30th Annual INCOSE international symposium

Virtual Event
July 20 - 22, 2020

# Do Product Lines Have Sweet Spots?

30th Annual INCOSE international symposium

Virtual Event
July 20 - 22, 2020

**Keith Harper** joined Rolls-Royce in 1985 and has over 30 years experience in the development of Control Systems and Software. He is the Controls North America Chief Design Engineer with accountability for technical review, Civil and Military certification, Safety and Process Assurance. He is a past member of the SAE E-36 Electronic Engine Controls Committee

**Andrew Pickard** joined Rolls-Royce in 1977 after completing a Ph.D. at Cambridge University in Fatigue and Fracture of Metals and Alloys. He is a Rolls-Royce Associate Fellow in System Engineering, a Fellow of SAE International, a Fellow of the Institute of Materials, Minerals and Mining, a Chartered Engineer and a member of INCOSE. He is immediate past Chair of the SAE Aerospace Council, represents Rolls-Royce on the INCOSE Corporate Advisory Board and is Chief of Staff for INCOSE.

# Contents

# What is a Product Line?

**Definitions**

**Product Line:** A collection of reusable System assets, with built-in variability
**Instantiation:** Tailoring the Product Line to create a valid, unique Application

**Application:** A verified product created from the Product Line

**Objectives**

- develop assets once, and then re-use across multiple Applications
- **maximum** Product Line reuse and **minimum** Application specific change.

Product Lines must by **systematically architected** and developed with **deliberate reuse** in mind.

# Where is the Sweet Spot for Product Lines?

**Consider**

- Scope of the product line?
  - How similar are the Products (requirements, interfaces and use cases)?
- Candidate architecture?
  - Based upon shared (same), similar (variable) and unique features (optional or alternate)
- Variability mechanisms?
  - a single common part number, or a reconfigurable modular part
  - a feature that is selectable (included or not included) or tailored (different parameters) or substituted (pre-defined alternates).
  - Can vary between systems, software and hardware

# Managing Variability

Feature Model provides Product Line tailoring “rules”

Instantiation process “how to” use the Product Line and instantiate an application

Assets include products and lifecycle data supporting maximum reuse

Our Example:

Existing helicopter engine

Multiple New applications

New Control System

## Feature Model

Modular features and requirements

Selection rules - mandated, with allowable tailoring, optional (select 0 or more) or inclusive/exclusive (AND, OR)

Allowable data ranges

## Instantiation Process

PL & application team roles

How to use the feature model

How to validate/verify application

Configuration control of PL & application

## Assets

Reqts, design, code, test, safety etc

Correctness on Product Line

Completeness on Application

PL & application configured separately

## Example

Single Hardware part number with input/output superset

i/o reconfigurable in layered s/w

Product Line software with in-built variability (logic and data)

# Developing a Product Line – Asset Example

The goal is to develop once, and then re-use multiple times

It's important to manage the lifecycle data as re-usable assets to the maximum possible extent.

