PHILIPS

Managing Software Assets in Product Populations

SASG Meeting June 7, 2005

Egbert Algra

Software architect, Philips Medical Systems, BU Medical-IT

Egbert.Algra@Philips.Com

Overview

- PMS introduction
- Problem introduction
- Approach
  - Things that work
  - Problems that remain
- Conclusion

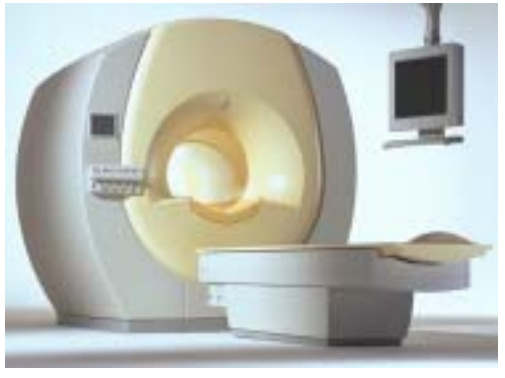
Introduction: PMS company

- Employees: 30.000
- Development sites:
  - The Netherlands: Best and Heerlen
  - Finland: Helsinki
  - Germany: Hamburg and Böblingen
  - Israel : Haifa
  - USA: Bothell and Seattle, Washington; Reedsville and Philadelphia, Pennsylvania; Andover, Massachusetts; Milpitas and Oxnard, California; Cleveland, Ohio; Chicago, Illinois
- Considerable growth
  - Doubled in size last 5 years
  - More organic and acquisition growth anticipated

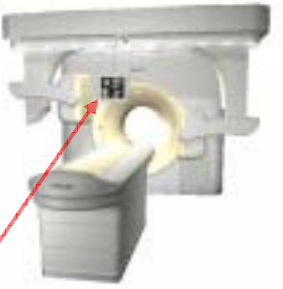
Introduction: PMS products

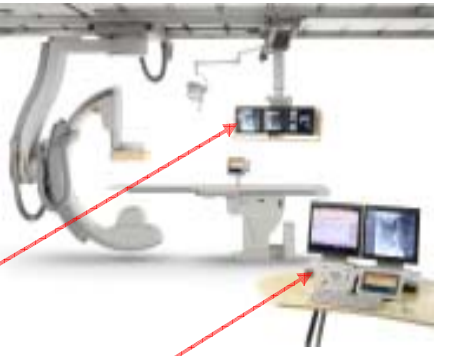
- Medical Imaging equipment (“modalities”)
  - X-ray
  - Ultrasound
  - Magnetic resonance
  - Computed tomography
  - Nuclear medicine, PET,
  - Radiation oncology systems
  - Patient monitoring
  - Information management and resuscitation products.
- Services (training and education, business consultancy, financial services and e-care business services)
- Worldwide market share in Top-3 segments
  - (competing GE and Siemens)

Product samples

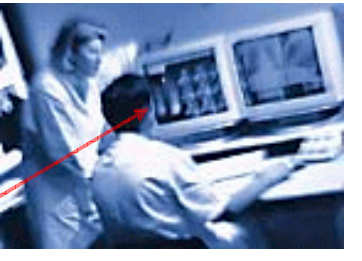


Product trends





- Combined products
  - Combined products (CT/PET, MR/Xray)
  - HW and SW
    - Including the ‘SW only’ parts
- Similarity towards the end-user
  - User interface ABC
    - Appearance, Behaviour, Concept
- Hospital integration
  - Similar clinical tasks at various workspots
    - Scanner, Operator console, Workstation, Office
- *Challenge*: (re)use same SW components
  - In the various product families



Problem Description

- Managing the medical imaging software assets as deployed in various product families
  - Assets: Medical imaging workspot for digital image handling and advanced clinical processing
    - Multi modality viewing (X-ray, MR, CT, US, NM)
    - Clinical image post processing and analysis
    - Clinical reviewing and reporting applications
  - Deployment on a variety of product families
    - Acquisition console
    - Advanced clinical processing workstation
    - PACS viewing client

Relevance & Benefits

- Software assets represent significant value (3+ MLOC), and are created at significant cost (300+ man-years).
- Product families approach:
  - Improves time-to-market
  - Reduces development and maintenance costs
  - Promotes common product ‘look-and-feel’ for the same functions on different products

