Zephyr™ Project
Developer Summit
June 8-10, 2021 ▪ @ZephyrIoT

Field Report: Setting up a Software Product Line (SPL) Architecture

GREGORY SHUE, LEGRAND

Agenda

- Introduction to Software Product Lines (SPL) – 6 minutes
- Zephyr Project Support for SPL Core Assets – 6 minutes
- To Achieve High Level of SPL Support in Zephyr (v2.6.0) – 3 minutes
- Discussion of Proposals – 12 minutes

What Is a Software Product Line (SPL)?

(from Software Product Line Essentials presentation, SEI Carnegie Mellon University, 2008)

Definition:

A software product line is

a set of software-intensive systems
sharing a common, managed set of features
that satisfy the specific needs of a particular market segment or mission
and that are developed from a common set of core assets in a prescribed way.

- a new application of a proven concept
- an innovative, growing concept in software engineering

Software product lines involve **strategic, planned reuse**
that yields **predictable results**.

A Software Product Line is NOT

(from Software Product Line Essentials presentation, SEI Carnegie Mellon University, 2008)

Clone and own: single-system development with reuse

- modifying code as necessary for the single system only

Fortuitous small-grained reuse

- reuse libraries containing algorithms, modules, objects, or components

Just component-based or service-based development

- selecting components or services from an in-house library, the marketplace, or the Web with no architecture focus

Just versions of a single product

- rather, simultaneous release and support of multiple products

Just a configurable architecture

- a good start, but only part of the reuse potential

Just a set of technical standards

- constraining choices without an architecture-based reuse strategy

Where has an SPL been used?

(from Software Product Line Essentials presentation, SEI Carnegie Mellon University, 2008)

Successful software product lines have been built for families of among other things

- mobile phones
- command and control ship systems
- satellite ground station systems
- avionics systems
- command and control/situation awareness systems
- pagers
- engine control systems
- mass storage devices
- billing systems
- web-based retail systems
- printers
- consumer electronic products
- acquisition management enterprise systems
- financial and tax systems
- medical devices
- farm fish management software

Product Line Adoption and Institutionalization

(from Software Product Line Essentials presentation, SEI Carnegie Mellon University, 2008)

Innovators and early adopters demonstrated the feasibility and the benefits of software product lines:

- CelsiusTech
- Cummins, Inc.
- Hewlett-Packard
- Motorola
- Nokia

The SEI and others have tried to lower the adoption barrier by codifying practices, writing case studies, perfecting methods useful in product line approaches, and engendering a software product line community.

Many organizations are now handsomely achieving their business goals using a software product line approach.

Speaking from Experience:
SPL Hall of fame - HP Owen FW Coop

SPL Hall-of-fame: Hewlett-Packard Owen Firmware Cooperative (1997 – present)

- For connected devices targeted to run 24/7 for >7 years without a power cycle
- First 7 generations designed around RTOS on 32b ARM9 in HP SOCs
- Evolved into a continuously-integrated world-wide git repository
- Product line included IoT printers, scanners, faxes, and camera docs
- Far more successful than originators imagined

SPL Strategy for Legrand

- Zephyr ecosystem
  - Following Zephyr's T2: Star Topology with application repository at the hub
  - All repositories tied into the Zephyr build system through `module.yml`
- Partitioning into repositories based on ownership and licensing
  - (50+) Zephyr Project modules and Zephyr repositories
  - (6+) Open source & commercial repositories
    - extending as modules with glue code, APIs, tests, samples, docs, manifest for extension development
  - (1) Proprietary manifest + module SPL application repository
    - multiple manifests necessary from using multiple Zephyr forks
- All functionality in composable, configurable subsystems, drivers, and data sets
- Workspace documentation in manifest repository
  - Documentation structure and tools reuse or parallel Zephyr
  - Requirement Tracing generation expected to be provided by Zephyr tools

Costs of a Software Product Line

(from Software Product Line Essentials presentation, SEI Carnegie Mellon University, 2008)

Core Assets	Costs
Architecture	Must support variation inherent in the product line
Software Components	Must be designed to be general without a loss of performance; must build in support for variation points
Test Plans, Test Cases, Test Data	Must consider variation points and multiple instances of the product line
Business Case and Market Analysis	Must address a family of software products, not just one product
Project Plans	Must be generic or be made extensible to accommodate product variations
Tools and Processes	Must be more robust [than required for single product development]
People, Skills, Training	Must involve training and expertise centered around the assets and procedures associated with the product line

What is needed for High Level Support of SPL?

1. Embrace Extensibility
   - Perpetually maintain full functionality across module extensibility.
   - Perpetually evaluate how each aspect of the ecosystem impacted by module extensibility.

