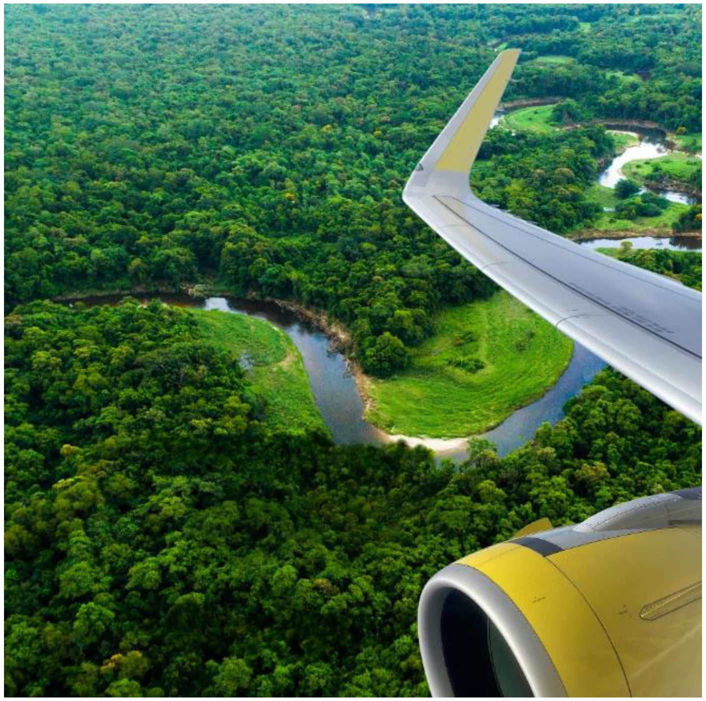
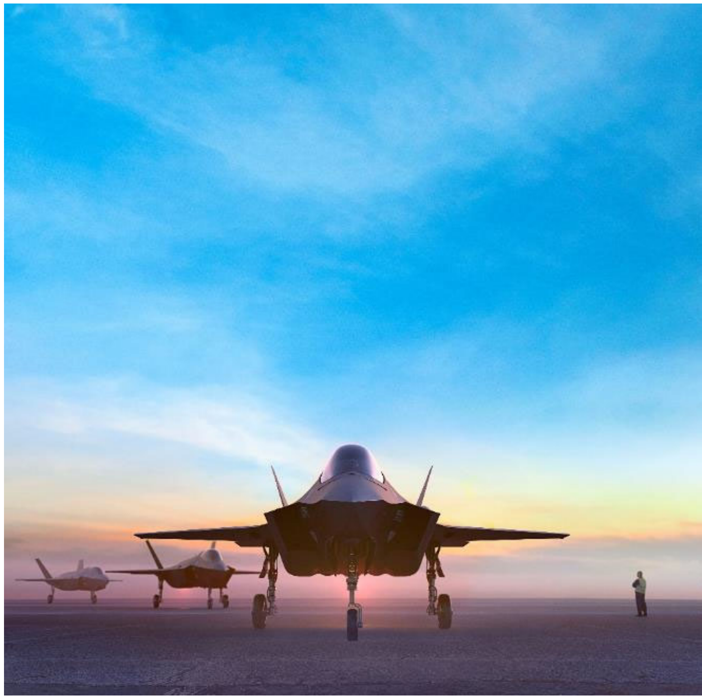




How Agile Enables FOSS Usage and Modernization

Kurt Mohr
Senior Fellow – Software Engineering

18 October 2023

Approved for Public Release

Disclaimer

The views and opinions expressed in the presentation are that of the author and do not necessarily reflect the views of the author's employer or other associated parties.

Always follow your company policies regarding the use of any third party software including Free and Open Source Software (FOSS).