Scope: operational aspects

- Operational aspects
  - Requirements management
  - Architecture
  - Variation
  - Project management
- Out of scope
  - Business aspects
    - Who Control
    - Financials
  - Organizational aspects
    - Component/platform groups versus product development groups

Approach overview: *software asset variation management*

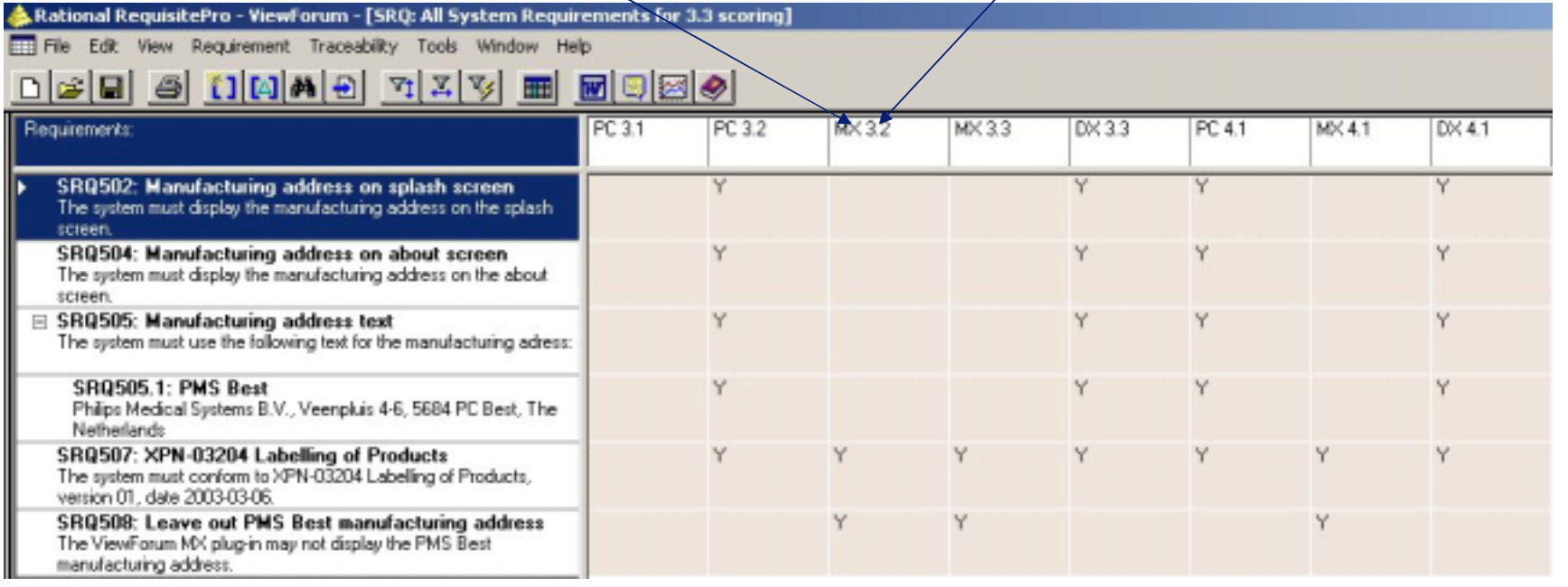
- Requirements
  - How to manage varying asset requirements for the different target deployment system (of which some are conflicting)?
- Architecture
  - How to deploy assets on different product system architectutes?
  - How to manage the variation in ‘standards’ implementations?
  - How to manage the creation of the various asset variants from the source base?
- Lifecycle
  - How to maintain the specific asset versions that are deployed in products?

Requirements approach: *requirements management variations*

- Experience: requirements vary for different deployment systems.
  - All asset requirements are gathered in a single requirements database (Rational RequisitePro)
  - Each requirements is equipped with ‘attributes’ identifying the applicable variants
  - The complete set of requirements from an asset variant is generated by a ‘query’ on the database