2. Embrace Live REUSE of Zephyr
   - Adjust patterns and organizations to also support treating Zephyr as a module + build tools.
   - Perpetually consider topologies where Zephyr is not the manifest repository.

3. Embrace Composability
   - Perpetually consider a “project” is simply a collection of functionality supported by a configuration of subsystems, libraries, and drivers.

Embrace Extensibility (Proposals)

Secure FW imposes requirements on the entire executable, so all code built into the image is affected.

1. *Proposal:*
   Define “codebase” in glossary to be “all the source for the Zephyr config/build/CI/doc generation system”.

Where Secure FW artifacts are needed, the artifacts also need to cover the entire codebase (e.g., all modules).

2. *Proposal:*
   All tools for FW configuration, build, debugging, testing, and document generation need to:
   - recognize and operate on the entire codebase found through `ZEPHYR[_EXTRA]_MODULES`
   - follow all settings in (explicit or implicit) `module.yml`

Embrace Extensibility (cont) (Proposals)

MISRA Dir 3.1 (Required), states “All code shall be traceable to documented requirements”, so generated requirements traceability artifacts will need to include the entire codebase. Supporting the T2 topology implies the Zephyr documentation tools can generate documentation (including requirements traceability) from the entire codebase.

3. *Proposal:*
   The Zephyr documentation is to be adjusted to include documentation found in the entire codebase.
   - How documentation is re-organized needs to be managed by the document maintainers & collaborators.

Embrace Live REUSE of Zephyr (Proposals)

Live REUSE includes topologies where the unmodified Zephyr repository is just another “module” in the workspace

4. *Proposal:*
   The Zephyr document generation tools support workspace-level content coming from the manifest repository.
   - Workspace:
     - Explicit: SPL name, Theme, Logo, Contents/shortcuts, Introduction (workspace level)
     - Generated: Module list (w/ versions), Glossary, Boards, Samples, API & symbol references
   - Module:
     - Explicit: Module name, Logo, Introduction, Guides, Release Notes?
     - Generated: Boards, Samples, API & symbol references

5. *Proposal:*
   Zephyr CI verifies support of all user-configurable settings in `module.yml`.

Embrace Composability (Proposals)

Composability means being able to configure boards, drivers and subsystems together into a working device without requiring crafting or tuning.

6. *Proposal:*
   Specify a location in the Zephyr directory tree structure for `*.conf` overlays. Provide documentation and an example to show how they are referenced in the project's `CMakeLists.txt`.

