OpenVMS migration to –i4 and beyond

OpenVMS Bootcamp 2015

Migrating OpenVMS systems to Integrity -i4 servers and beyond

Colin Butcher CEng FBCS CITP
Technical director, XDelta Limited
www.xdelta.co.uk

xdelta

Agenda

Current Situation
Analysis and design
Example – data replication
Software engineering
Discussion

xdelta

Personal background

- Systems architect specialising in mission critical systems
- Engineering background
- Wide range of experience (satellite flight control, air traffic monitoring, finance data, healthcare, etc.)
- Started XDelta in 1996

XDelta – what we do

- Lead mission-critical systems projects
- Deliver world class services in demanding environments
- Strategic planning, technical leadership and project direction with clarity of vision and an eye for detail
- Systems engineering for availability and performance
- Ensure long term success through skills transfer

xdelta

OpenVMSmigration.com

- XDelta in collaboration with specialist organisations
- Complementary skills to solve difficult problems
- Strategic planning, technical leadership and project direction
- Ensure long term success through skills transfer
- Gartner (2009):
  - Identified XDelta as one of few companies world-wide capable of OpenVMS migration related projects

xdelta

Current situation (1)

Most OpenVMS systems purpose written:

- Tight integration with operating system and infrastructure
- Multi-site mission-critical capabilities
- Stable - running 24x7 with minimal disruption

Well engineered operating system:

- Well structured and documented
- Scales very well from small to large implementations

Distinct culture:

- People who like to understand things
- People who like to do things properly

xdelta

Current situation (2)

Where are you starting from ?

Where do you want to get to ?

There is a range of solutions, for example:

- Interim port to Integrity –i4, then to x86 – safe, sensible
- Wait for x86 to be ready – early adopter
- Run unsupported systems

xdelta

Current situation (3)

Evaluate your options:

- Changing platform is something you do infrequently
- New platform has to be stable for many years
- You will probably change platform again in 10+ years
- OpenVMS has a great track record of platform support
- What level of support do you need for h/w and s/w ?

An inherent conflict:

- Customers need stability of their chosen platform
- Customers need new hardware and new features without breaking stuff they rely on
- Computer industry wants to keep selling new stuff
- Computer industry doesn't like supporting old stuff

xdelta

Where to start ?

Don’t start in the wrong place:

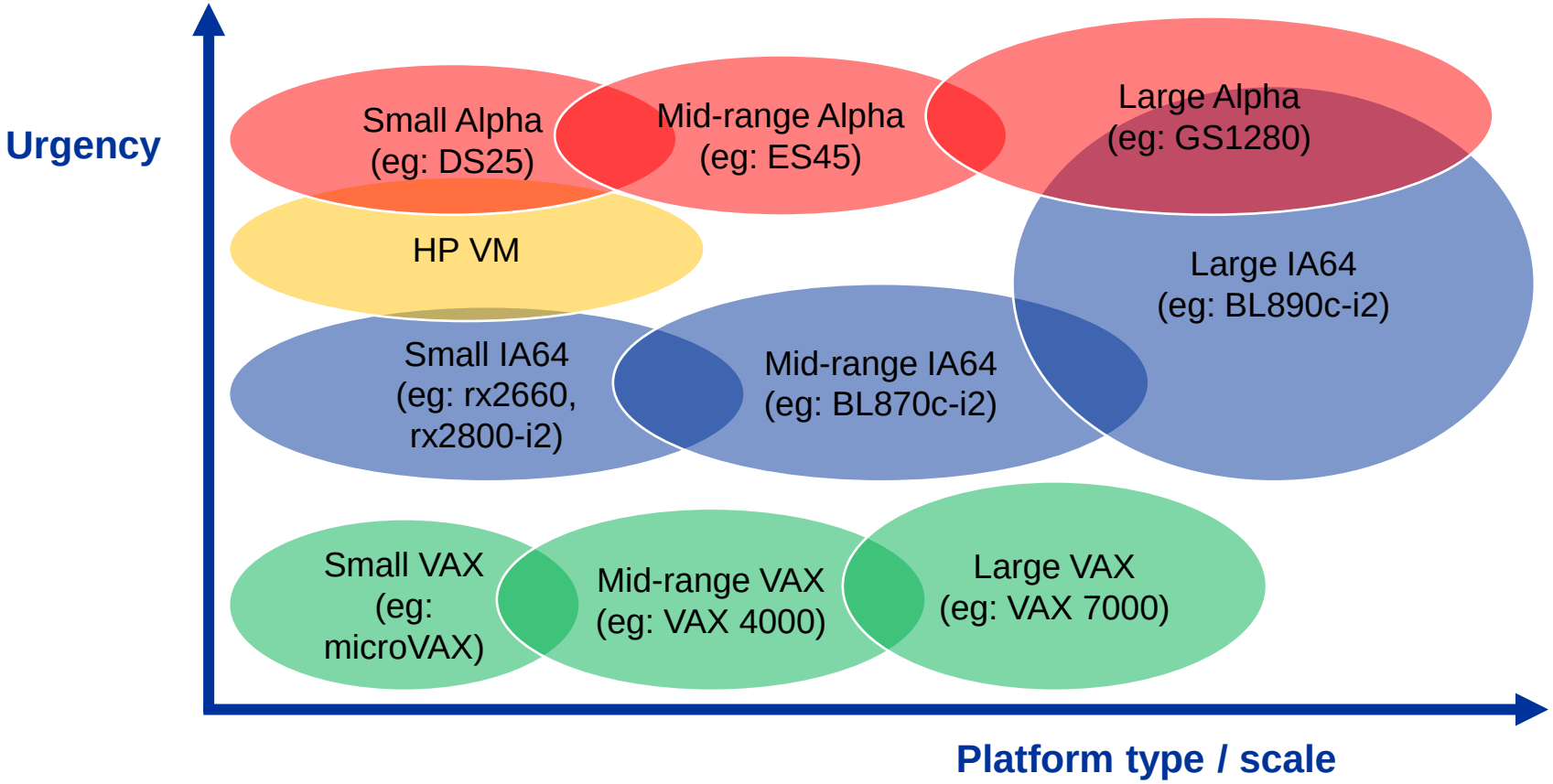
- Understand systems so they can be migrated without disruption
- It is a far bigger problem than just porting the code
- Use this as an opportunity to “do the right thing”
- Don’t rush, get started early

Migration

- Define criteria for success and how to meet them
- Design, plan and implement
- Test and demonstrate
- Transition and data migration with minimal disruption

xdelta

Urgency



xdelta

Complexity

Most systems have evolved over many years:

- PDP11 to VAX to Alpha to Integrity is not unusual

A range of languages:

- Basic, Fortran, Cobol, Macro32, C, C++, Pascal, Ada, Java, DCL, etc.

A variety of workloads:

- Interactive, Batch, Real-time, Backups, etc.

Set up in different ways:

- Naming conventions, Programming styles, DCL usage, etc.

People have moved on:

- Knowledge, Documentation, Design techniques, etc.

xdelta

Holistic approach

Look at the systems from a business perspective:

- Is our business constrained by technology ?
- Can the systems cater for business expansion and change ?
- How can we avoid disruption when we make changes ?
- Are the systems stable and maintainable ?
- How can we exploit technology to get a competitive advantage ?
- What could we do differently / better ?
- How can we make best use of the investment we make ?
- Do we have the resources, budgets and time ?

xdelta

Overview (1)

Understand the business environment

- How might the systems need to adapt and evolve

Define criteria for success

- Security, availability, performance, data access, etc.

Plan, implement, test and demonstrate

- Thinking ahead, temporary changes, simulation, preparation

Undertake transition with minimal disruption

- Data migration, back-out plans, avoid big-bang

xdelta

Overview (2)

Migrating between platforms involves many things, eg:

- Understanding all aspects of existing systems
- Having all the code
- Having the build environment for the whole system
- “Going native” with the target platform
- Avoiding non-portable code, unless absolutely necessary for performance and availability
- Taking advantage of new capabilities in the platform and infrastructure
- Breaking the migration down into manageable pieces
- Designing the transition process
- Designing how to test the new systems
- Migrating archived data

Overview (3)

Migrating between platforms may also require:

- Re-architecting the applications
- Taking existing defects with you
- Lots of experimentation
- Learning new things
- Temporary software to make the transition easier
- Temporary equipment to provide additional resource
- Making changes to the existing systems first
- Accepting that some things will be different

System architecture

Platform irrelevant

- Business processes and logic flow
- Business requirements for security, availability, performance, expansion, connectivity, etc.