Example:
requirements management variations

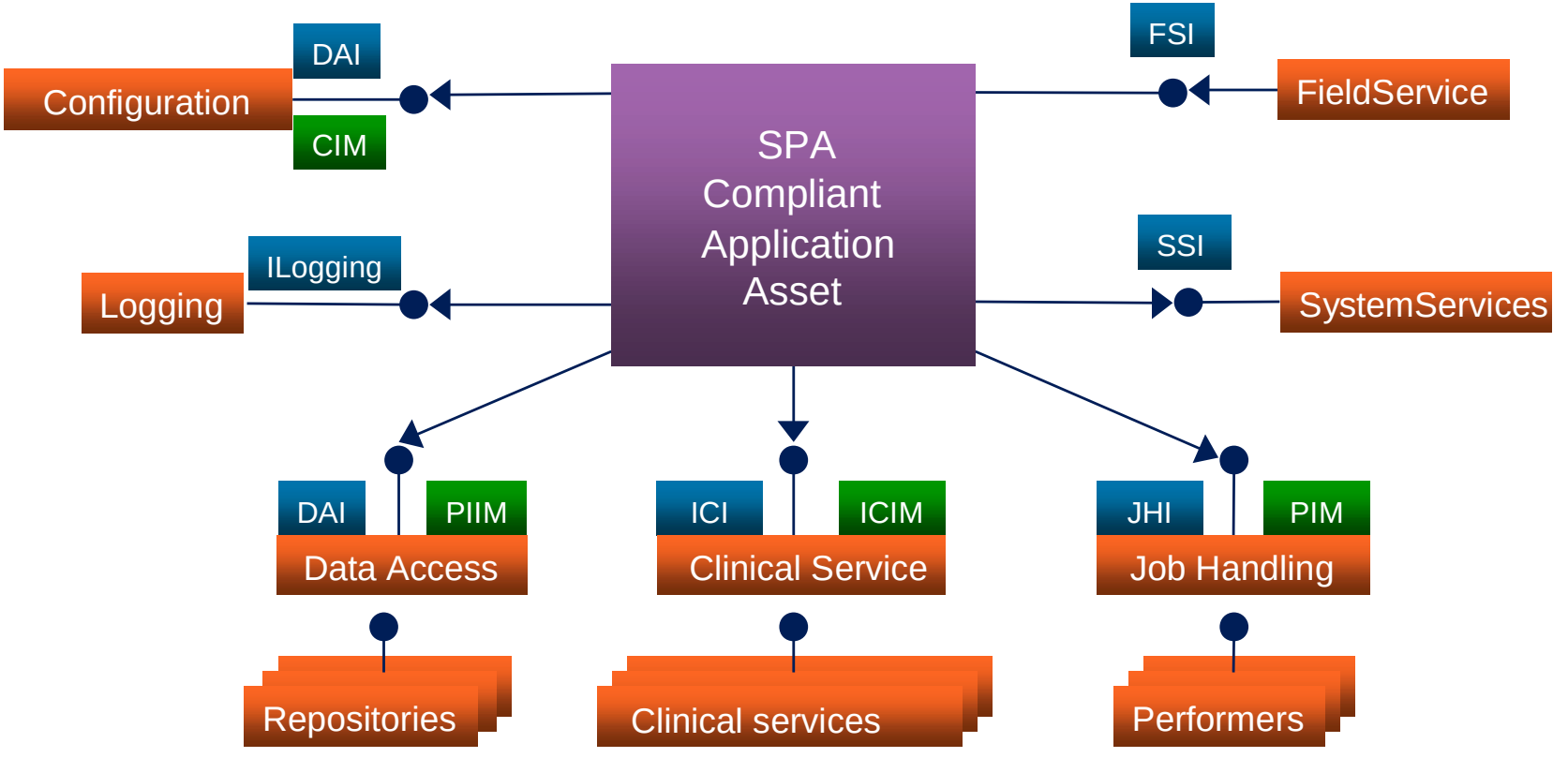
- Asset requirements have attributes for applicable variants (PC, MX, DX) and versions (3.1, 3.2, …)



Architecture approach:
product system architecture standardization

- Definition of PMS shared (software) product architecture (SPA)
  - Provided and required interfaces
  - Information models
- Definition of PMS-wide information models, enabling the exchange and sharing of medical (imaging) data and system configuration data
- Assets adhere to the agreed interfaces and information models, and can be used within other PMS products

Example:
product architecture standardization



Known issue 1:
Standards evolve

- Standards (interfaces) change over time
  - Sources:
    - New use-cases,
    - New requirements
    - Changing requirements
    - Changing/evolving external standards (DICOM)
- Policy
  - Introduce only backwards compatible changes
    - Additional interfaces
    - Data model extensions
  - Manage the changes
    - Organisation of change-control boards
    - Documentation