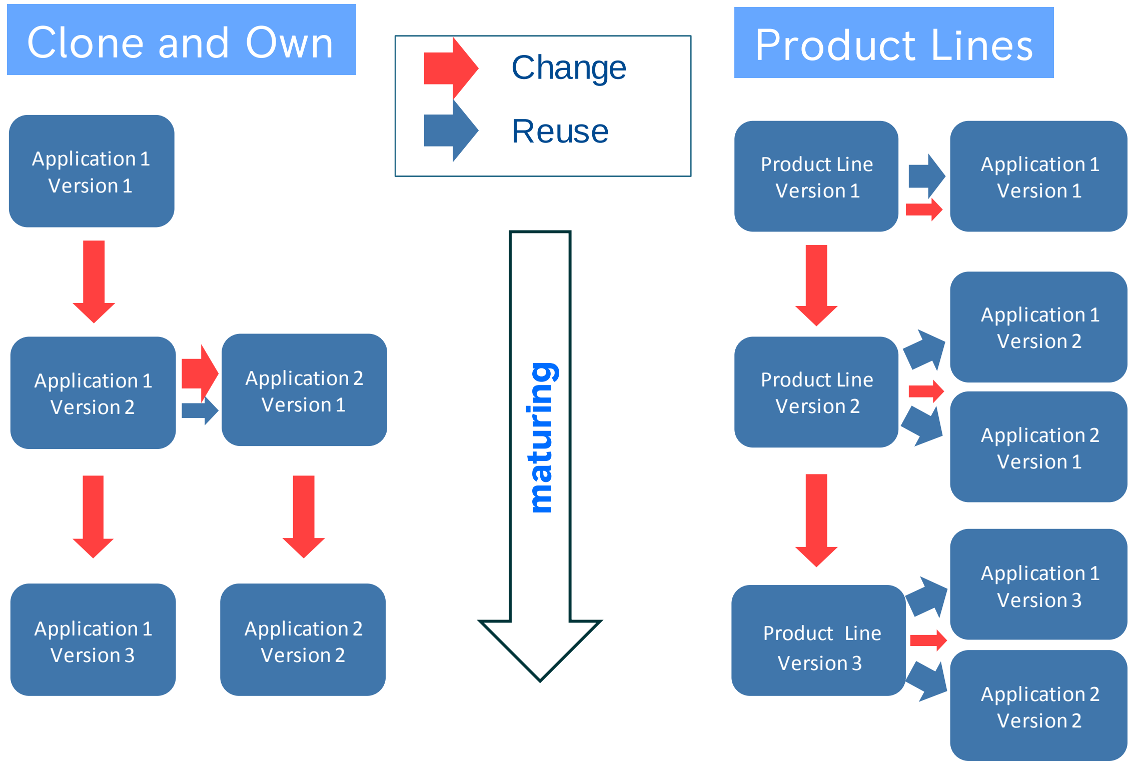
The Application must then demonstrate the integration of those assets and the completeness of the Application itself.

Helo applications achieved >90% requirements reuse from the Product Line

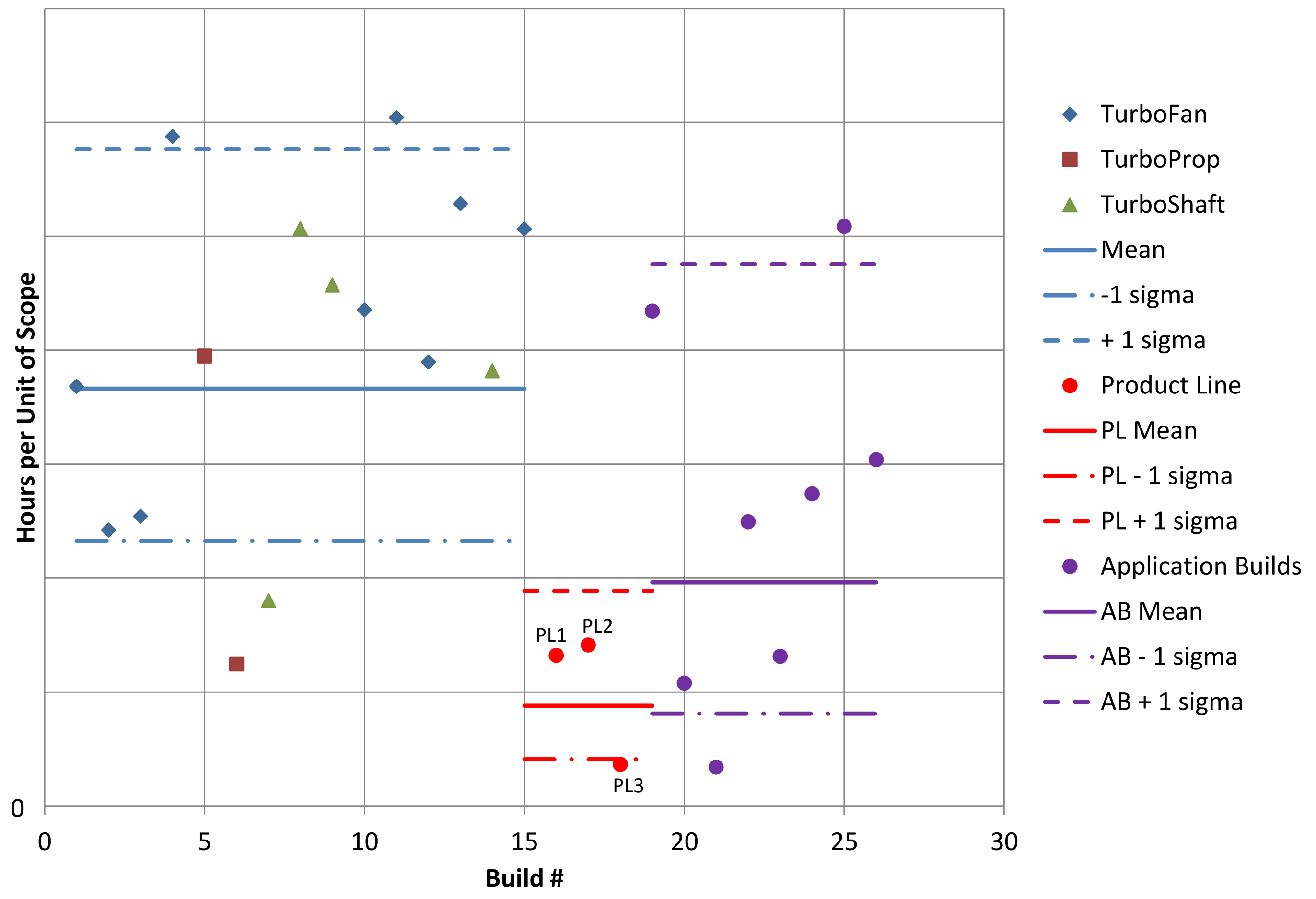
| System Development | | System Verification | |
|---|---|---|---|
| Product Line | Application | Product Line | Application |
| Requirements | Instantiated Requirements | Review evidence (correctness) | Review evidence (completeness) |
| Feature Model | Selected Features | Traceability/Validation data | Validation Matrix |
| Safety Case | Instantiated System Safety Analysis | Verification Cases & Procedures * | Instantiated Verification Cases & Procedures |
| | Interface Control Document | | Verification results |
| | | | Verification Coverage Analysis |
| **Software Development** | | **Software Verification** | |
| Product Line | Application | Product Line | Application |
| Requirements | Instantiated Requirements | Review evidence (correctness) | Review evidence (completeness) |
| Design | Instantiated Design | Traceability/Validation data | Validation Matrix |
| Source Code | Instantiated Source Code | Verification Cases & Procedures * | Instantiated Verification Cases & Procedures |
| | Executable | Component test results * | Software/Software Integration results |
| | | | Hardware/Software Integration results |
| | | | Verification Coverage Analysis |
| | | [*across allowable PL range] | Run-time and memory analysis |