Platform independent:

- To what extent can we isolate implementation from platform ?
- Abstraction layers to hide platform specific details
- Choice of languages, etc.

Platform specific:

- Mechanisms that implement functions (eg: volume shadowing)
- Requires detailed understanding of behaviour

OpenVMS features

Commonly used features:

- Shared-everything clustering
- Multi-site disaster-tolerant clusters
- Host-based volume shadowing
- RMS indexed files
- DECnet (internal and external)
- Logical names
- Shared memory regions and caches
- ASTs, mailboxes, etc.

Consistent architectural design:

- Calling standard
- System services
- Run-time libraries (language specific)

xdelta

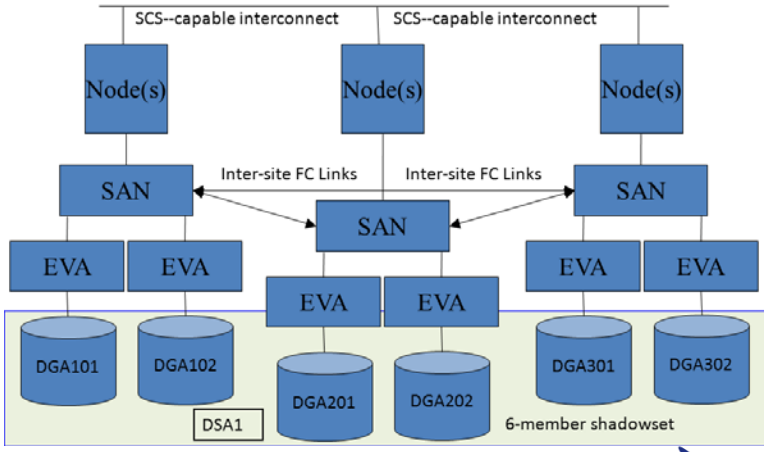
Holistic thinking

Data replication is a good example:

- Necessary for many reasons – why do we need it in this case ?
- Wide range of possible ways to achieve it
- How many techniques can you think of to replicate data ?
- What advantages / disadvantages come with each technique ?
- Is there a “best technique” ?

Some ideas:

- Just copy files
- Application based
- Database mechanisms
- Operating system based
- Storage infrastructure based



xdelta

Analysis

Understand the purpose of the systems:

- What do they do ?
- How do they work ?
- Why was it done that way ?

Reverse engineering:

- Get inside the head of the original designers
- Understand flows and structure:
- Identify key aspects and business logic of systems

Functionality mapping:

- Identify key features that require equivalent mechanisms

Re-engineering:

- Meet business needs for long term

xdelta

Code conversion

Code analysis and conversion:

- Don’t worry about how much code there is
- Most code uses a subset of available features
- Largely a problem with “old”, less portable languages
- Language conversion may be less difficult than you think
- Maintain the target code, not the original source
- Be able to move it again in the future

Abstraction layers:

- Hides platform specific functions

Code re-factoring:

- Removes “dead” code and cleans up flow of control
- Improve performance and testing

Clean up the code

Code re-factoring:

- Make code more maintainable
- Make code more portable
- Restructure code and improve layout for readability
- Remove “dead” code
- Improve documentation
- Use standard language features
- Check compilation on multiple platforms
- Use abstraction layers
- Make changes on source platform first

Code migration to VMS

Code migration – language specific:

- Run-time libraries with standard interfaces

Code migration – platform specific:

- System services (eg: locking, process creation, logical names)
- IO operations (eg: ASTs, SMG routines)

Code migration – file system and file naming:

- Mapping of file naming conventions
- Data migration and data types

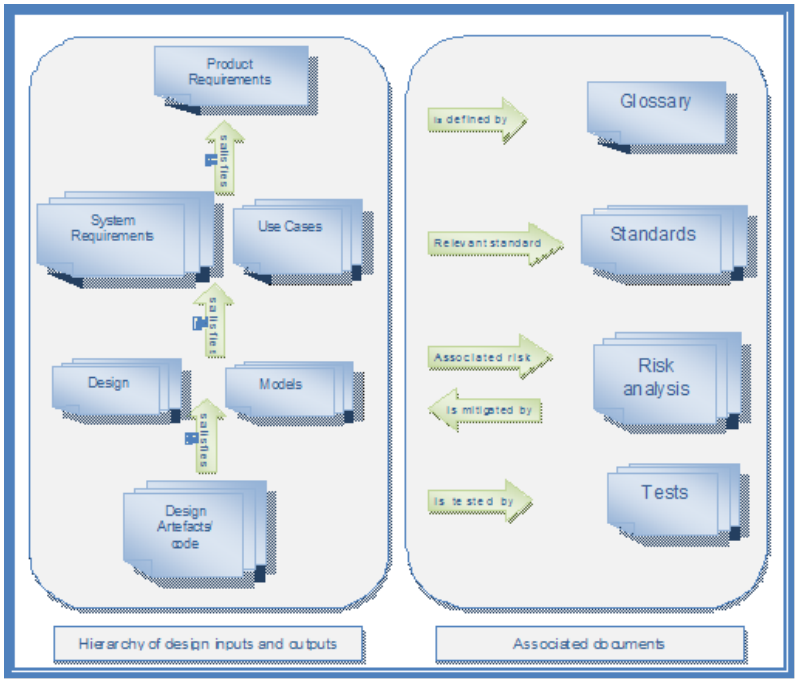
Software engineering

Requirements:

- Product tested against requirements
- Affordances and constraints
- UML definitions developed from Use Cases

UML:

- Standardised language for description of systems
- Provides range of diagrams to explore system's architecture
- Define and experiment with static and dynamic aspects of system



xdelta

Abstraction layers

Encapsulate the way something works:

- Maintains a consistent interface to the main code stream
- Allows the detailed operations to change behind the scenes

An aid to portability:

- Common set of routines used by applications
- Makes it easier to replace an operating system call
- Makes it easier to replace a file system operation or run time library function when changing from one language to another

Hide the underlying behaviour:

- Design and problem solving requires a complete understanding

Code analysis (1)

Analysis:

- Identify language specific run-time library calls
- Identify system service calls
- Identify file system and IO calls
- Identify subroutine calls and macros
- Examine and display code structure
- Derive “use cases”

xdelta

Code analysis (2)

Benefits:

- Most code uses a subset of available features
- Recognise patterns within code
- Conversion may be less difficult than you think
- Improve code performance
- Improve testing of code

Scale and complexity:

- Most code will not be a problem – maybe 90% or so
- Some code will be difficult
- Don’t worry about how much code there is

Code analysis (3)

Example techniques:

- Search tools
- Editor learn sequences, eg: TPU
- Parser tools, eg: LEX, YACC, Flex, Bison, etc.
- Purpose-written small programs, even in DCL
- Open Source tools and compilers

