinary team performance. For example, teams with members drawn from different disciplines and departments can train together in a scenario that encourages interaction, mutual understanding, and building trust among team members. Such training breaks down barriers that otherwise naturally arise when work must be done across disciplines and departments.

   This practice also encompasses establishing staff backup and redundancy at all levels of the organization. For key personnel, not only it is important to have backups who can step in; organizations should also have identified qualified successors to staff members in key positions if those positions are vacated.

   Training is not a one-time event. The organization should provide periodic refreshment training for all key functions so that responsibilities and skills are not forgotten in the stress of disruptive events.

3. **Communication and awareness**. Resilient organizations make establishing and maintaining communications with stakeholders a key objective in all operational resilience-management practices—both during normal operations and during periods of stress. Communication is always important, but it is particularly essential during times of disruption. The organization should plan in advance exactly who will contact whom during and following disruptive events. Plan who will communicate with stakeholders, including both customers and suppliers, to share information and make stakeholders aware of the status of the situation. In addition, develop communication methods (newsletters, email notifications, community meetings, etc.), channels (public relations activities, peer and professional organizations, etc.), infrastructure, and systems (such as emergency alerting via mobile devices).

   This practice includes both internal and external communication. An organization should report ongoing measurement of operational performance and resilience-management activities and disseminate that information across the enterprise to ensure that all organizational units are operating with an up-to-date picture of the organization's operations. External communication tasks may require providing information to news media about its resilience efforts or efforts to contain an incident or event. As appropriate, establish responsibility for planning for and executing crisis communications among first responders, other emergency and public service staff, and law enforcement.

4. **Risk management.** Organizations must identify, analyze, and mitigate risks to assets that could adversely affect the operation and delivery of high-value services. Because an organization cannot protect against every possible threat, risk management involves identifying critical services and operations, identifying the assets that enable their delivery, and prioritizing them. Based on the strategic objectives established in Practice 1, an organization identifies, analyzes, and prioritizes the set of risks that it will monitor and mitigate. This means that some risks will not be addressed, whether intentionally or accidentally. The goal of risk management is to limit exposure to the latter, but an organization can simply accept some

risks and monitor them as residual risks (e.g., a price increase for a critical purchased component). In this way, the organization knows that it has an exposure but has attempted to intelligently limit that exposure.

Risk management is a continuous process involving identifying new risks, updating the status and disposition of identified risks, determining how to handle the risks (e.g., prevent, mitigate, monitor, or accept), and implementing the selected risk-handling option. For most organizations, this includes cyber risks—and, more specifically, software vulnerabilities and malware. A large body of work by the Software Engineering Institute and the MITRE Corporation describes specific vulnerabilities and software weaknesses. In particular, MITRE has established a large resource in its Common Vulnerability and Enumeration (CVE) repository, where it makes classes of vulnerabilities and solutions available.

5. **Incident management.** Incident management is one of the disciplines that most naturally comes to mind when one considers operational resilience management. It is the end-to-end handling of a disruptive event from the time that something happens to when it is detected, triaged, and resolved. Disruptive events include deliberate or inadvertent harmful actions of people, failed internal processes, technology failures, and external events such as natural disasters and power outages.

   Implementing this practice begins before an incident occurs, when an organization plans for and assigns roles and responsibilities, including those for key stakeholders and decision makers (for escalation). Operational staff should be trained not only in delivering the services and conducting the operations for which they have responsibility but also in the results and effects to expect from performing these services and operations. Operational staff are often the first staff capable of detecting an incident; thus such training should make them more sensitive to unexpected deviations from "normal" results and effects.

   Once an incident is detected, the first step is to carefully note the circumstances of the incident, declare the incident, and preserve evidence. The organization may have prepared an immediate workaround for just such an incident. If so, that workaround is often implemented by the same staff who detect the incident. Otherwise, the organization analyzes the incident to develop an appropriate response, including recovery actions that minimize the disruption. When analyzing the incident, the incident-handling team looks for patterns or similarities to other incidents that they may have seen in the past. The organization may perform a root-cause analysis and identify and evaluate multiple candidate solutions.

   The next steps are to implement the solution—respond and recover. The incident-handling team should also ensure that the organization communicates with key stakeholders, who can provide needed resources and expertise immediately or later in the incident resolution.

