CEC Continuing Education Center

DesignNews

Best Practices for Designing Real-Time Embedded Systems

DAY 5 : The Best Practices Lightning Round

Sponsored by

Digi-Key ELECTRONICS

informa markets

Webinar Logistics

- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Attendee Chat’ by maximizing the chat widget in your dock.

Continuing Education Center

Sponsored By

Course Sessions

- System Level Design Philosophy
- Designing a Hardware-less System
- It’s All About the Data
- Testing Your Way to Design Success
- The Best Practices Lightning Round

CEC Continuing Education Center

Sponsored By

1

Best Practices for SDLC

The Software Development Life Cycle (SDLC) defines the processes, standards and best practices used to develop and maintain a software system.

BENINGO EMBEDDED GROUP

Best Practices for SDLC

1. Periodically audit your SDLC and adjust based on your current needs. (Tune it!)
2. Use a tool like JAMA to record your requirements and use cases in a single place.
3. Use software features to plan sprints (Feature-based development).
4. Select the development model that best meets your needs and customers.
5. Use a project management tool to plan, track and monitor progress.
6. Don't go overboard! Build out only as much SDLC process as you need to be successful.
7. Automate and templatize as much as you possibly can. (It saves time in the long run).
8. Setup and leverage continuous integration and continuous deployment processes.
9. Ensure that your SDLC is traceable from requirements through testing.
10. Define your metrics and measure them throughout the development cycle.

CEC Continuing Education Center

Sponsored By

2

Best Practices for Software Design - Architecture

A properly architected software system will not only provide flexibility and scalability, it will also help minimize the effort and resources needed to implement and maintain the software.

Best Practices for Software Design - Architecture

Architecture is the decomposition of a system into its core components, the arrangement of those components and how they interact with each other.

A good architecture:

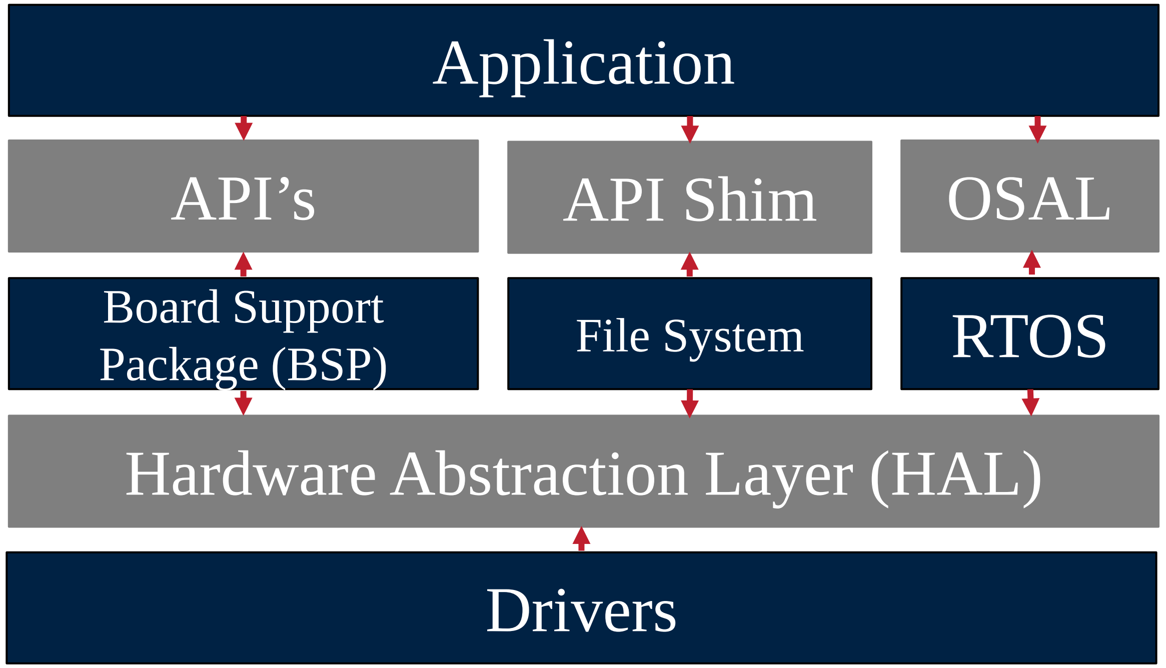
- Is understandable
- "Easy" to implement
- Maintainable
- Deployable
- Scalable
- Flexible
- Abstracted
- Hardware Independent