# Developing a Product Line - Genealogy

- In “clone and own”
  - new Applications are separately maintained copies, with ad-hoc re-use sharing requirements, validation and solutions where possible.
  - each Application is built off its previous version
- In contrast, Product Lines form the development back bone,
  - new applications (or new versions of Applications) re-instantiated from the Product Line.
- **Delay creating multiple applications until PL mature**



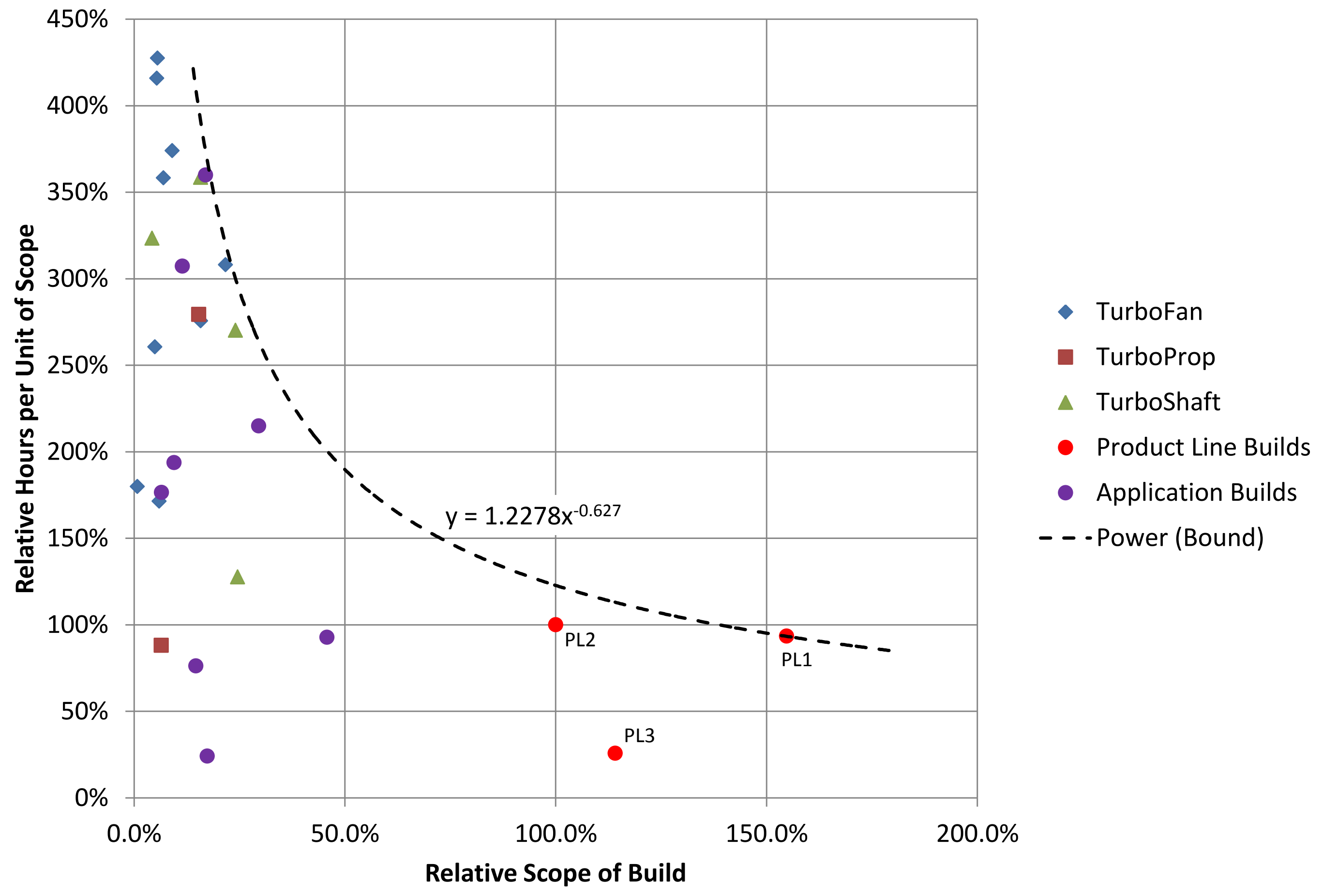
# Software Metrics - Cost



Cost per unit of scope, by build

- "Scope" for each Problem Report is the number of software modules opened to make the changes to address the problem, and the scope for the build is the sum of modules opened to address the Problem Reports in the build
- Statistical hypothesis testing showed a significantly lower mean cost per unit of scope of the Product Line builds.
- Combined with the higher proportion of functional problem reports in these builds, this shows that Product Line builds are a very cost-efficient approach to adding functionality to control system software.