   Once the incident is closed, the organization should conduct a postmortem analysis to determine if the organization should make any improvements to its overall incident management, risk management, and service delivery (operations) processes. The organization should define measures to help evaluate the effectiveness of its responses to disruptive incidents. It will analyze those measures of effectiveness to determine where to improve its practices.

6. **Service continuity.** This practice entails ensuring the continuity of essential operations and services during and following a disruptive event. Service continuity may include business continuity, disaster recovery, crisis management, and pandemic planning.

   Activities encompassed by this practice include developing service-continuity plans, assigning roles and responsibilities, and then testing plans and running exercises to ensure that the plans are robust. For example, the organization should establish plans about what to do with its workforce if it must evacuate its facility and stand up an alternative facility to continue operations. Tests and exercises can cover a wide range of activities and may include computer simulations.

   Organizations should ensure the continuity of the services they provide through careful preparation and

planning. The resilient organization tracks the location of key personnel and backup personnel, so that in the event of an incident, they can put recovery plans into action. Through exercises and drills, the organization assures that everyone knows his or her roles. When a Hurricane Sandy happens, the resilient organization does what it has rehearsed.

7. **Critical asset protection.** Critical assets (e.g., information, technology, facilities) that support high-value services must be identified, protected, and maintained. In particular, an organization must ensure that it applies adequate controls to protect the confidentiality, integrity (i.e., information security), and availability of information essential or entrusted to the business. Such controls can include maintaining an up-to-date inventory of the information that the organization must protect, on what devices that information resides, and over what networks it may be transmitted. In addition, an organization should have practices for configuring, tracking, protecting, and maintaining its IT assets (e.g., workstations, laptops, mobile devices, and network components).

   Protecting critical assets requires continually identifying and mitigating threats to the asset (e.g., as part of a comprehensive risk-management practice, discussed in Practice 4); improving, retiring, and adding new controls to the asset to maintain its integrity; and establishing appropriate identity and access management to limit access to the asset. Critical asset protection also includes facility protection, such as for an organization's IT assets, and includes facilities for backup and recovery.

8. **External-dependencies management**. An organization must identify and manage dependencies on external entities, such as its supply chain. Key elements of this practice include prioritizing external dependencies, managing risks arising from external dependencies, and formalizing relationships with external entities. Organizations should make sure that formal and contractual agreements are in place with external entities and that everyone understands what is expected from each party, in particular with respect to disruptions in delivery of critical components or services. To ensure preparedness, an organization should proactively monitor and manage the performance of external entities to make sure they meet expectations.

9. **Secure software development and integration**. Organizations must ensure that software that enables or performs the delivery of critical services and operations satisfies resilience requirements. An organization derives resilience requirements for such software in part from its resilience-management activities, including governance and program management (Practice 1), service continuity (Practice 6), and critical asset protection (Practice 7). For example, mitigating a particular threat to an asset may impose resilience (and security) requirements on the software that controls it or access to it. An organization should also elicit or collect requirements from stakeholders, including customers, end users, suppliers, other partners, and regulatory authorities. Multiple frameworks provide recommended practices for software development that address security and other resilience-related topics (see Learn More for more information).

How to derive more benefit from the recommended practices for managing operational resilience?