An architecture should:

- Minimize the systems lifetime costs
- Improve programmer productivity

Best Practices for Software Design - Architecture

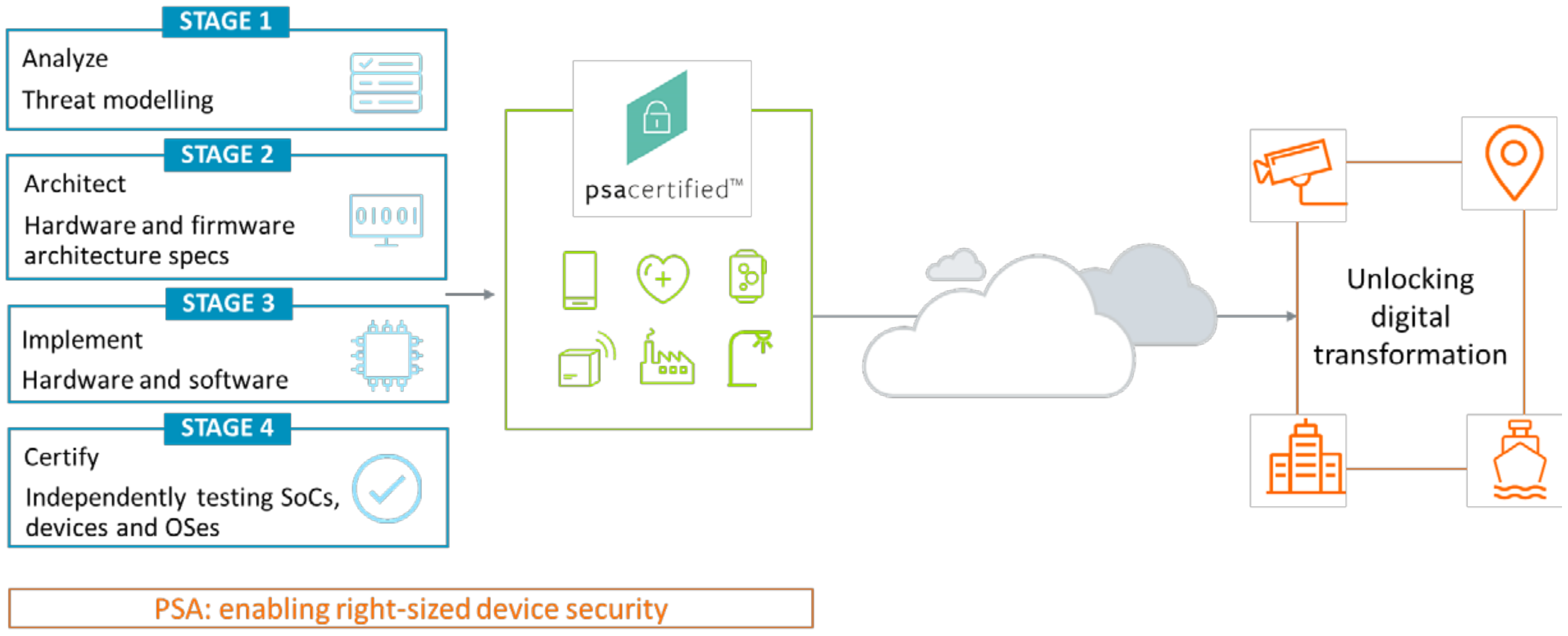
Several diagrams are required in order to fully understand a software system architecture.