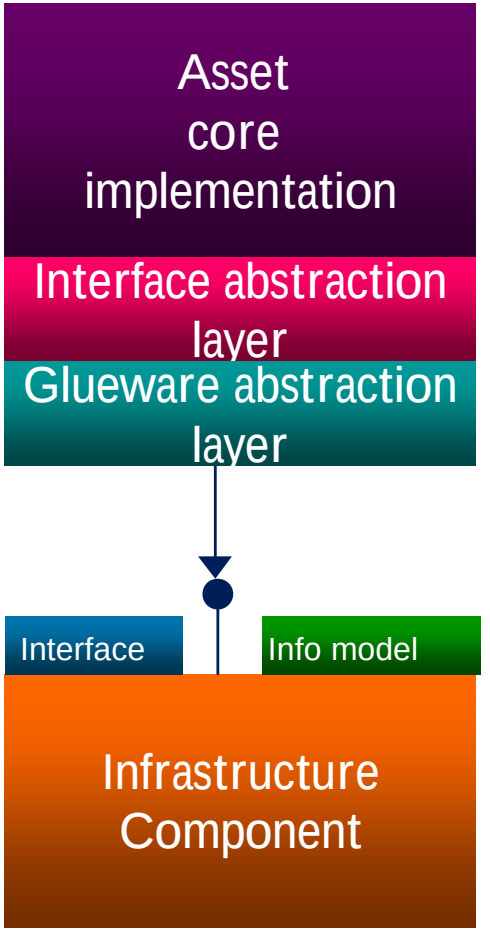
Known issue 2:
Implementation of the standard vary

- Interface implementations deviate from standards
  - Differences in standards interpretation (behavior!)
  - 'Bugs' in released systems
- Approach: introduce 'layers' between the asset and the infrastructure interface implementation
  - Internals of the software asset is based on 'abstraction layer' to the interface
  - Variations in interface implementation are managed in the *interface abstraction layer*.
    - Stable internal privately managed interface of 'standards' towards core asset component
    - Variation of 'real' standards is mapped to internal interface
    - Maps 'system plumbing' interfaces into convenient 'Asset API'

Known issue 3:
Usage of incompatible component technologies

- Components are created using various technologies
  - C/C++/Objective-c, Java, .NET, …
  - Where the systems integrate the tech
- Approach: middleware technology
  - COM as ‘glueware’ between technologies
  - ‘Glueware layers’ hiding the COM details to the assets
    - And removing the glueware whenever possible
    - Proven for various technologies
      - Java, .NET, C/C++/Objective-C