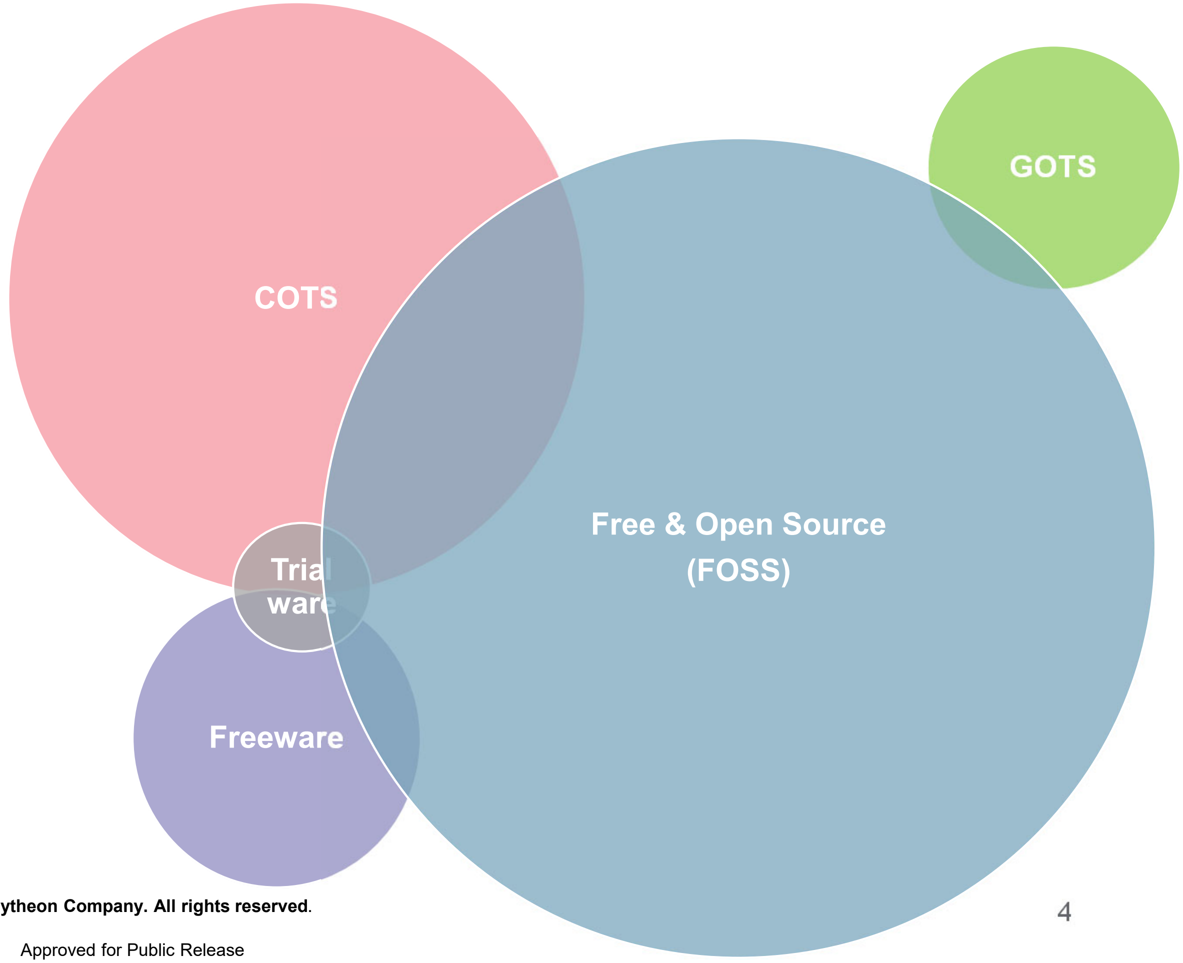
Agenda

- FOSS definition
- The value of FOSS in development
- FOSS risks in non-agile environments
- How Agile can increase FOSS acceptance
- Integrating FOSS into the Agile lifecycle
- Agile methodology differences in FOSS enablement
- Real world examples of the need for FOSS speed
- FOSS evolution timeframe challenges
- Recommendations and Best Practices

What is FOSS

COTS	Commercial Off-The-Shelf software is Software and hardware that already exists and is available from commercial sources. It is also referred to as off-the-shelf. This can include commercial versions of FOSS packages.
GOTS	Government Off-The-Shelf software is Software and hardware that already exists and is available from Government sources.
Free	"Free" refers to software that users typically have the freedom to run, copy, distribute, study, change and improve the software.
Open Source	"Open Source" refers to software with its source code publicly available. However, this does not automatically mean the software can be used for any purpose. Generally, a license is associated with the software that defines the run rules of usage. See the broadly accepted Open Source Initiative for more details. "Open Source" refers to the ability to develop in peer-to-peer community. Open Source in this context does not necessarily mean that anyone can use the software for any purpose without permission by the software owner, as many times FOSS requires a license to download and use the software. Open Source software will frequently have license terms that require the licensee to grant back certain rights to the open source community.
Freeware	"Freeware" refers to software available at no charge but not distributed as Open Source. Freeware often comes with license restrictions that forbid or restrict sharing. There is no agreed-upon set of rights or licenses, and every publisher defines its own rules for freeware. Note: This excludes commercially available software provided free for a limited time.
Trialware	Commercially available software that is provided free of charge for a limited time or with limited functionality.

Scale of software in use in some domains



Approved for Public Release

Value of FOSS in Development

- Reduction in custom code
  - FOSS capabilities and features can now be considered commodity
  - Interfaces to FOSS are generally available as reusable low code/no code
    - E.g. Kafka connectors
- Transferable knowledge
  - Mature tooling likely to be available on multiple programs and companies
    - E.g. DevSecOps tool chains can be common
  - Documentation and best practices are in abundance from public sources
- Defect introduction and correction timeframe
  - Most impactful issues addressed by the community
  - Source code for proposed changes can be available even prior to release build
- Increased development speed
  - Availability of example interface code can reduce trial and error
  - Lessons learned from others using FOSS limits time spent taking bad paths

Value of FOSS in Operations

Coverity report shows open source Can have significantly fewer defects

https://www.wired.com/2013/05/coverity-report

- System stability
  - Number of users of the FOSS package can show it is stable
  - Stable versions are labelled so users can choose between that or the latest
- Stable well documented interfaces
  - Most FOSS uses some sort of Semantic Versioning to indicate interface changes
  - Disruptive changes are generally limited because of the large user base
  - Documentation on interfaces is plentiful
- Code efficiency
  - Community peer reviews identify inefficiencies, improving operational effectiveness
  - Large user base locates and rectifies performance issues quickly, reducing resource requirements
- Reduced time to market
  - Less development means less time
  - Stable FOSS doesn't need to be tested by you every system release for every feature
    - You don't need to test a commodity, only that the system is using the commodity (e.g. test drive the car, not each tire)

Apache NiFi can be scaled to process billions of messages a second

Processing one billion events per second with NiFi - Cloudera Blog

Raytheon
An RTX Business

FOSS Value Depends on Organizational Agility

- The speed of FOSS can be no greater than the speed of your organization
  - How many people have to approve a change?
  - How much bureaucracy will there be

- Making changes and adapting to emerging FOSS is important
  - Do you have to know everything at the time of the proposal?
  - What if there is a newer FOSS package that may be better?

- Metrics can be misleading
  - Are you locked in to Earned Value so experimenting could ‘look’ like a problem?
  - Is up front cost of integration of FOSS going to disrupt the Integrated Master Schedule?

The benefits are clear, but can you really achieve them?

FOSS can be great, can we get the benefits?

- Reduction in custom code
  - FOSS capabilities and fea commodity
  - Interfaces to FOSS are generally code/no code

Depends on how fast you can get FOSS integrated

- Transferable knowledge
  - Mature tooling likely to be companies
  - Documentation and best practic sources

How recent is the version? Are we working on 'old' FOSS?

- Defect introduction and correction timeframe
  - Most impactful issues ad
  - Source code for proposed chan to release build

As the FOSS community fixes issues, how fast can you get them in house?