Best Practices for Software Design - Security

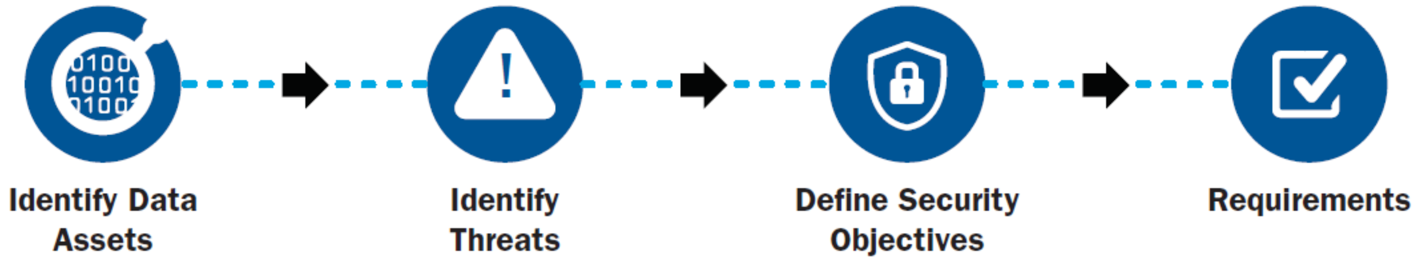
Platform Security Architecture (PSA).

- An open device security framework with independent testing.


STAGE 1
Analyze
Threat modelling
STAGE 2
Architect
Hardware and firmware architecture specs
01001
STAGE 3
Implement
Hardware and software
STAGE 4
Certify
Independently testing SoCs, devices and OSes
psacertified™
Unlocking digital transformation
PSA: enabling right-sized device security

Best Practices for Software Design - Security

Perform a threat-based security analysis to define your security objectives and requirements.



Data assets that exist in nearly all IoT devices include:

- The firmware
- Unique ID
- Passwords (flash, users, etc)
- Encryption keys (to control device, secure communication, etc)

How important is security to your product development efforts?

- Must Have
- Nice to have
- Not needed
- Other

Best Practices for Software Design

1. Manage your component dependencies carefully to minimize coupling.
2. Separate your architecture into design and implementation architectures.
3. Follow SOLID principles when you architect your software.
4. Use a UML tool like Enterprise Architect or Visual Paradigm.
5. Design the software to be hardware and language agnostic.
6. Identify data generators, sinks and transfers to architect the system.
7. Use class diagrams to define Hardware Abstraction Layers (HALs) and API's.
8. Don't forget to consider security implications in your architecture.
9. Identify the data assets that you need to protect … it won't necessarily be everything!
10. Remember, security through isolation! Hardware based isolation to layer the software to protect it.