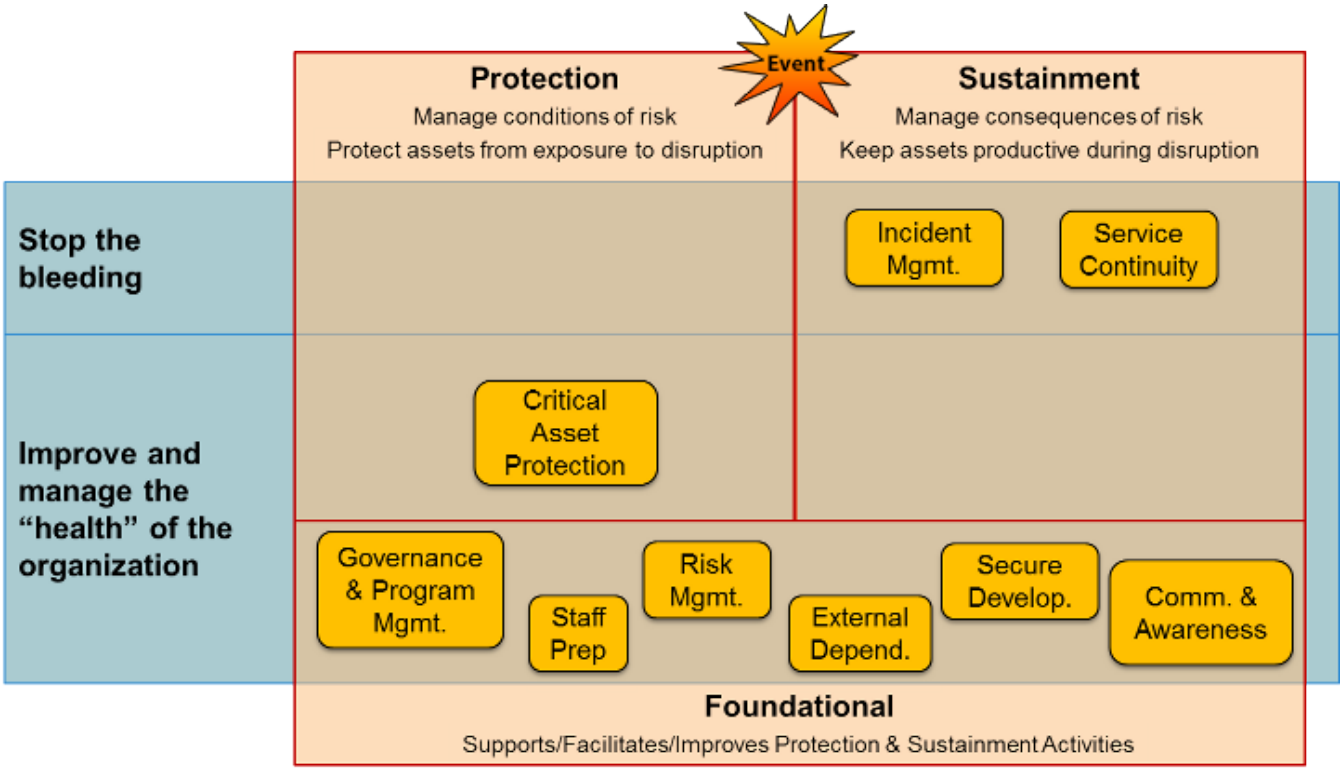
The following activities will help organizations achieve greater success when adopting the above practices for operational resilience management:

1. **Coordinate the implementation of these practices.** Implementing these practices requires competence in several disciplines (incident management, asset protection, risk management, etc.). Organizations that create a separate solution or team to deal with each practice will find their operational resilience-management activities to be inefficient and difficult to manage due to the overlaps (e.g., where do incident management, disaster recovery, and asset protection and sustainment begin or end?). Just as the implementation of each operational resilience-management practice should be driven by business objectives, so should their collective implementation. Organizations will improve their operational resilience by taking an integrated approach to implementing these activities and ensuring that there is adequate coordination among them.

Begin by gathering representatives from the different disciplines and departments to develop end-to-end scenarios that describe how the organization should respond to particular threats (as described in Practice 2). Identify which disciplines or departments (e.g., incident analysis, disaster recovery, and crisis communication) to involve at each stage of the response, including afterward, when making improvements to processes and training for service delivery, service continuity, and information security. Then determine how the organization should coordinate its activities in such scenarios. Such rehearsals or simulations help identify superior ways to implement the operational resilience-management practices.

The following diagram may help you remember the purpose of each resilience-management practice. The two practices in the "Stop the bleeding" row deal primarily with resolving incidents. The "Improve and manage" row of the diagram depicts the practices that provide infrastructural and foundational support for establishing, facilitating, measuring, and improving asset protection and operations sustainment activities. The position of those practices in the diagram also indicates their role in protecting and sustaining the health of the organization and continually improving operational resilience-management activities. The diagram illustrates the need for all the operational resilience-management practices to work together.