- Increased development speed
  - Availability of example interface
  - Lessons learned from others us bad paths

Lessons learned may be to update to the latest version… can you?

- System stability
  - Number of users o e
  - Stable versions are label that or the latest

An old version can be stable, but may have cyber or other risks

- Stable well documented interfaces
  - Most FOSS use some s indicate interface ch
  - Disruptive changes are g rge user base
  - Documentation on interfa

Documentation is plentiful but if you don't update regularly, jumping multiple major versions can be risky

- Code efficiency
  - Community peer review
  - Large user base locates ar quickly

If you can't or don't update regularly, your version could have inefficiencies

- Reduced time to market
  - Less developme means l
  - Stable FOSS doesn't n release for every feature
    - You don't need to test a using the commodity

Not taking advantage of new features in a timely manner can mean implementing yourself and then backing it out later

Raytheon
An RTX Business

Approved for Public Release

FOSS Integration in Non-Agile Environments

- Considerations for non-Agile environments
  - When is the FOSS selected?
  - What happens when FOSS capabilities change?
  - Is there schedule available to patch and upgrade?
  - How do you respond to security findings in the FOSS community?

- How stringent are the requirements and design?
  - Can you make changes after Critical Design Review?
  - If a new FOSS package comes available, are you able to take advantage?
  - Are you able to prototype to burn down risk?

FOSS in non-Agile is possible, but many questions need to be answered

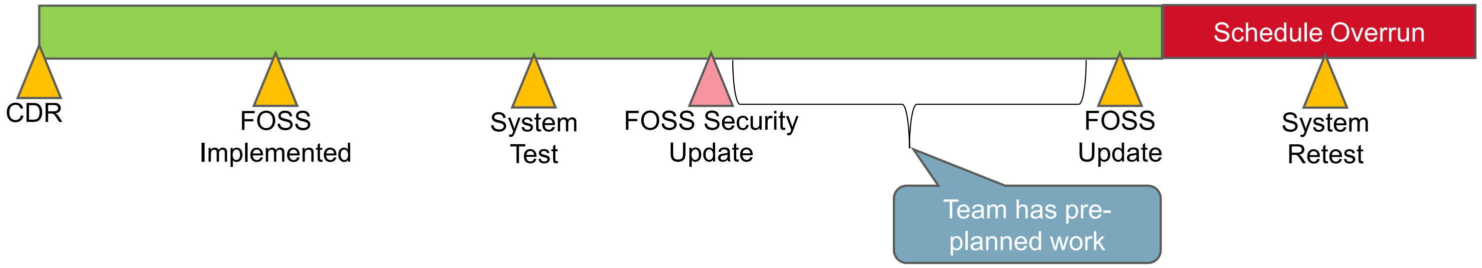
FOSS Integration in Non-Agile Environments

- Risks in Waterfall FOSS usage
  - FOSS packages could be implemented in development schedule but never updated
  - Duration it takes from security finding to system update could be long
  - Configuration of the FOSS could be evolving as the team learns
    - Could be cost and schedule implications
  - Inability to take advantage of new features quickly
- Design can't evolve as new FOSS packages become available
  - Stuck at a moment in time of the Critical Design Review
  - Prototyping and experimentation could be limited
  - We have to assume the initial proposed architecture is accurate

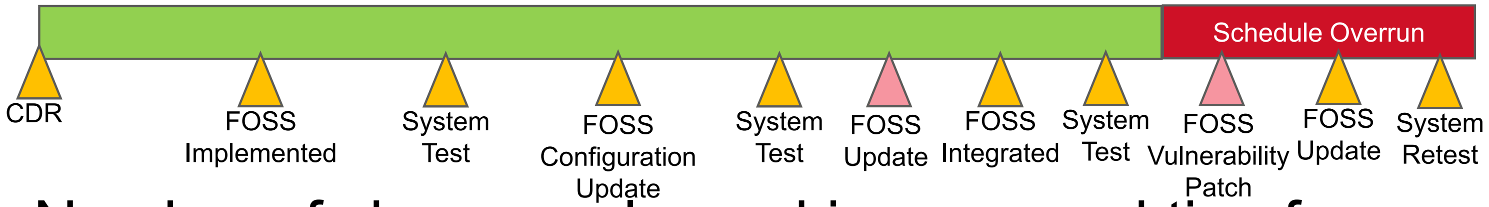
There are risks in Waterfall as well as other development methodologies

FOSS Updates in Waterfall

- Notional update timeline

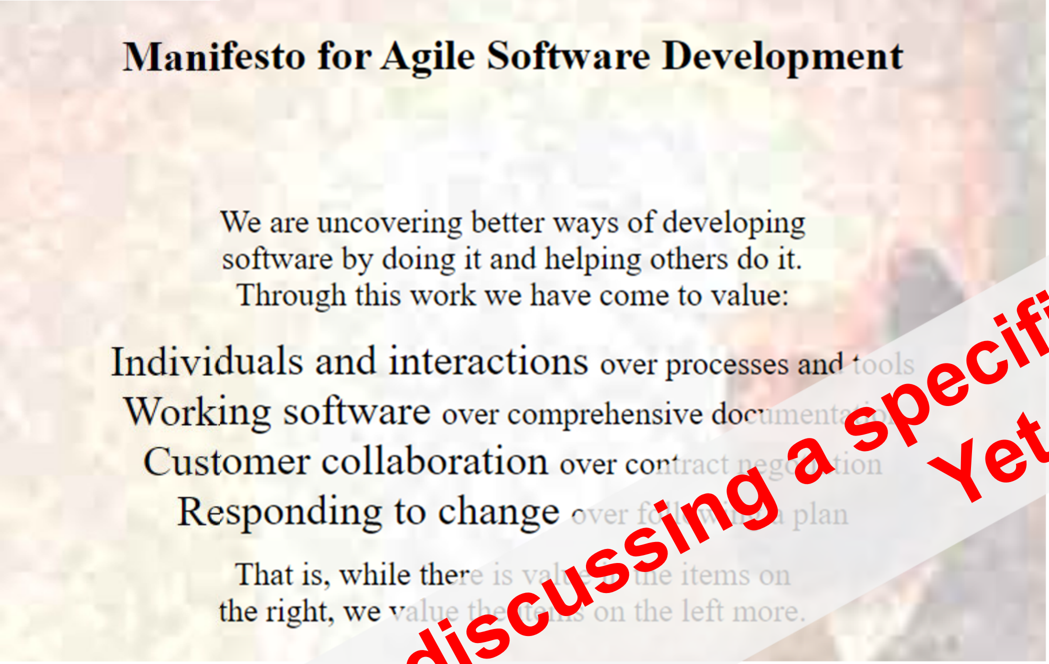


- Test cadence – How often are we running tests?