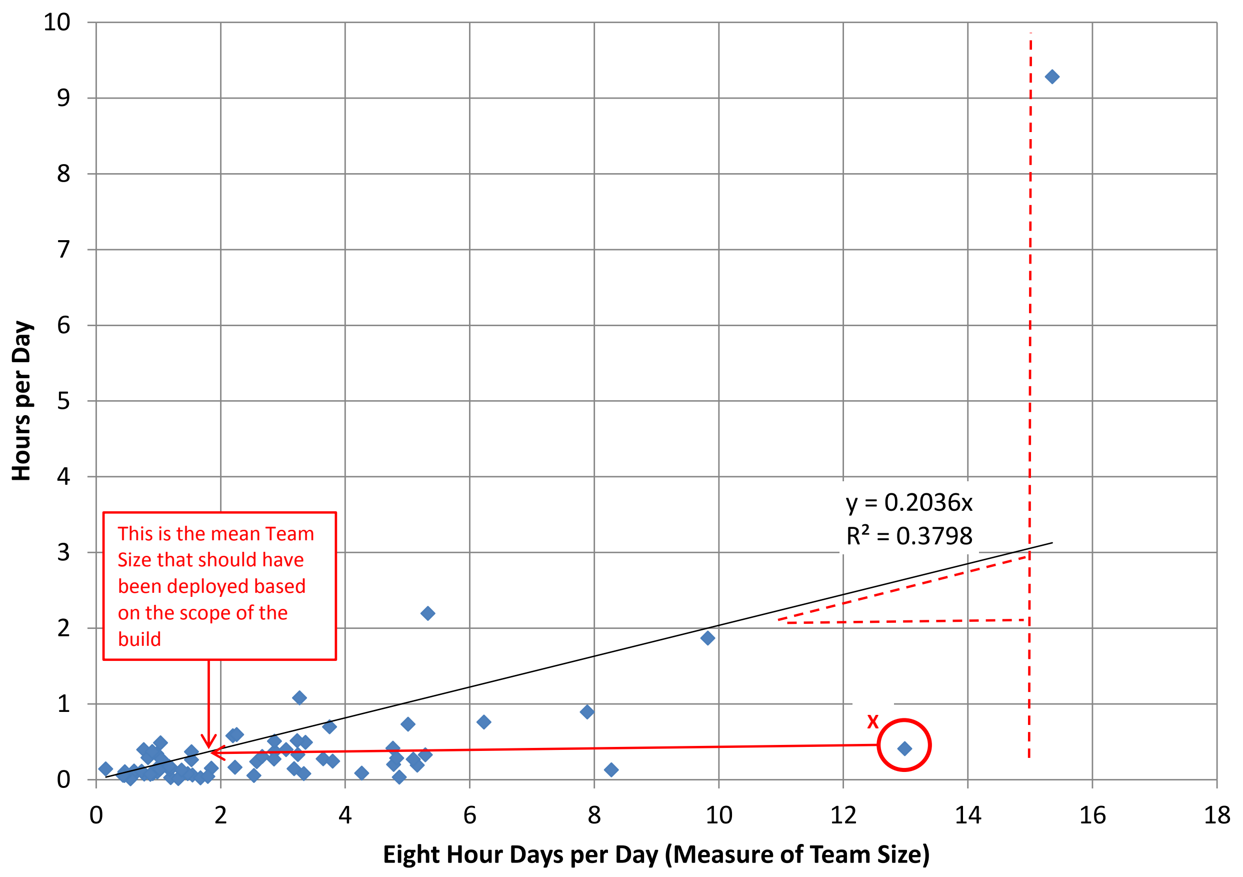
# Software Metrics - Cost



This figure compares the relative cost (hours) per unit of scope for the builds with the relative scope of the build (normalized to build PL2).

- There is much more scatter in cost per unit of scope for smaller scope builds than for larger builds; the power law fit is to the upper bound of all of the builds.
- The Product Line builds are between two and three times larger than the largest of the Application and Non-Product Line builds – another driver for the cost-efficiency of the Product Line builds.