Interface abstraction layer



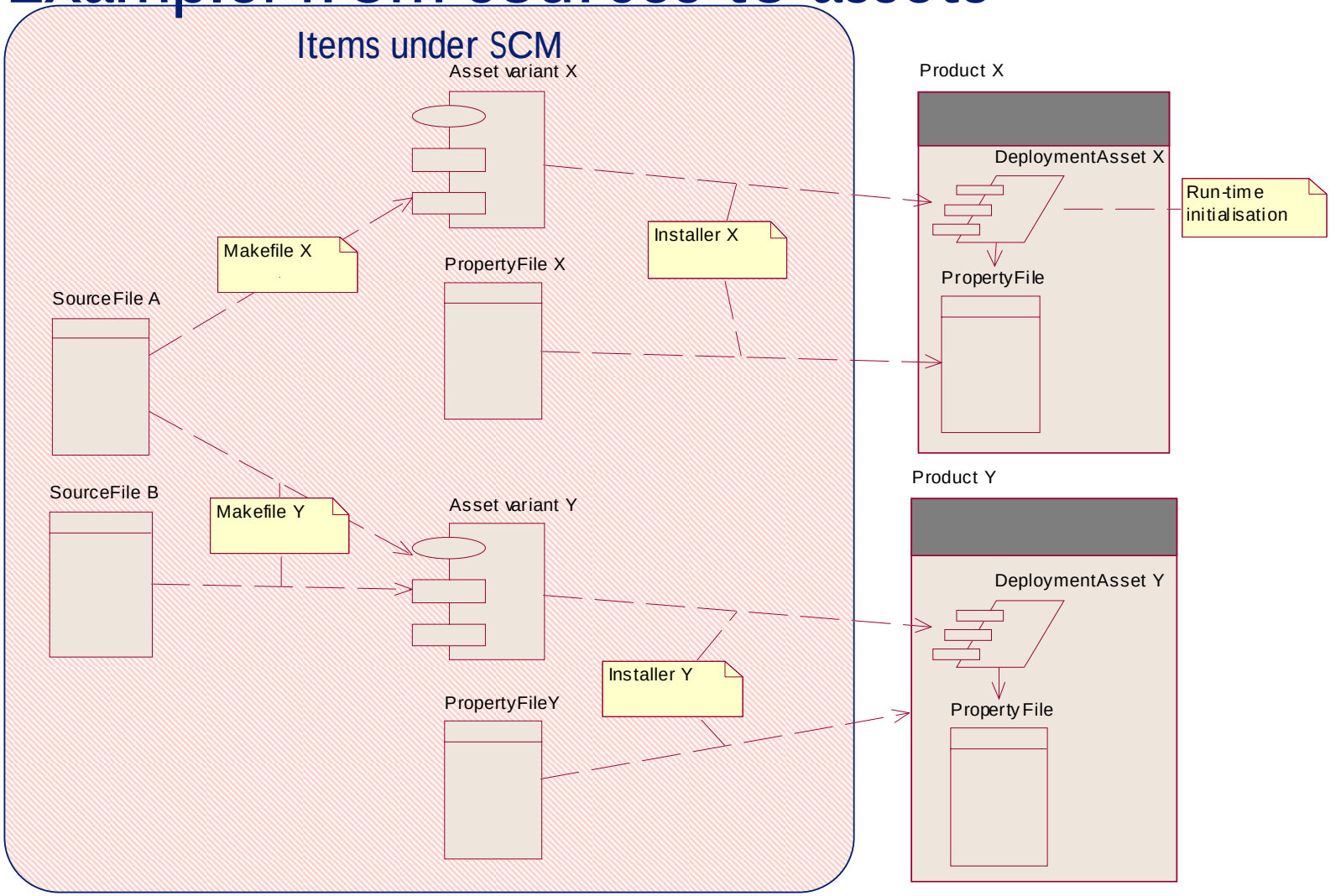
Known issue 4:
Side-by-side deployment and versioning

- Multiple versions of a ‘common’ library or component on a single system
  - Both ‘own’ components and 3rd party.
- Approach
  - Microsoft .NET assembly versioning
    - (only solves part of the problem)
  - We still need to gain experience here!

Architectural approach:
Software asset variant creation

- A single source-code base from which all assets are created
  - With identified *variation points* at different levels
    - Compile time (e.g. makefiles resulting in different executables or libraries for the variants)
    - Installation time (e.g. COM registration of different component variant, installation of different set of configuration property files)
    - Configuration time (e.g. product engineering, service or end-user configuration elements)
    - Run time (e.g. hardware availability based variation)