2. **Maintain currency with relevant standards.** In the past 10 years, standards have exploded across all disciplines in national and international efforts to deal with the growing number of cybersecurity failures. The number of standards dealing with preparedness planning has quadrupled since 2005. An organization should develop an integrated approach to updating its processes to maintain compliance with standards relevant to its business. For example, when ISO/IEC Standard 27034 Information Technology—Security Techniques—Application Security was published, its guidance affected business managers, IT managers, developers, auditors, and end users. An organization should involve designers, programmers, acquisition managers, IT staff, and users to determine what changes are needed to preserve the effectiveness of operational



resilience-management activities while addressing this standard.

3. **Understand compliance issues.** Compliance issues affect all the recommended practices. An organization must not only follow federal and state legislation and regulations but also be aware that state-by-state differences exist. For example, state requirements vary for notifications about data breaches, and this will inform the organization's communication practices. However, an organization should view compliance as an outcome of an integrated operational resilience-management program, not a goal. Simply following a rule may not be sufficient to plan for and mitigate risk; new risks arise much faster than the rate of legislation.

Food for thought. Could what happened to Target happen to your organization? What will you do in the next few days and weeks to better prepare your organization to mitigate such attacks and the disruptions they cause to your mission, services, and operations?

Learn More

For comprehensive information about operational resilience management, please see:

CERT® Program. *Resilience Management*. http://www.cert.org/resilience

MITRE Corporation. *Cybersecurity*. http://www.mitre.org/capabilities/cybersecurity/resiliency

Weick, Karl & Sutcliffe, Kathleen. *Managing the Unexpected: Resilient Performance in an Age of Uncertainty* (2nd ed.). John Wiley & Sons, 2007.

For more information about frameworks and maturity models, please see:

Allen, Julia & Mehravari, Nader. Presented at the "Buyer Beware: How to be a Better Consumer of Security Maturity Models." RSA Conference, San Francisco, CA, February 2014.

Bodeau, Deborah, Graubart, Richard, Picciotto, Jeffrey, and McQuaid, Rosalie. *Cyber Resiliency Engineering Framework* (MTR110237). MITRE, 2012. http://www.mitre.org/publications/technical-papers/cyber-resiliency-engineering-framework

Caralli, Richard A., Allen, Julia H., & White, David W. *CERT Resilience Management Model (CERT-RMM): A Maturity Model for Managing Operational Resilience*. Addison-Wesley Professional, 2011.

CMMI Product Team. *CMMI for Development, Version 1.3* (CMU/SEI-2010-TR-033). Software Engineering Institute, Carnegie Mellon University, 2010. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9661

CMMI Product Team. *CMMI for Services, Version 1.3* (CMU/SEI-2010-TR-034). Software Engineering Institute, Carnegie Mellon University, 2010. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9665

Department of Energy. *Cybersecurity Capability Maturity Model (C2M2) Program*. DoE, 2014. http://energy.gov/oe/cybersecurity-capability-maturity-model-c2m2-program

Department of Homeland Security. *Cyber Resilience Review (CRR)*. DHS, 2014. http://www.us-cert.gov/ccubedvp/self-service-crr

ISACA. *COBIT 5: A Business Framework for the Governance and Management of Enterprise IT*. ISACA, 2012: http://www.isaca.org/COBIT/Pages/default.aspx

McGraw, Gary et al. *Building Security In Maturity Model*. http://www.bsimm.com (2012).

Microsoft. Security Development Lifecycle. http://www.microsoft.com/security/sdl/default.aspx (2012).

National Institute of Standards and Technology. *Cybersecurity Framework* (Version 1). NIST, 2014. http://www.nist.gov/cyberframework/index.cfm

World Economic Forum. *Partnering for Cyber Resilience: Risk and Responsibility in a Hyperconnected World—Principles and Guidelines*. WEF, 2012. http://www3.weforum.org/docs/WEF_IT_PartneringCyberResilience_Guidelines_2012.pdf

For more information about risk management, please see:

CERT Program. *OCTAVE* and *OCTAVE Allegro*. Carnegie Mellon Software Engineering Institute, 2012 http://cert.org/resilience/products-services/octave/index.cfm

FAIR (Factor Analysis of Information Risk); http://en.wikipedia.org/wiki/Factor_analysis_of_information_risk

International Organization for Standardization/International Electrotechnical Commission (ISO/IEC). *Risk Management: Principles and Guidelines* (ISO/IEC 31000:2009). ISO, 2009.