- Number of changes planned in proposed timeframe
  - Proposal: Initial integration + 6 month patching cycle
  - Reality: Initial integration + security updates as they are located + feature updates quarterly that could improve the system

What is “Agile” within this Context?



Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

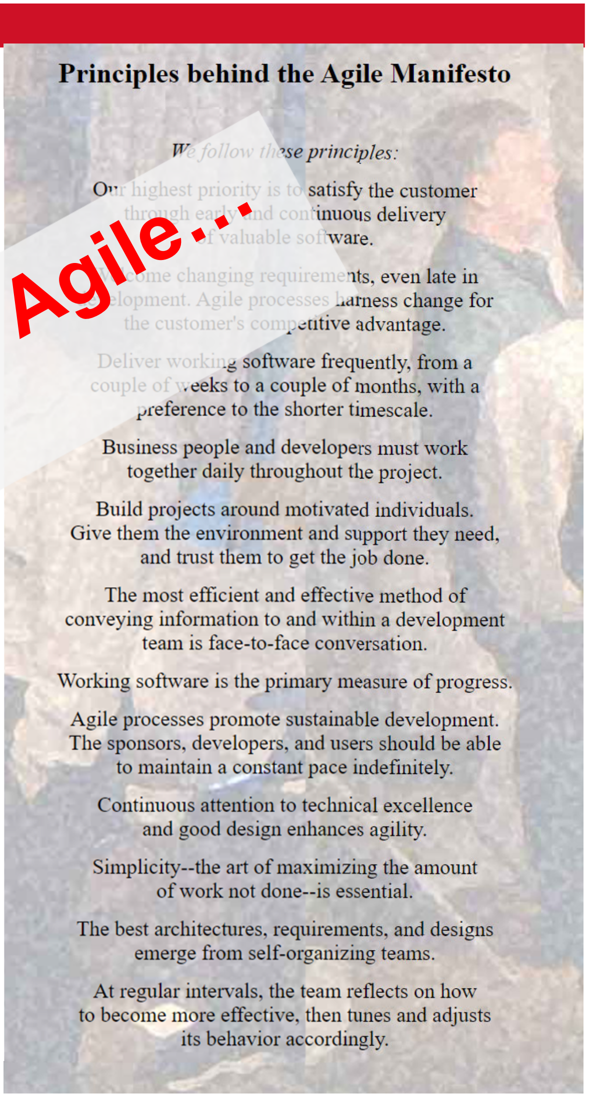
Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

- Working software is the core criteria
- Principles are customer and delivery focused
- Technical excellence and good design
- Simplicity and constant improvement

Not discussing a specific type of Agile… Yet

https://agilemanifesto.org/



Principles behind the Agile Manifesto

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Enables FOSS

- Agile allows updates to be prioritized based on need and value
  - Backlog prioritization
  - Customer awareness
  - Architectural value

- Not all updates are equal
  - Security updates: What is your exposure to the threat that was fixed
  - Feature additions: Adding features when you need them, not when they are available
  - Bug fixes: If the bug is visible within the system context

Agile allows dynamic contextual responsiveness to FOSS updates

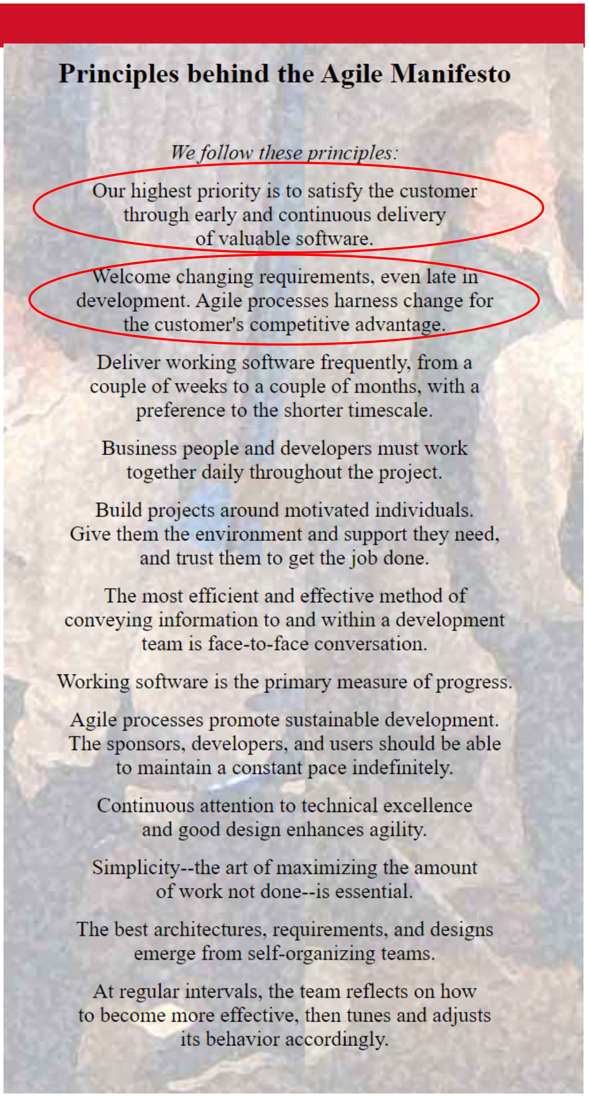
Agile Enables FOSS

- Obsolescence
  - Version obsolescence: Replacing outdated versions that are end of life
    - Can be managed more effectively in Agile with fewer disruptions
  - FOSS obsolescence: Removing unsupported or unused FOSS
    - Taking components out of a system can be integrated into an Agile cadence
    - Allows management of risk against resources and schedule
- Replacement
  - Newer or better FOSS products can be planned for integration
  - Side by side comparison within a Sprint allows for informed decisions
  - Opportunities to run multiple FOSS options in parallel for incremental transformation