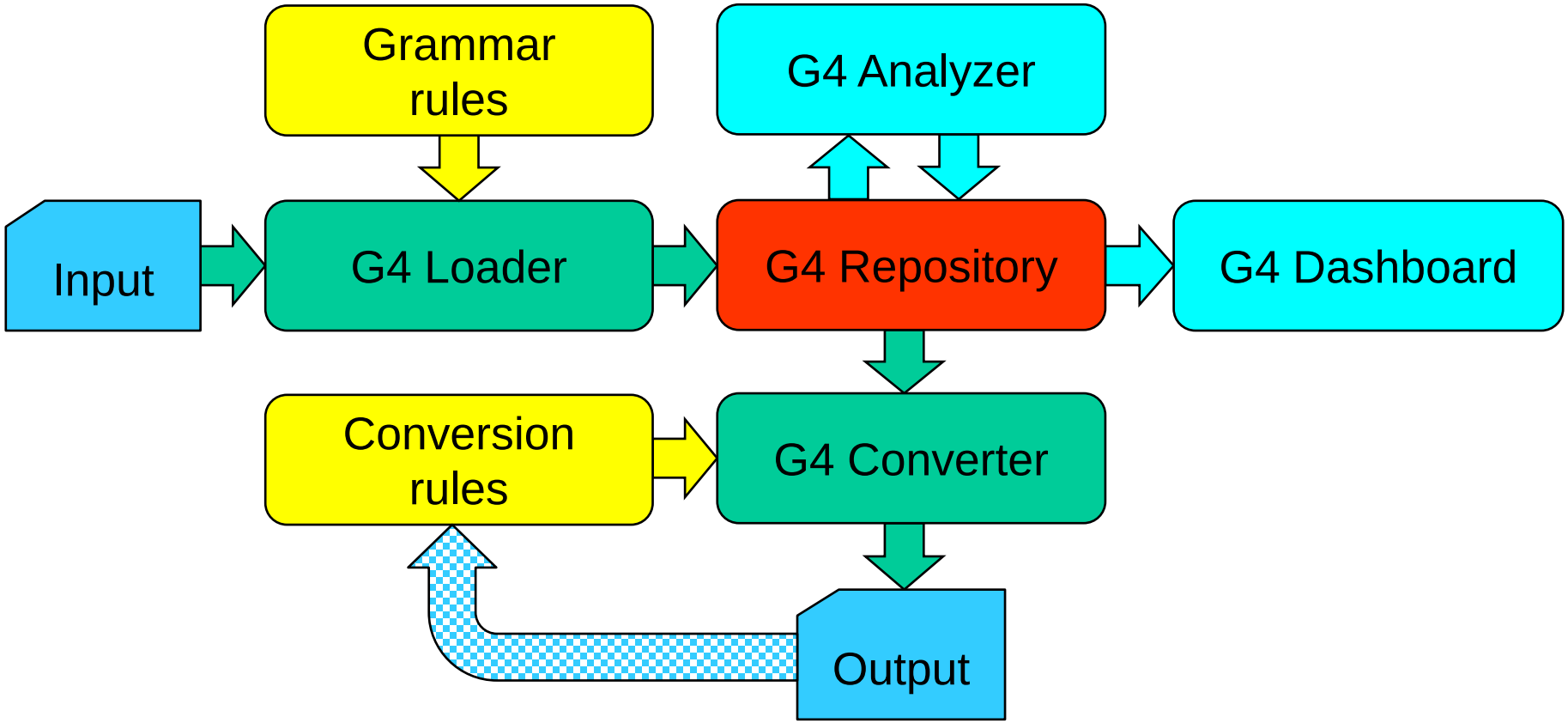
Commercially available tool vendors:

- Analysis, call trees, flow of control, etc.
- Syntax mapping based on language grammars
- Automated conversion from one language to another
- Needs libraries of routines to implement equivalent and new functions, not necessarily to implement OpenVMS-like behaviour

xdelta

Cornerstone G4 Platform

Cornerstone Technology

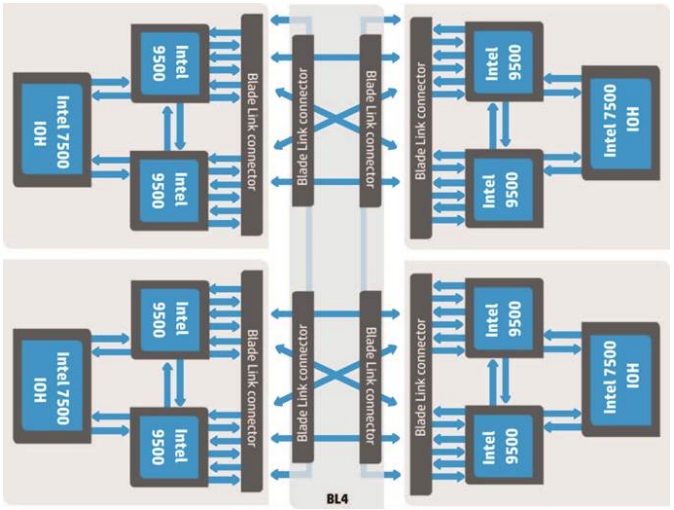


xdelta

High performance

High performance systems:

- Native hardware for best performance
- Virtualisation - no
- Emulation - no



Performance constraints:

- Bandwidth and Latency
- Achievable IO operations rate
- Parallelism within the applications
- Memory usage and NUMA effects

Limiting factors are often outside the system itself:

- Network infrastructure
- Storage arrays and SAN fabric infrastructure
- Inter-operability with other systems

xdelta

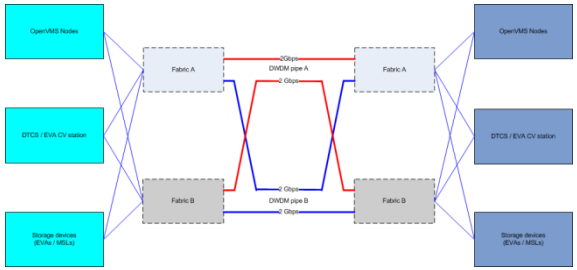
High availability

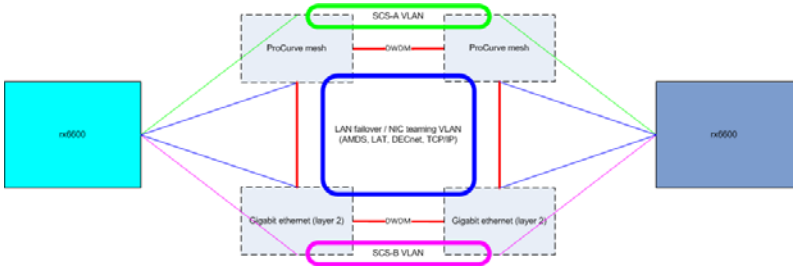
High availability systems:

- Carefully integrated with a consistent overall architecture
- No single points of failure
- Make best use of the capabilities of the individual components
- Understand the whole system and write as little code as needed

How to migrate with minimal disruption ?

The closer you get to zero downtime, the harder it is!





xdelta

Experience – big data

Big data:

- Was GS1280s / EVAs, now BL890s / EVAs / 3PAR
- Multiple sites, many nodes, OpenVMS core, Linux data distribution layer
- Very tightly integrated, written in several languages

First move: GS1280 to BL890-i2:

- Split up applications and move much of the functionality out to the Linux data distribution layer
- Re-structure the systems to accommodate intended growth.
- Moving core applications from GS1280 to BL890-i2 took 18+ months.
- Performance engineering was key:
  - How to get BL890 to outperform GS1280?
  - How to get parallelism and scalability in Linux data distribution layer?
  - How to get IO throughput (disc and network)?

xdelta

Experience – split-site OpenVMS cluster

Healthcare:

- Database on OpenVMS Integrity (rx2800's)
- Client systems and desktops across country, many of them mobile
- Multiple sites, 4-way HBVS, application fails over to alternate node
- MSA storage, tape backup
- Mix of terminal emulators (systems management) and virtualised applications / web applications to standardised desktop
- Intermediate layer to run applications with database on OpenVMS

Gradual evolution:

- Was regional, became national (merged data)
- Was Alpha, moved to Integrity, then replaced hardware with new
- Moving some application functionality off VMS to middleware
- Rewriting VMS specific code in Java or C++
- Planning ahead for change with minimal disruption

xdelta

Project approach

The holistic approach is essential:

- Business impact and risk
- Architectural structure
- Design and planning
- Transition options
- Support options
- Inter-relationships and complexities
- Collaboration with your suppliers

xdelta

Project delivery

Extra workload and different skills:

- It will consume resources, so start assessment and planning
- You may not have all the skills and/or the time you need
- Get started, take the time to do it well

xdelta

Don’t run out of runway …

xdelta

OpenVMS migration to –i4 and beyond

OpenVMS Bootcamp 2015

Thank you for your participation

Colin Butcher CEng FBCS CITP
Technical director, XDelta Limited
www.xdelta.co.uk

xdelta