Sponsored By

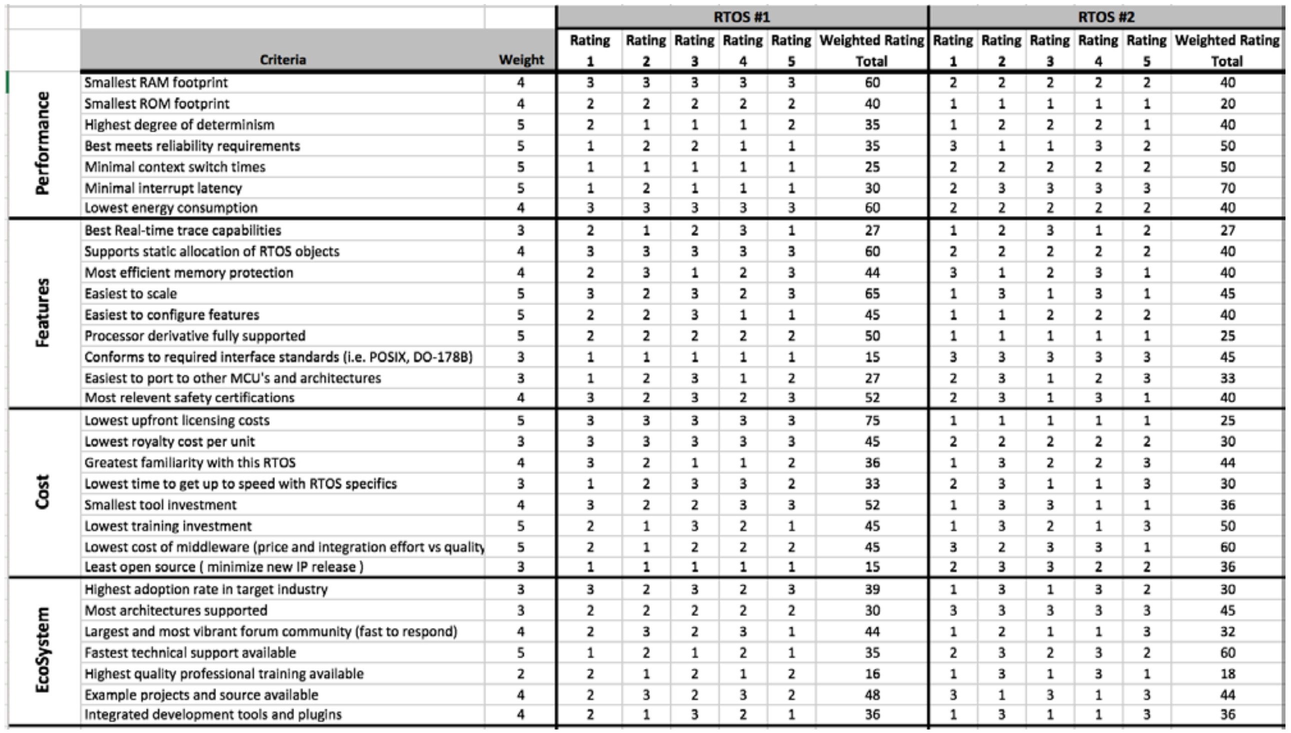
3

Best Practices for Frameworks and Open-Source Software (OSS)

Software systems today are too complex to completely implement from scratch. Leveraging frameworks, open source software and writing reusable software is required to be successful.

Best Practices for Frameworks and OSS

Use a KT Matrix to evaluate and select components in an unbiased manner that best fit your needs.



			RTOS #1						RTOS #2					
	Criteria	Weight	Rating 1	Rating 2	Rating 3	Rating 4	Rating 5	Weighted Rating Total	Rating 1	Rating 2	Rating 3	Rating 4	Rating 5	Weighted Rating Total
Performance	Smallest RAM footprint	4	3	3	3	3	3	60	2	2	2	2	2	40
	Smallest ROM footprint	4	2	2	2	2	2	40	1	1	1	1	1	20
	Highest degree of determinism	5	2	1	1	1	2	35	1	2	2	2	1	40
	Best meets reliability requirements	5	1	2	2	1	1	35	3	1	1	3	2	50
	Minimal context switch times	5	1	1	1	1	1	25	2	2	2	2	2	50
	Minimal interrupt latency	5	1	2	1	1	1	30	2	3	3	3	3	70
	Lowest energy consumption	4	3	3	3	3	3	60	2	2	2	2	2	40
Features	Best Real-time trace capabilities	3	2	1	2	3	1	27	1	2	3	1	2	27
	Supports static allocation of RTOS objects	4	3	3	3	3	3	60	2	2	2	2	2	40
	Most efficient memory protection	4	2	3	1	2	3	44	3	1	2	3	1	40
	Easiest to scale	5	3	2	3	2	3	65	1	3	1	3	1	45
	Easiest to configure features	5	2	2	3	1	1	45	1	1	2	2	2	40