The world is not static, Agile lets us roll with the FOSS flow

FOSS Enables Agile

- Not only does Agile enable FOSS, but FOSS supports the Agile manifesto
- Early and Continuous Value
  - FOSS features are well documented
  - Enabling a feature can be implementing an interface to the FOSS
- Changing Requirements
  - New FOSS can be introduced to fulfill changing requirements
  - Changes in requirements can be evaluated against FOSS capabilities



Principles behind the Agile Manifesto

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

https://agilemanifesto.org/

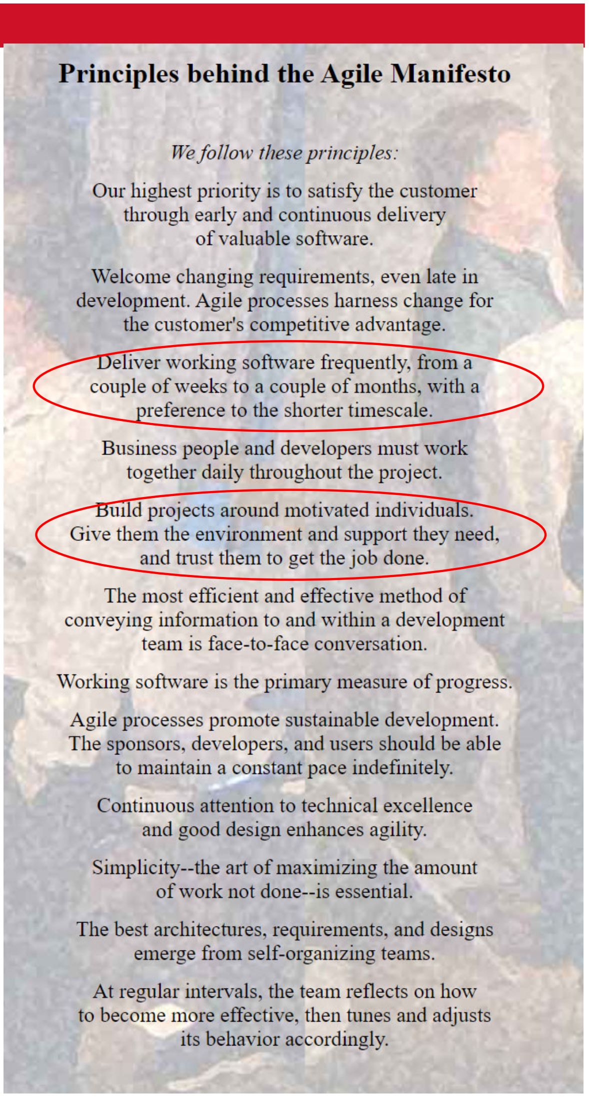
Raytheon
An RTX Business

Approved for Public Release

FOSS Enables Agile

- Frequent Updates
  - FOSS installation and configuration is generally trivial
  - Updating versions can be done within a sprint

- Providing a Stable Environment
  - Development environment FOSS is widely used (e.g. Integrated Development Environment)
  - FOSS for testing can increase speed of adoption (e.g. Selenium, Cucumber)
  - Documentation and example code are available for low ramp up time



Principles behind the Agile Manifesto

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

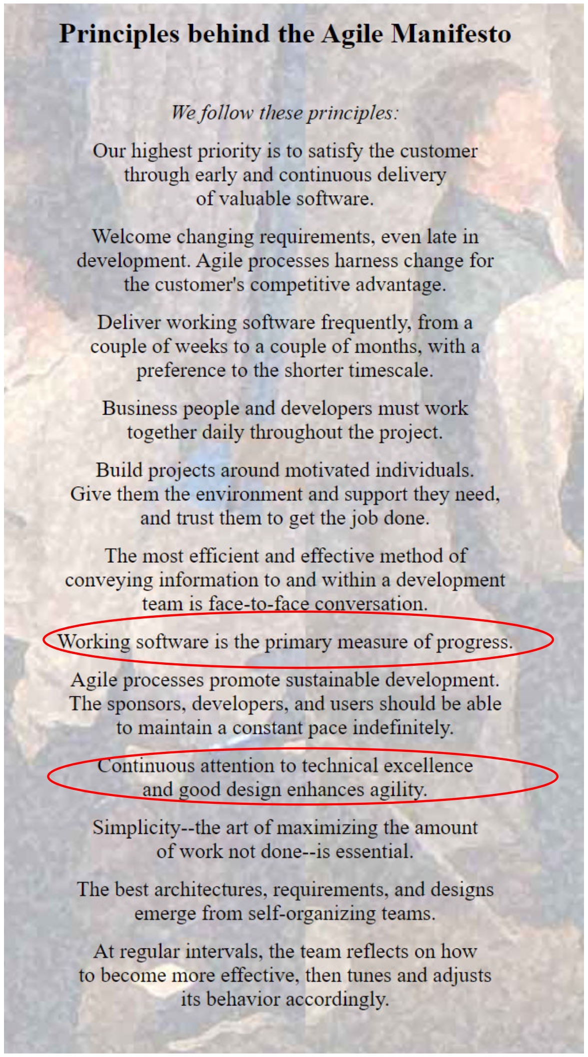
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

https://agilemanifesto.org/

FOSS Enables Agile

- Working Software
  - FOSS software is well tested and widely used
  - Low risk of getting a software package that doesn't work