Example: from sources to assets



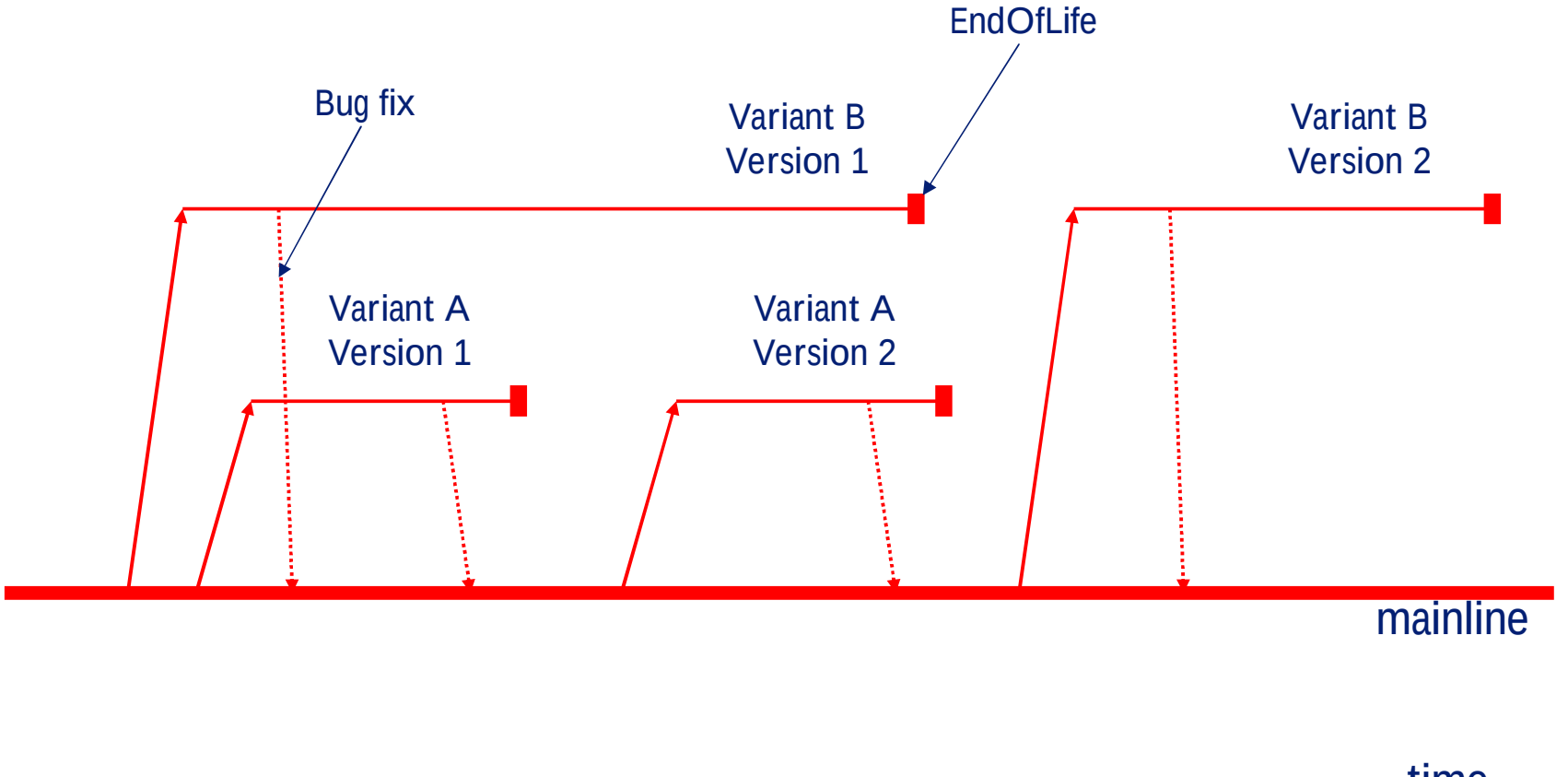
Architectural variation programming guidelines

- Program ‘capability-driven’ I.s.o. variant driven
  - Avoid explicit ‘version-checks’ in software
    - “If (version == SystemA) then Enable store function”
  - Instead program ‘capability’ or ‘property’ dependent
    - “If (database.IsWritable) then Enable store function”
    - Or: “If (Properties.EnableDatabaseWrite) then …”
- Design variation mechanism up-front
  - Choose the requ
  - Balancing flexibility and transparency
- Minimize amount of variant-specific code

Lifecycle Approach:
SCM variant version management

- Rational ClearCase version management of the released variant versions
  - As an identifiable branch of the sources
  - Self-contained, including the build- and installation environment
  - With tool support for merging fixes from one version to another
- In a single ‘mainline’ stream all variant changes are integrated to prevent diversions.

Example
SCM variant version management



Lifecycle Approach:
Project management

- Development projects delivering an update of multiple versions of an asset
- Issues
  - Product validation (testing)
  - Product documentation
    - Design, testing
    - Merging
- No “silver bullets”

Conclusion and Outlook

- Variation is inevitable, so be prepared.
- An integral approach to manage asset variation is required
  - Requirements Management
  - Product Architecture
  - Asset creation variation points
  - Software Configuration Management
- Useful commodity development tooling is available to assist the variation management.
- Project management of asset variant development is topic for future considerations.

Credits

Part of this work has been carried out in the context of a EU project

- Eureka! 2023 Programme,
  - ITEA project ip02009,
    - FAMILIES
- http://www.esi.es/en/Projects/Families/


PHILIPS