ISO/IEC. *Information technology – Security techniques – Information security risk management* (ISO/IEC 27005:2011). ISO, 2011.

Chartered Institute of Purchasing and Supply. "Business Continuity in the Supply Chain: Embedding Resilience." Business Continuity Institute & Chartered Institute of Purchasing & Supply, 2010.

ISACA. *COBIT 5: A Business Framework for the Governance and Management of Enterprise IT*. 2014. http://www.isaca.org/COBIT/Pages/default.aspx

Joint Task Force Transformation Initiative. *Guide for Conducting Risk Assessments* (NIST Special Publication 800-30, Revision 1). National Institute of Standards and Technology, 2012. http://csrc.nist.gov/publications/nistpubs/800-30-rev1/sp800_30_r1.pdf

MITRE Corporation. *Supply Chain Risk Management*. MITRE, 2014. http://www.mitre.org/publications/systems-engineering-guide/enterprise-engineering/systems-engineering-for-mission-assurance/supply-chain-risk--management

MITRE Corporation. *Software and Supply Chain Assurance Forum*. MITRE, 2014. https://register.mitre.org/ssca

National Institute of Standards and Technology. *Managing Information Security Risk: Organization, Mission, and Information System View* (SP 800-39). NIST, 2011. http://csrc.nist.gov/publications/nistpubs/800-39/SP800-39-final.pdf

Simchi-Levi, David, Schmidt, William, & Wei, Yehua Wei. "From Superstorms to Factory Fires: Managing Unpredictable Supply-Chain Disruptions." *Harvard Business Review* (Jan./Feb. 2014).

Tomhave, Ben, Heidt Erik T., & Robins, Anne Elizabeth. *Comparing Methodologies for IT Risk Assessment and Analysis*. Gartner, 2014. https://www.gartner.com/doc/2659816/comparing-methodologies-it-risk-assessment

Wikipedia. FAIR (Factor Analysis of Information Risk). 2014. http://en.wikipedia.org/wiki/Factor_analysis_of_information_risk

For more information about external-dependencies management, please see:

CMMI Product Team. *CMMI for Acquisition, Version 1.3* (CMU/SEI-2010-TR-032). Software Engineering Institute, Carnegie Mellon University, 2010. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9657

Harrington, Lisa H., Sandor Boyson, & Thomas M. Corsi. *X-SCM : The New Science of X-treme Supply Chain Management*. New York: Routledge, 2011.

National Institute of Standards and Technology. *Security and Privacy Controls for Federal Information* (SP 800-53, Rev. 4). NIST, 2013. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf

National Institute of Standards and Technology. *Supply Chain Risk Management Practices for Federal Information Systems and Organizations* (SP 800-161). NIST, http://csrc.nist.gov/publications/drafts/800-161/sp800_161_draft.pdf

For more information about resilience engineering, please see:

Hollnagel, Erik, Paries, Jean, Woods, David, & Wreathall, John. *Resilience Engineering in Practice: A Guidebook*. Ashgate, 2013.

Mehravari, Nader & Allen, Julia H. *Cyber Risk and Resilience Management, Security and Survivability* (Podcast). CERT Program, Carnegie Mellon Software Engineering Institute, 2013.

Mehravari, Nader & Allen, Julia H. *Demand for an Integrated Approach to Better Manage Risk* (Podcast). CERT Program, Carnegie Mellon Software Engineering Institute, 2013.

Mehravari, Nader & Allen, Julia H. *Making the Case for Operational Resilience* (Podcast). CERT Program, Carnegie Mellon Software Engineering Institute, 2012.

Sheffi, Yossi. *The Resilient Enterprise: Overcoming Vulnerability for Competitive Advantage*. MIT Press, 2007.

Woods, David D., & Leveson, Nancy. *Resilience Engineering: Concepts and Precepts*. Ashgate, 2006.

For more information about getting started with operational resilience management, please see:

Bodeau, Deborah. *Jumpstart Resiliency with What You've Got*. MITRE, 2013. http://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-blog/jumpstart-resiliency-with-what-you%E2%80%99ve-got

SANS Institute. *Twenty Critical Security Controls for Effective Cyber Defense: Consensus Audit Guidelines (CAG), Version 3.1*. SANS, 2011. http://www.sans.org/critical-security-controls

For more information about resilience policy development, please see:

Department of Homeland Security. *Enabling Distributed Security in Cyberspace: Building a Healthy and Resilient Cyber Ecosystem with Automated Collective Action*. DHS, 2011.