- Technical Excellence
  - FOSS provides clear interfaces for integration
  - Architectures and designs become less complicated and more "low code" integration



Principles behind the Agile Manifesto

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

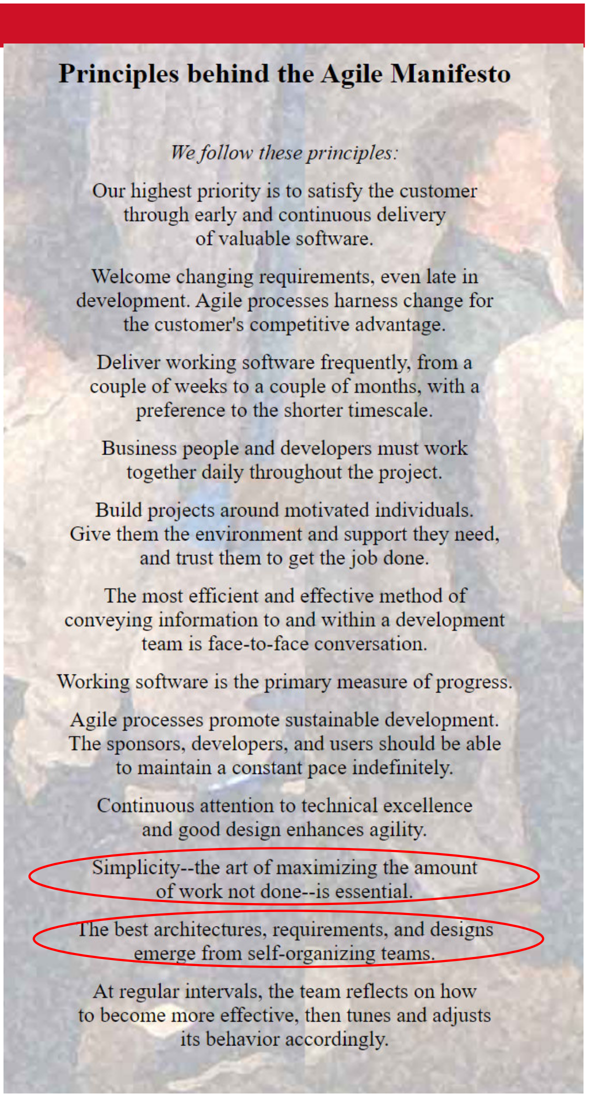
The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

https://agilemanifesto.org/

FOSS Enables Agile

- Simplicity
  - Existing documentation means no guess work on what it will do
  - APIs and connection methods generally readily available
  - Stringing FOSS together in known patterns can reduce custom code volume
- Emergent Architectures
  - Experimentation and trial and error can be performed to find the "best" FOSS for the situation
  - Side-by-side comparison removes paper trade studies and guess work



Principles behind the Agile Manifesto

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

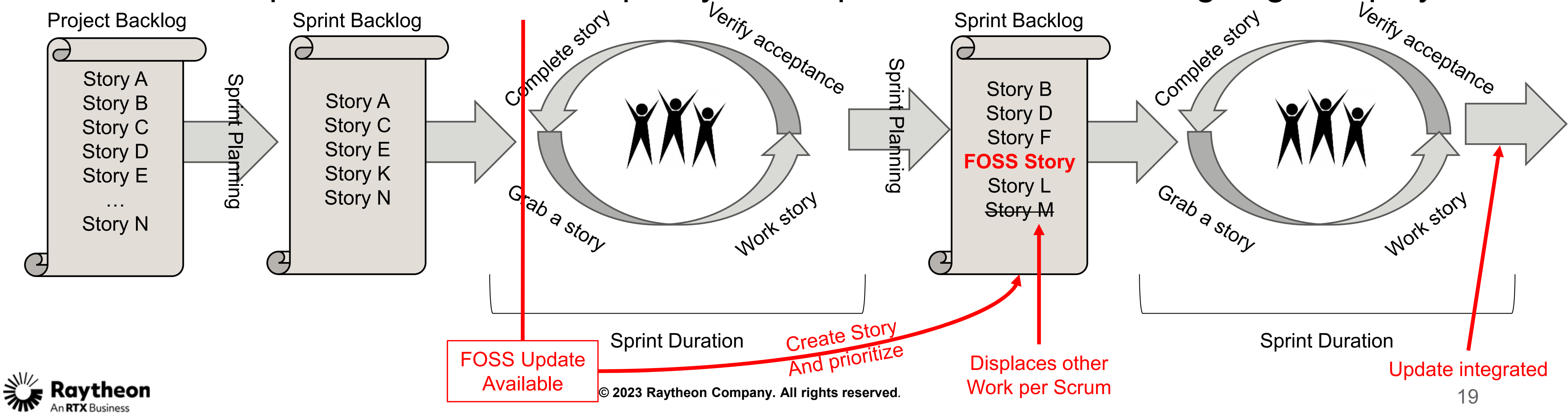
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