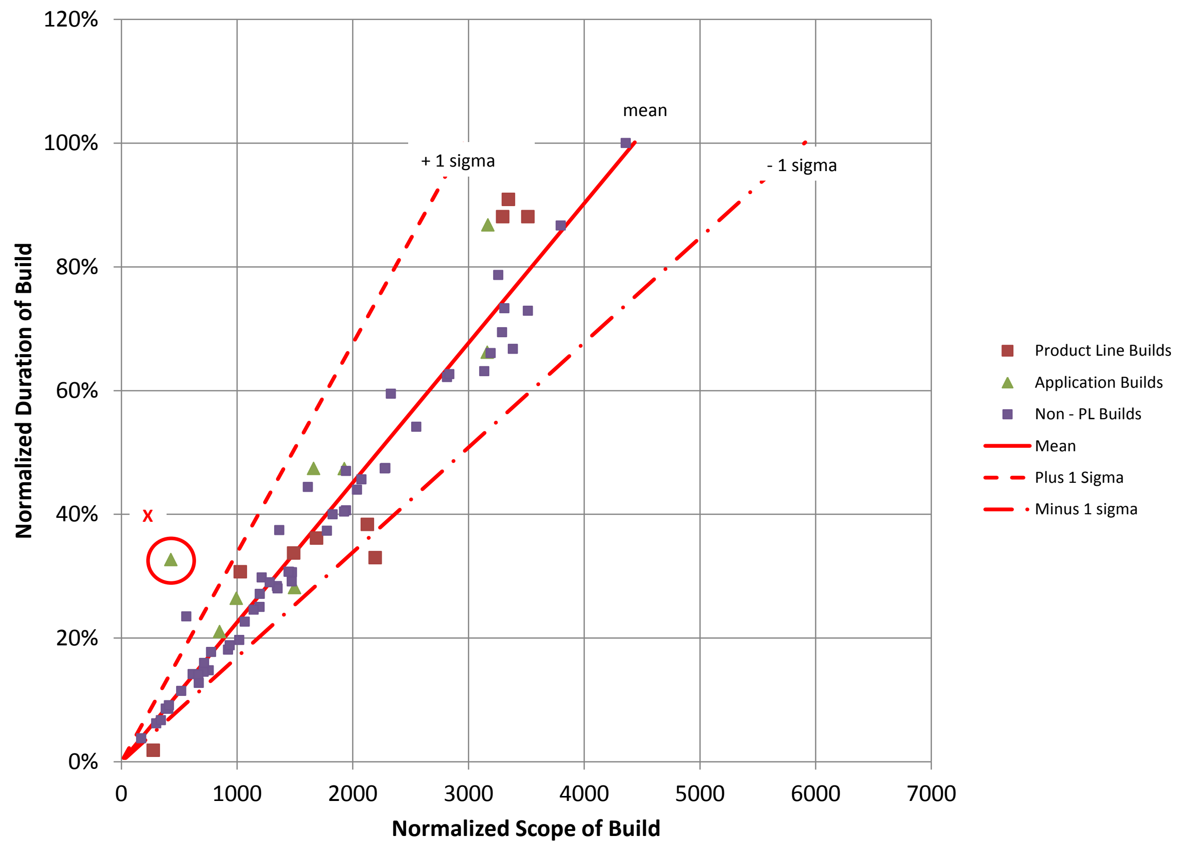
# Software Metrics - Schedule

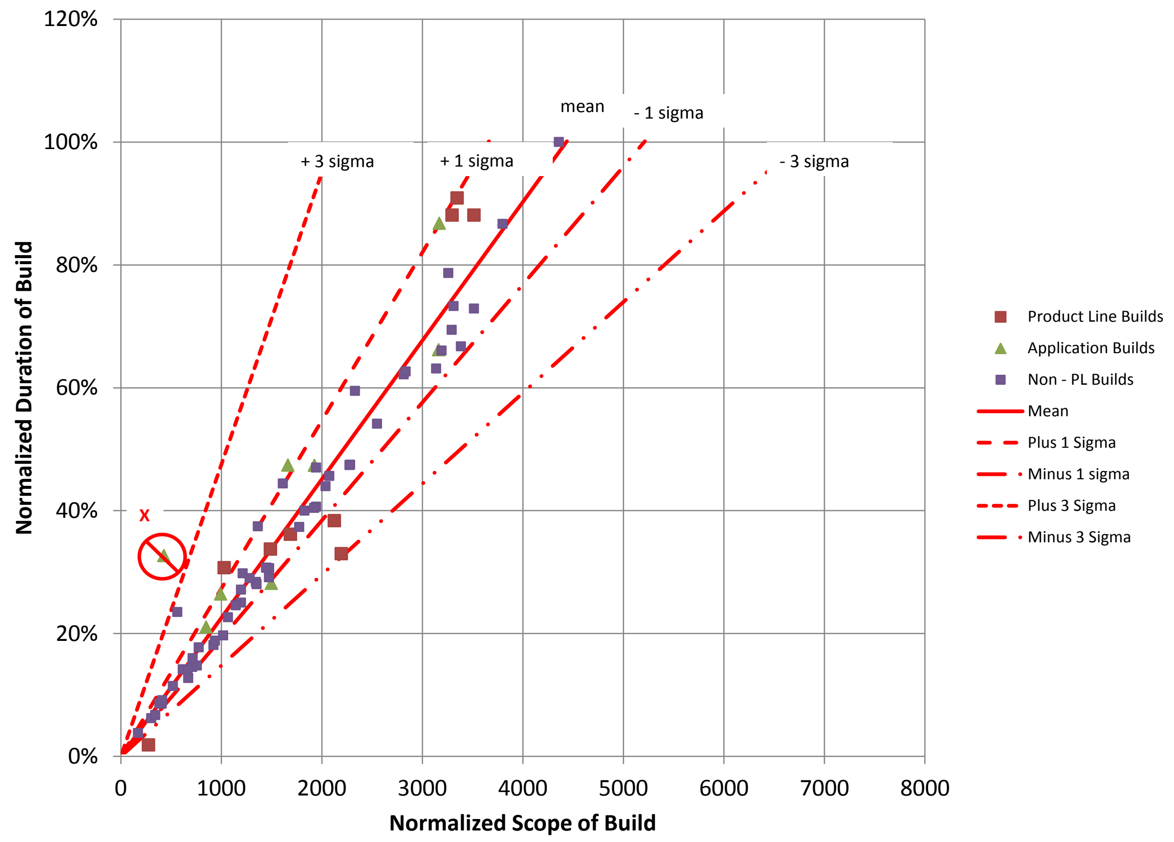


The schedule for software builds is much more strongly impacted by team size than the cost for the build. An approach is required to normalize for team size (Pickard and Nolan, 2012)

- Team size is measured as number of 8 hour days per day deployed on the build
- Team size is normalized to 15 8-hour days per day using the slope of the mean line through the builds
- Build "X" employed a very large team relative to the scope of the build as a means of compressing the schedule of the build.

# Software Metrics - Schedule





There is a statistically significant difference in variance if point "X" is included in the schedule correlation.  The results show that deploying a large team on a build to address schedule compression has an adverse impact on the schedule correlation, in that more people are needed in the team, given the scope of the build, to achieve the compressed schedule than would be predicted by the schedule correlation.

# Software – Product Line Maturity

**Application Builds - Parallel/Duplicate Activities**

| Metric | Build | PL Issue Fixed in Multiple Apps | PL Issue fixed Concurrently in the PL and App Builds | PL Issue Fixed in the App and in the Next PL Build | Application - Specific Issue |
|---|---|---|---|---|---|
| PRs | A | 25% | 0% | 50% | 25% |
| | B | 63% | 0% | 13% | 25% |
| | C | 93% | 0% | 0% | 7% |
| | D | 0% | 81% | 0% | 19% |
| | E | 40% | 0% | 0% | 60% |
| Modules Opened | A | 20% | 0% | 42% | 38% |
| | B | 43% | 0% | 8% | 49% |
| | C | 84% | 0% | 0% | 16% |
| | D | 0% | 87% | 0% | 13% |
| | E | 13% | 0% | 0% | 87% |

Builds A to D built off the immature Product Line.

Build E built off the mature Product Line

## Benefits

- Tangible benefits (expected) included cost, schedule, quality, and maturity.
  - Some of these could have been due to the improved system engineering focus on the Product Line.
- Clear benefits when Instantiating an Application from a mature Product Line