	Processor derivative fully supported	5	2	2	2	2	2	50	1	1	1	1	1	25
	Conforms to required interface standards (i.e. POSIX, DO-178B)	3	1	1	1	1	1	15	3	3	3	3	3	45
	Easiest to port to other MCU's and architectures	3	1	2	3	1	2	27	2	3	1	2	3	33
	Most relevent safety certifications	4	3	2	3	2	3	52	2	3	1	3	1	40
Cost	Lowest upfront licensing costs	5	3	3	3	3	3	75	1	1	1	1	1	25
	Lowest royalty cost per unit	3	3	3	3	3	3	45	2	2	2	2	2	30
	Greatest familiarity with this RTOS	4	3	2	1	1	2	36	1	3	2	2	3	44
	Lowest time to get up to speed with RTOS specifics	3	1	2	3	3	2	33	2	3	1	1	3	30
	Smallest tool investment	4	3	2	2	3	3	52	1	3	3	1	1	36
	Lowest training investment	5	2	1	3	2	1	45	1	3	2	1	3	50
	Lowest cost of middleware (price and integration effort vs quality	5	2	1	2	2	2	45	3	2	3	3	1	60
	Least open source (minimize new IP release)	3	1	1	1	1	1	15	2	3	3	2	2	36
EcoSystem	Highest adoption rate in target industry	3	3	2	3	2	3	39	1	3	1	3	2	30
	Most architectures supported	3	2	2	2	2	2	30	3	3	3	3	3	45
	Largest and most vibrant forum community (fast to respond)	4	2	3	2	3	1	44	1	2	1	1	3	32
	Fastest technical support available	5	1	2	1	2	1	35	2	3	2	3	2	60
	Highest quality professional training available	2	2	1	2	1	2	16	1	3	1	3	1	18
	Example projects and source available	4	2	3	2	3	2	48	3	1	3	1	3	44
	Integrated development tools and plugins	4	2	1	3	2	1	36	1	3	1	1	3	36