https://agilemanifesto.org/

Approved for Public Release

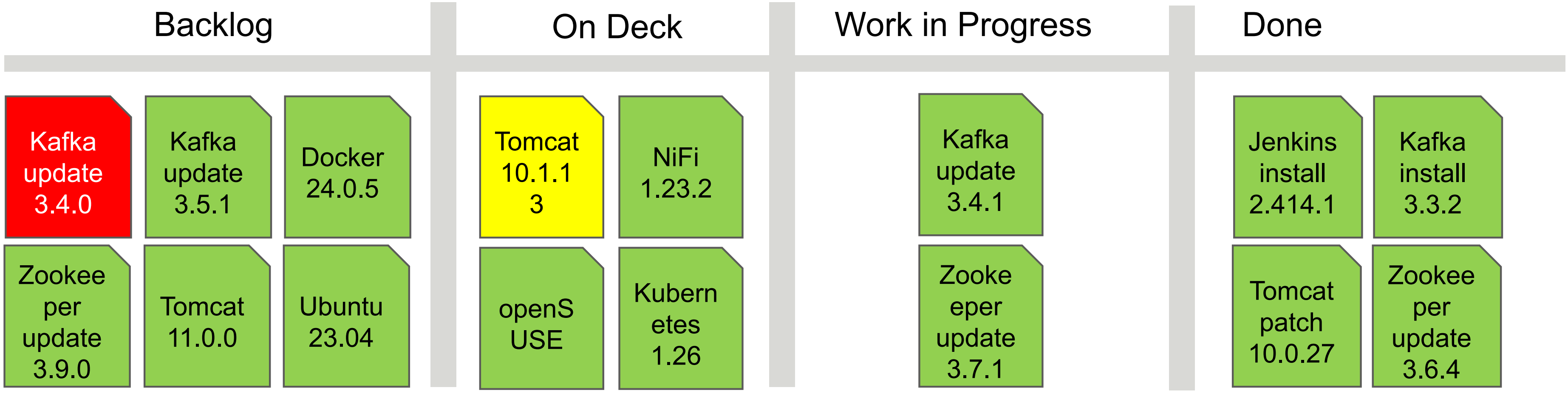
Not all Agile are the Same

- Scrum
  - Adding items to the backlog assumes knowledge of releases
  - Sprint duration drives the responsiveness
    - Assume 2 week sprint, FOSS update comes out after sprint start so could be 4 weeks to release update
  - Relative priorities and team capacity can impact if and when changes get deployed



Not all Agile are the Same

- Kanban
  - Maintaining velocity and limiting Work in Progress requires FOSS prioritization
  - Prioritizing each FOSS addition and update independently
    - No set cadence as with other approaches
  - Some updates may be ‘skipped’

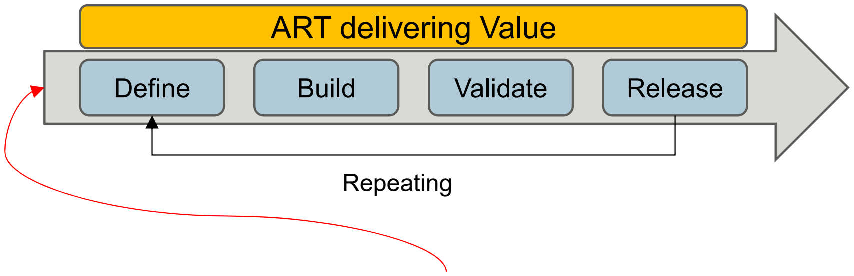


Backlog	On Deck	Work in Progress	Done
🟥 Kafka update 3.4.0	🟨 Tomcat 10.1.13	🟩 Kafka update 3.4.1	🟩 Jenkins install 2.414.1
🟩 Kafka update 3.5.1	🟩 NiFi 1.23.2	🟩 Zookeeper update 3.7.1	🟩 Kafka install 3.3.2
🟩 Docker 24.0.5	🟩 openSUSE		🟩 Tomcat patch 10.0.27
🟩 Zookeeper update 3.9.0	🟩 Kubernetes 1.26		🟩 Zookeeper update 3.6.4
🟩 Tomcat 11.0.0			
🟩 Ubuntu 23.04			

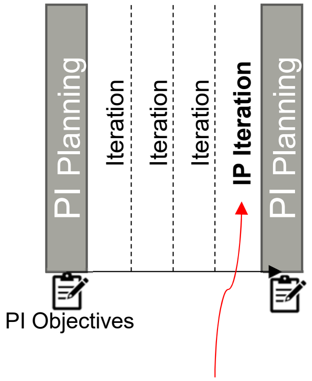
Approved for Public Release

Not all Agile are the Same

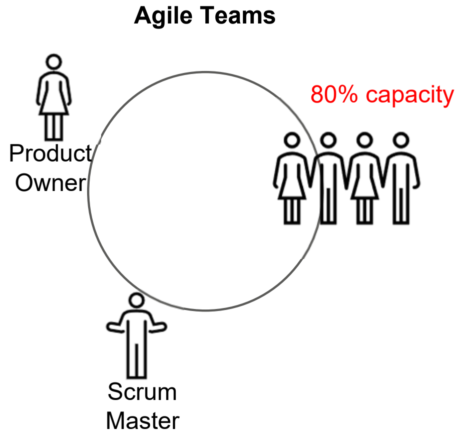
- Scaled Agile Framework (SAFe)
  - Balances intentional and emergent designs with focus on business value
  - Establishes Program Increments that funnel content into release trains
    - Uses Innovation and Planning iteration for new work
  - States that personnel should not be planned at 100% capacity to allow adaptability
  - Supports SAFe Scrum and Kanban teams as part of the Release Train