International Organization for Standardization/International Electrotechnical Commission (ISO/IEC). *Information technology — Security Techniques — Information Security Management Systems — Requirements* (ISO/IEC 27001:2005(E)). ISO, 2005.

ISO/IEC. *Information Technology — Security Techniques — Code of Practice for Information Security Management* (ISO/IEC 27002:2005(E)). ISO, 2005.

Keogh, Miles & Cody, Christina. *Resilience in Regulated Utilities*. National Association of Regulatory Utility Commissioners. http://www.naruc.org/Grants/Documents/Resilience%20in%20Regulated%20Utilities%20ONLINE%2011_12.pdf

Mehravari, Nader. "Effects of Recent Federal Policies on Security and Resiliency Landscapes," RSA Conference, San Francisco, CA, February 2014.

Motef, John D. *Critical Infrastructure Resilience: The Evolution of Policy and Programs and Issues for Congress*. Congressional Research Service, August 2012. http://www.fas.org/sgp/crs/homesec/R42683.pdf

Open Web Application Security Project (OWASP). https://www.owasp.org/index.php/Main_Page

World Economic Forum. *Risk and Responsibility in a Hyperconnected World—Pathways to Global Cyber Resilience*. WEF, 2012.

About the Author(s)

Dr. Nader Mehravari is with the CERT® Division of the Software Engineering Institute (SEI) at the Carnegie Mellon University. His current areas of interest and research include operational resilience, cybersecurity and resilience management, protection and sustainment of critical infrastructure, preparedness planning, and associated risk management principles and practices.

Pamela Curtis is a Senior Researcher on the Resilient Enterprise Management Team in the CERT Program at the Software Engineering Institute. Curtis conducts analytical studies and investigations and develops models and assessments related to improving and measuring operational resilience. She has over 25 years of experience in the information technology domain as a systems analyst, programmer, process improvement team leader, technical communicator, and manager. Curtis holds a BA with a concentration in Management from Simmons College and an MS in Management Information Systems from Boston University.

Julia Allen is a senior member of the technical staff within the CERT Program at the Software Engineering Institute (SEI), a unit of Carnegie Mellon University in Pittsburgh, PA. Allen is engaged in developing and transitioning executive outreach programs in enterprise security and governance, as well as conducting research in software security and assurance. Prior to this technical assignment, Ms. Allen served as acting Director of the SEI for an interim period of six months, as well as Deputy Director/Chief Operating Officer for three years. Before joining the SEI, she was a vice president in embedded systems software development for Science Applications International Corporation, and managed large software development programs for TRW (now Northrop Grumman).

Article Submission Policy

The CSIAC Journal is a quarterly journal focusing on scientific and technical research & development, methods and processes, policies and standards, security, reliability, quality, and lessons learned case histories. CSIAC accepts articles submitted by the professional community for consideration. CSIAC will review articles and assist candidate authors in creating the final draft if the article is selected for publication. However, we cannot guarantee publication within a fixed time frame. *Note that CSIAC does not pay for articles published.*

AUTHOR BIOSAND CONTACT INFORMATION

When you submit your article to CSIAC, you also need to submit a brief bio, which is printed at the end of your article. Additionally, CSIAC requests that you provide contact information (email and/or phone and/or web address), which is also published with your article so that readers may follow up with you. You also need to send CSIAC your preferred mailing address for receipt of the Journal in printed format. All authors receive 5 complementary copies of the Journal issue in which their article appears and are automatically registered to receive future issues of Journal

COPYRIGHT:

Submittal of an original and previously unpublished article constitutes a transfer of ownership for First Publication Rights for a period of ninety days following publication. After this ninety day period full copyright ownership returns to the author. CSIAC always grants permission to reprint or distribute the article once published, as long as attribution is provided for CSIAC as the publisher and the Journal issue in which the article appeared is cited. The primary reason for CSIAC holding the copyright is to insure that the same article is not published simultaneously in other trade journals. The Journal enjoys a reputation of outstanding quality and value. We distribute the Journal to more than 30,000 registered CSIAC patrons free of charge and we publish it on our website where thousands of viewers read the articles each week.