  - Reduced the effort involved, and enabled smaller, less experienced and more efficient teams to produce the Application.
  - Previous “clone and own” efforts had produced a new certified Application in 24 months. With Product Lines this was reduced to 12.
- More intangible benefits emerged.
  - Product Lines promote a standard house-style.
  - Team flexibility improved as there was a larger pool of engineers familiar with the product, process, tools and lifecycle data.
  - Maturing features becomes easier as there were multiple use-cases contributing to the robustness of a single Product Line solution.

## Conclusions

- The Product Line goal is to develop once, and then re-use multiple times.
- In addition to exploiting the planned variability, it's important to manage the lifecycle data as re-usable assets to the maximum possible extent, minimizing both effort and technical risk
- Building Applications from an immature Product Line causes a loss in efficiency as a higher proportion of the changes are deployed multiple times rather than once in a Product Line build.
- The Software Metrics analysis shows that
  - Product Line Builds are a very efficient way of adding functionality to a control system; the builds concentrate effort on software creation and verification, they are large scope builds with more of the build effort focused on adding functionality, and they require significantly less cost (hours) per unit of scope to be deployed
  - Like traditional development, deploying a large team to address schedule compression has an adverse impact, in that more people are required to achieve the compressed schedule than would be predicted by the schedule metrics and the scope of the build
- Building a Product Line should not be undertaken lightly. It is a big investment, but presents a huge benefit once the Product Line is mature and Application builds can be created and certified as airworthy much faster and at lower cost than had previously been the case with "clone and own" builds.

30th Annual INCOSE international symposium

Virtual Event
July 20 - 22, 2020

www.incose.org/symp2020