Known FOSS updates can be inserted in the ART



FOSS innovation can be performed in the IP iteration

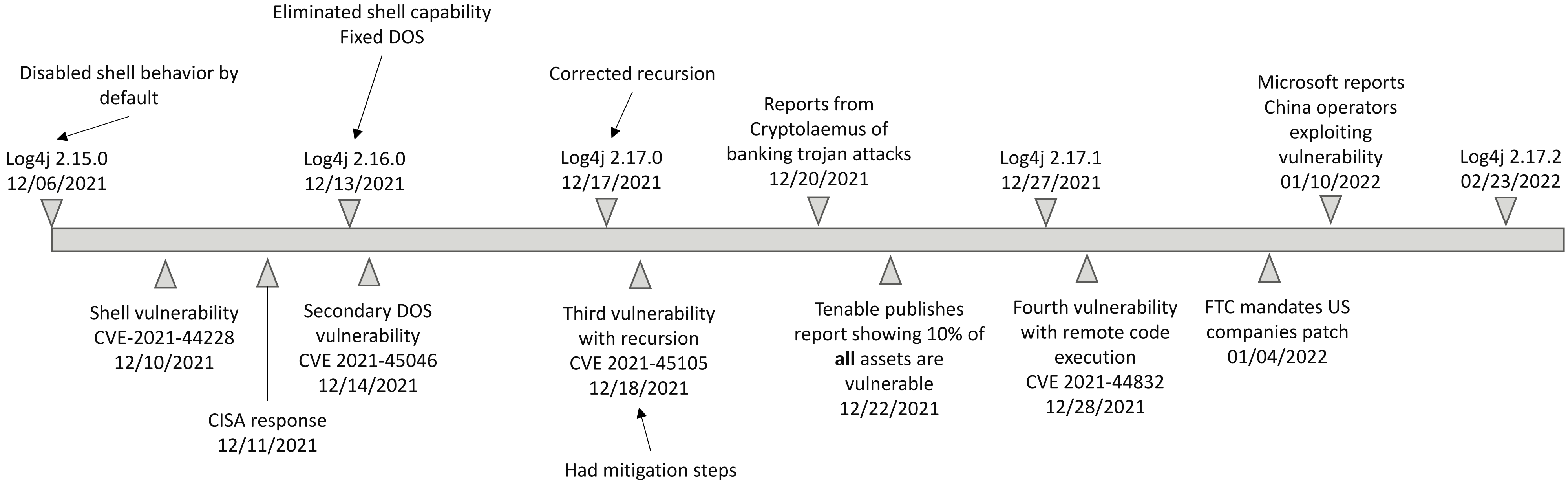


Critical FOSS updates may be possible in ‘spare’ capacity

Real World FOSS – the Need for Speed

- The Log4j example
  - How fast things change
  - Bad guys get vulnerability notices too!

Note: Previous update was
03/06/2021
Log4j 2.14.1



Disabled shell behavior by default

Log4j 2.15.0
12/06/2021

Shell vulnerability
CVE-2021-44228
12/10/2021

CISA response
12/11/2021

Eliminated shell capability
Fixed DOS

Log4j 2.16.0
12/13/2021

Secondary DOS vulnerability
CVE 2021-45046
12/14/2021

Corrected recursion

Log4j 2.17.0
12/17/2021

Third vulnerability with recursion
CVE 2021-45105
12/18/2021

Had mitigation steps

Reports from Cryptolaemus of banking trojan attacks
12/20/2021

Tenable publishes report showing 10% of **all** assets are vulnerable
12/22/2021

Log4j 2.17.1
12/27/2021

Fourth vulnerability with remote code execution
CVE 2021-44832
12/28/2021

FTC mandates US companies patch
01/04/2022

Microsoft reports China operators exploiting vulnerability
01/10/2022

Log4j 2.17.2
02/23/2022

Could you respond fast enough? What does it do to other planned work?

*Data from https://nvd.nist.gov/

Real World FOSS – Other Examples

- OpenSSL
  - Vulnerability CVE-2014-0160 that allowed an attacker to read from the memory of servers and clients running vulnerable versions of the software
- Bash shell
  - CVE-2014-6271 allowed attackers to execute arbitrary code on a system
- Apache Struts
  - Multiple high profile vulnerabilities identified including CVE-2017-5638 for remote command injection
- Libssh
  - CVE-2018-10933 allowed attackers to bypass authentication including on Cisco routers

There are many more, but this shows they can be very significant

*Data from https://nvd.nist.gov/

FOSS Evolution – a Timeline

- The rate of change for FOSS packages depends on several factors
  - Maturity of FOSS
  - Adoption rates
  - Contribution rates
  - Vulnerabilities

Rate of change can be a rollercoaster

- You can plan for a cadence, but can't rely on it
  - Sometimes there is a fast and furious update cycle
  - New features added can result in a series of patches

The best plans can fall apart

Different FOSS packages update on different timelines

FOSS Evolution - Exemplars

- Apache NiFi
  - 2023 – 6 updates as of this research
  - 2022 – 9 updates, 1 a month with some gaps
  - 2021 – 7 updates, 2 months with 2 releases, remaining releases several months apart
  - 2020 – 7 updates, 5 of which were in the first 3 months

- Kubernetes
  - Releases happen “approximately” 3 times a year
  - Several instances where there are 5+ in a year

FOSS isn’t updated on your schedule, it is updated on the contributors

Recommendations and Best Practices

- Planning for change and evolution
  - Assume constant change and new technologies
  - Reserve capacity for modernization and experimentation
- Establish an update cadence
  - Plan to update FOSS packages on a regular basis
  - Updating patch versions is lower impact than waiting and moving multiple versions concurrently
  - If there isn’t an update, you can pull other work in
- Monitor boards
  - Security vulnerabilities, FOSS updates, community forums
  - Proactive is easier than reactive

Have a plan and realize it will change

A cadence will help with ‘normal’ updates

Plan for change and you won’t be disappointed!

Recommendations and Best Practices

- Avoid static thinking
  - Know that what was bid/proposed may be outdated by the time you implement
  - Evolving/emergent architecture isn’t a bad thing
- Learn from others
  - Knowledge from people who have used the FOSS before can reduce ramp up
  - External and internal “common configurations” and “lessons learned” can improve reuse
- Encouraging experimentation and failures
  - Buying down risk and failing fast helps ensure the right solution is found
  - Experimenting with new or replacement technologies can help articulate the value

Trial and error can be invaluable

Thank You

Speaker Bio

Kurt Mohr is a Senior Engineering Fellow with over 20 years in Software Engineering and Architecture spanning both commercial and defense. Kurt is the Software Technical Director for Raytheon Common Engineering.

He has a strong background in multiple Agile development methodologies including Scrum, Kanban, and Scaled Agile. Kurt has architectural experience with multiple forms of software systems including cloud, bare metal, and embedded as well as highly scalable distributed computing. Kurt holds multiple industry certifications in addition to his formal degree pursuits in Aeronautical Computer Science, Engineering Management, and Organizational Management