Zephyr™ Project
Developer Summit

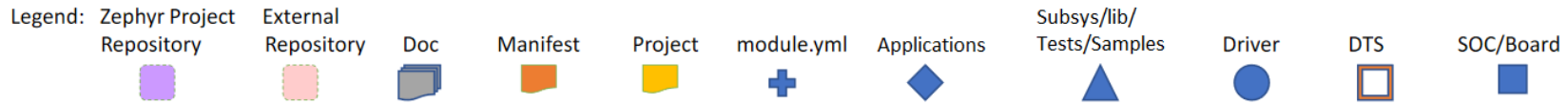
Discussion


Zephyr™ Project
Developer Summit
June 8-10, 2021 ▪ @ZephyrIoT

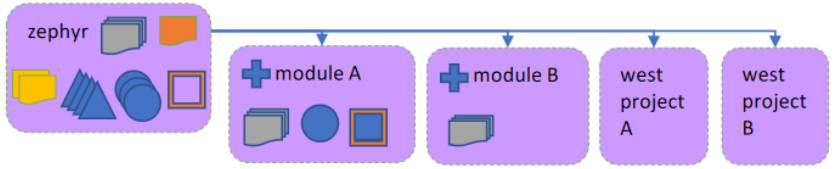
Zephyr™ Project
Developer Summit

Backup Slides

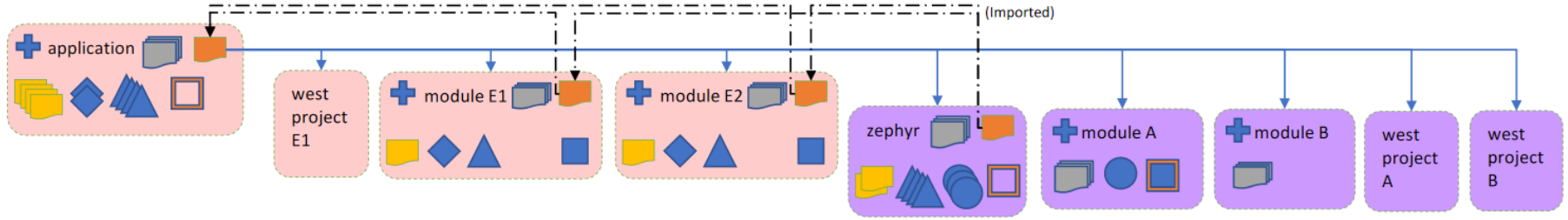
Zephyr Project Supported Topologies



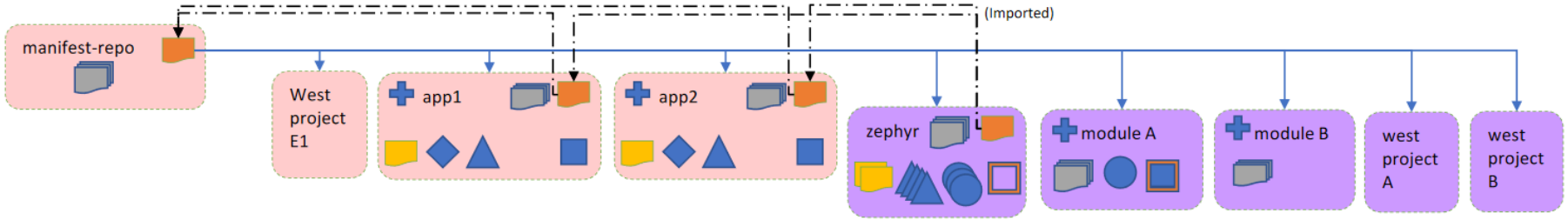
T1: Star topology, zephyr is the manifest repository



T2: Star topology, a Zephyr application is the manifest repository



T3: Forest topology, freestanding manifest repository



SPL Core Asset Support: Architecture

Must support **variation inherent in the product line**

Supported by:

- Kconfig, macros
- Configurable, pattern-based build artifact path
- Composable feature sets (e.g., `*.conf` overlays)
- API caller instance pointer and client data

For High Level of SPL Support

- Control library exposure separate from APIs
- Guidance on when to caller's stack instead of own
- Standard API for run-time adjustments to board variances
- Clean, controlled shutdown

SPL Core Asset Support: Software Components

Must be **designed to be general** without a loss of performance;
must **build in support for variation points**

Supported by:

- Boards, DTS
- Drivers, subsystems, libraries
- Tests, samples, twister, code coverage
- Modules
- Multiple manifest support

For High Level of SPL Support

- Config/build/CI/doc tools supporting modules
- Tree for non-test, non-sample projects
- Design docs with
  - Thread, stack, mutex usages
  - Relative prioritization assumptions
  - Latency tolerances

SPL Core Asset Support: Test Plans, Test Cases, Test Data

Must consider **variation points** and **multiple instances** of the product line

Supported by:

- `ztest`, `tests/`, `samples/`
- `twister`, code coverage (branch & statement)
- Doxygen tags for requirements
- Requirements (coming)
  - Statements, organization, tracing

For High Level of SPL Support

- `twister` to reference
  - `module.yml:tests`, `module.yml:samples`
- Identify open-source tool for MC/DC coverage
- Test Plans, Cases and Data must reflect variation points

SPL Core Asset Support: Business Case and Market Analysis

Must **address a family of software products**, not just one product

Supported by:

- Focused on Secure IoT devices driven by MCU
- T2: Star topology w/ app as hub (i.e., reuse Zephyr)
- `samples/`, `tests/`, `boards/` are trees
- Multiple SOC arch support
- `*.conf` overlays

For High Level of SPL Support

- Zephyr directory tree for non-test, non-sample projects
- Zephyr directory for shareable `*.conf` overlays
- Zephyr sample of a project with an empty `main()`

SPL Core Asset Support: Project Plans

Must be generic or **be made extensible** to accommodate product variations

Supported by:

- `ZEPHYR[_EXTRA]_MODULES`
- `modules.yml` settings
  - Build
  - Samples
  - Tests

For High Level of SPL Support

- `twister` and doc generation to reference settings in `module.yml`
- Update Zephyr documentation structure to support
  - composable modularity
  - when Zephyr is only another “module” in the workspace

SPL Core Asset Support: Tools and Processes

Must be more robust *[than required for single product development]*

Supported by:

- Zephyr Project processes
- Zephyr Project CI

For High Level of SPL Support

- User-configurable settings must be reliably functional and fully supported (e.g., those in `module.yml`).

SPL Core Asset Support: People, Skills, Training

Must involve training and expertise centered around the **assets** and **procedures** associated with the product line

Supported by:

- TSC
- Maintainers
- Collaborators
- Zephyr Project documentation
- Working Groups

For High Level of SPL Support

- Consistently follow existing processes
- Clearly define terms and consistently apply them
  - “codebase”, “module”
- Recognize that requirements tracing (and documentation generation in general) needs to be just as important and extensible as the source code