Best Practices for Frameworks and OSS

Think quality. Through-out development, integration and deployment constantly analyze and measure your software quality.

Define the metrics that mean quality software to your organization such as:

- Cyclomatic Complexity rating
- Software builds with zero errors and warnings
- Set and measure desired code coverage during testing
- Statically analyze code for coding standard violations

Create an automated reporting system that can be ran when code is checked in.

Which of the following is the most important thing to check about OSS?

- Cyclomatic Complexity rating
- Software builds with zero errors and warnings
- Set and measure desired code coverage during testing
- Statically analyze code for coding standard violations

Best Practices for Frameworks and OSS

1. Write your software to be abstracted and hardware independent.
2. Select software that has an active community, not a one-off example.
3. Perform a software audit and quality analysis.
4. Have open-source software licenses reviewed by an attorney.
5. Use an abstraction layer to remove dependencies.
6. Leverage integrated software when possible to minimize issues.
7. Build an acceptable testing process to verify the software's robustness.
8. Carefully read any existing documentation and example code.
9. Perform an unbiased analysis to determine if the software is right for you.
10. Be extremely careful when upgrading to a new software version.

Continuing Education Center

Sponsored By

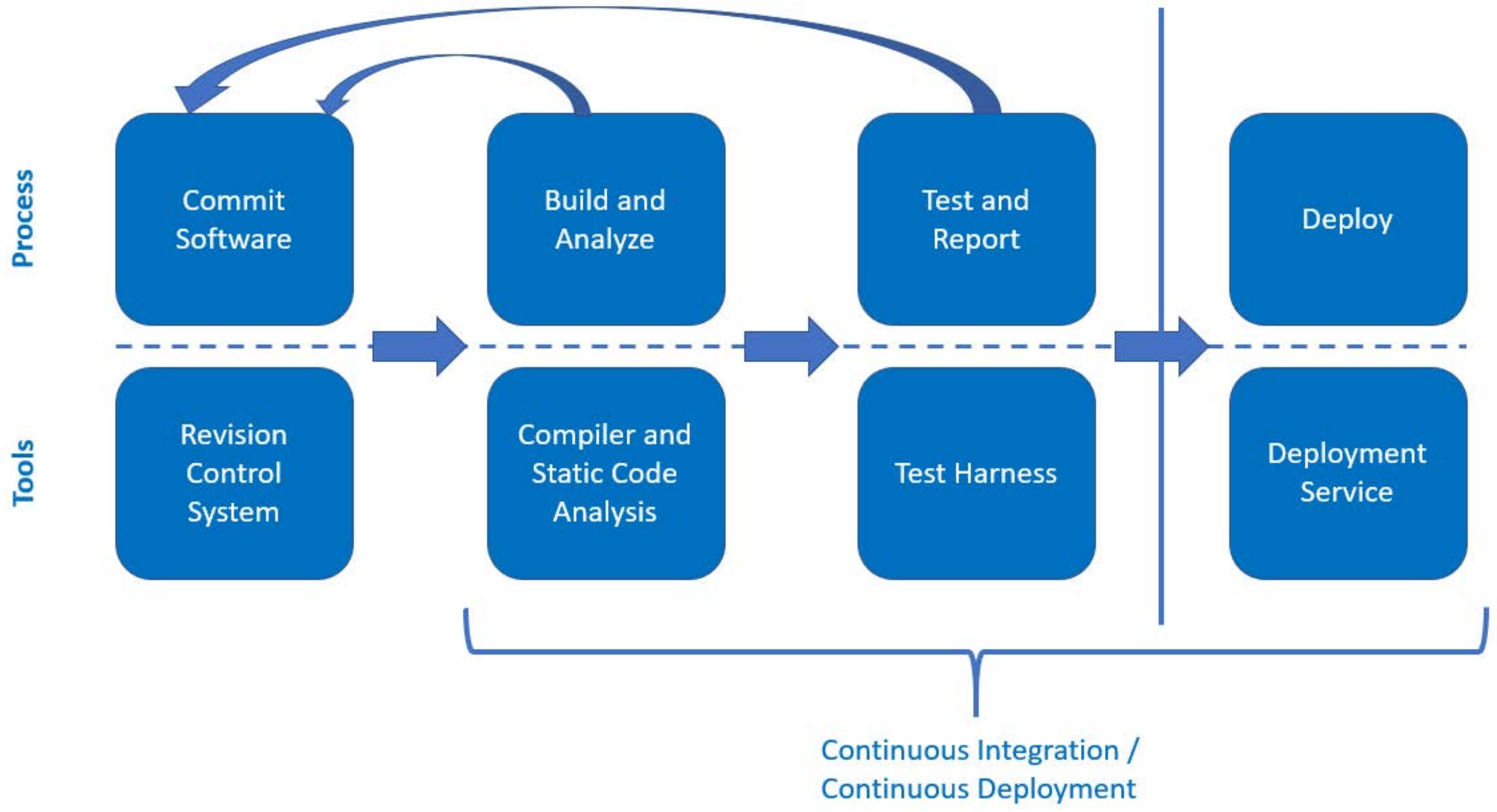
4

Best Practices for Testing

Testing is the most neglected phase by many companies. Testing has evolved over the past several years to not just be a core requirement, but the foundation on which quality systems are built.

Best Practices for Testing

Create a continuous integration and continuous deployment (CI/CD) pipeline.



Best Practices for Testing

Test Driven Development (TDD) is a technique that advocates writing a failing test first before writing code that makes the test pass.

TDD provides several benefits and best practices such as:

- Unit testing a function
- Generating tests that catch a failure mode
- Can be migrated to an automated build server
- Provides regression testing
- Decreases the time spent debugging

How much experience do you have with CI/CD?

- Expert user
- Intermediate user
- Beginner
- What's CI/CD?

Best Practices for Testing

1. Testing today needs to be automated. Software is too complex for manual testing.
2. Take the time to design a build pipeline that fits your organization.
3. Setup a build server to automatically compile and test your software.
4. Develop your software using techniques like test driven development (TDD).
5. Select a commonly used test harness like CppUTest.
6. Don't forget to create tests for component integration.
7. Create an automated pipline for CI/CD.
8. Test your software on target, not just on a build server.
9. Take the tie to create blackbox tests for frameworks and open source software that you are using.
10. Leverage tools such as Jenkins and CPU models to automate your testing process.

Thank you for attending

Please consider the resources below:

- www.beningo.com
    - Blog, White Papers, Courses
    - Embedded Bytes Newsletter
        - http://bit.ly/1BAHYXm

From www.beningo.com under

- Blog > CEC – Best Practices for Real-Time Embedded Systems

CEC Continuing Education Center

DesignNews

Thank You

Sponsored by

Digi-Key ELECTRONICS

informa markets