FOR INVITED AUTHORS:

CSIAC typically allocates the author one month to prepare an initial draft. Then, upon receipt of an initial draft, CSIAC reviews the article and works with the author to create a final draft; we allow 2 to 3 weeks for this process. CSIAC expects to have a final draft of the article ready for publication no later than 2 months after the author accepts our initial invitation.

PREFERRED FORMATS:

- Articles must be submitted electronically.
- MS-Word, or Open Office equivalent (something that can be edited by CSIAC)

SIZE GUIDELINES:

- Minimum of 1,500 – 2,000 words (3-4 typed pages using Times New Roman 12 pt font) Maximum of 12 pages
- Authors have latitude to adjust the size as necessary to communicate their message

IMAGES:

- Graphics and Images are encouraged.
- Print quality, 300 or better DPI. JPG or PNG format preferred

Note: Please embed the graphic images into your article to clarify where they should go but send the graphics as separate files when you submit the final draft of the article. This makes it easier should the graphics need to be changed or resized.

CONTACT INFORMATION:

CSIAC
100 Seymour Road Suite C102
Utica, NY 13502
Phone: (800) 214-7921
Fax: 315-351-4209

Michael Weir, CSIAC Director
John Dingman, Managing Editor
Email: info@csiac.org

CSIAC is a DoD sponsored Information Analysis Center (IAC), administratively managed by the Defense Technical Information Center (DTIC), technically managed by the Air Force Research Laboratory (AFRL) in Rome, NY and operated by Quanterion Solutions Incorporated, Utica, NY.

ABOUT THE JOURNAL OF CYBER SECURITY AND INFORMATION SYSTEMS

CSIAC JOURNAL EDITORIAL BOARD

Distribution Statement
Unclassified and Unlimited

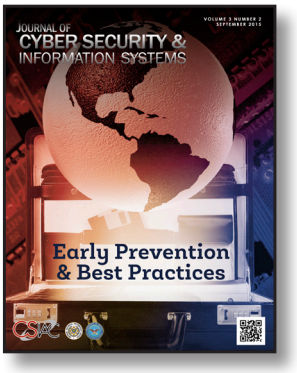
CSIAC
100 Seymour Road
Utica, NY 13502-1348
Phone: 800-214-7921 • **Fax:** 315-732-3261
E-mail: info@csiac.org
URL: https://www.csiac.org/

ABOUT THIS PUBLICATION

The Journal of Cyber Security and Information Systems is published quarterly by the Cyber Security and Information Systems Information Analysis Center (CSIAC). The CSIAC is a DoD sponsored Information Analysis Center (IAC), administratively managed by the Defense Technical Information Center (DTIC). The CSIAC is technically managed by Air Force Research Laboratory in Rome, NY and operated by Quanterion Solutions Incorporated in Utica, NY.

COVER DESIGN

Shelley Howard
Graphic Designer
Quanterion Solutions, CSIAC



ARTICLE REPRODUCTION

Images and information presented in these articles may be reproduced as long as the following message is noted:

"This article was originally published in the Journal of Cyber Security and Information Systems Vol.III, No II"

In addition to this print message, we ask that you notify CSIAC regarding any document that references any article appearing in the *CSIAC Journal.*

Requests for copies of the referenced journal may be submitted to the following address:

Cyber Security and Information Systems
100 Seymour Road
Utica, NY 13502-1348

Phone: 800-214-7921
Fax: 315-732-3261
E-mail: info@csiac.org

An archive of past newsletters is available at **https://journal.csiac.org.**

Cyber Security and Information Systems Information Analysis Center
100 Seymour Road
Suite C-102
Utica, NY 13502

Return Service Requested

Journal of Cyber Security and Information Systems – September 2015
Early Prevention & Best Practices

— IN THIS ISSUE —

Increasing Assurance Levels Through Early Verification with Type Safety
By Rick Murphy 2

Looking at M&S Education Through the Prism of the Video Game Industry
By John Lawson III 12

Agile at Scale (AAS)
By Robert L. Nord, Ipek Ozkaya - Software Engineering Institute 15

Managing Operational Resilience
By Julia H. Allen, Pamela Curtis, Nader Mehravari - Software Engineering Institute 21