



Programming on OpenVMS

Observations and Best Practices

Brett Cameron, presented by Gerrit Woertman

© 2011 Hewlett-Packard Development Company, L.P.
The information contained herein is subject to change without notice



Abstract



The presenter has been maintaining, developing, and modernizing business-critical applications on OpenVMS, UNIX, and Windows for 23 years and has been helping customers to do the same for almost as long. During this time the presenter has learnt a lot of lessons the hard way and has encountered the good, bad, and the ugly of software development exploits on the OpenVMS platform (and other platforms for that matter). The presenter firmly believes that software development does not have to be as difficult as it often seems to be, and if a few key fundamental principles are adhered to and the project team has at their disposal the right tools (and knows how to use them correctly), it should be possible to achieve positive results without too many unpleasant surprises along the way. During this talk, we will consider some of these fundamental principles (which are common to all software development projects) and we will take a whistle-stop tour of some of the tools that are available to developers on the OpenVMS platform, and how to best leverage those tools.

Agenda



- Introduction
 - Software engineering history lesson
 - Things go wrong
 - Software engineering principles
- So you're planning on implementing a new OpenVMS-based software solution (or modernizing an old one)
 - Some key considerations
 - Some observations and suggestions
 - The good, the bad, and the ugly
- Some specific comments on developing applications for OpenVMS
- Questions

Introduction (1)



- Software Engineering and Computer Science are still young fields, but they have seen remarkable developments during the last four decades
 - Present day software applications are characterized quite differently when compared with early computer programs:

Early programs	Present day applications
Quite small	Large
Written by one person	Developed by teams over long periods
Written by an expert in the application area	The programmers have no expert knowledge of the application area
Used by experts in the application area	The programmers are not the end users
Solved technical problems	The problems addressed concern everyday life
Input was numeric (from punched card or tape)	Input can be in a huge variety of forms
Output was numeric (printed)	Output can be in many media
Run on-line	Run in real-time

Introduction (2)



- Despite the remarkable developments in software since the 1950s, the techniques used to produce software have not kept up!
- Some of the problems are:
 - Many people still consider programming an art
 - The personalized nature of many programs has made maintenance very difficult
 - Many programmers have not been formally educated in Software Engineering
 - Management often does not appreciate that software projects cannot be managed like other manufacturing projects

The *software crisis* (1)



- The software crisis was first identified in the late 1960s
 - It alludes to a set of problems encountered in the development of software:
 - Problems associated with how we develop software
 - How we maintain a growing volume of existing software
 - How we expect to keep pace with the growing demand for more software
- Use of the term software crisis can be criticized as being melodramatic
 - But the phrase does serve a useful purpose by highlighting the real problems found in all areas of software development
- The software crisis is characterized in many ways:
 - The schedule and cost estimates of software projects are often grossly inaccurate
 - The productivity of software developers has not kept pace with demand
 - Quality of software is sometimes less than adequate

The *software crisis* (2)



There is no single best approach to a solution for the software crisis. But, by combining comprehensive methods for all phases in software development we can achieve a discipline for software development.

These methods include:

- Better tools for automated development
- More powerful building blocks for software implementation
- Better techniques for software quality assurance
- An overriding philosophy for coordination, control and management

The software crisis (3)



In response to the software crisis, the term *Software Engineering* was coined at a NATO conference in 1968. This was in the belief that software design, implementation and maintenance could be put on the same footing as traditional engineering disciplines. The conference concluded that Software Engineering should adopt the philosophies and paradigms of established engineering disciplines, and this would solve the software crisis.

IEEE definition of software engineering:

“Software engineering is the systematic approach to the development, operation, maintenance, and retirement of software.”

However, the software production process is not like traditional engineering. Some of the unique features of software are:

- Software is logical rather than physical, and therefore has characteristics that are considerably different from those of hardware
- Software is developed or engineered; it is not manufactured in the classical sense
- Software costs are concentrated in engineering
- Software does not wear out, but it does deteriorate
- Most software is custom built, rather than assembled from existing components

The software crisis (4)



Does the Software Crisis still exist?

Robert L. Glass, author of *Software Runaways*, believes it does not. For every widely publicized software failure, there are several unpublicized successes. His book contains an interesting collection of articles about runaway software projects.

Glass has extensively surveyed many runaway projects and has found that the main reasons for failure are (in order of decreasing importance):

1. Project objectives not fully specified
2. Bad planning and estimating
3. Technology new to the organization
4. Inadequate project management methodology
5. Insufficient senior staff on the team
6. Poor performance by suppliers of hardware/software
7. Performance (efficiency) problems

The software crisis (5)



- Okay, so what is a Software Runaway?
 - *“A project that goes out of control primarily because of the difficulty of building the software needed by the system”*
- ... or KPMG’s definition:
 - *“A runaway project is one which has failed significantly to achieve its objectives and/or has exceeded its original budget by at least 30%”*

Either way, it’s something to be avoided!

Software engineering principles



People often say that software development knowledge has a 3-year half-life: half of what you need to know today will be obsolete within three years. This is probably right in the domain of technology related knowledge, however there is another body of software development knowledge that will stay relevant throughout your career – the *software engineering principles*.

In 1987, Fred Brooks published a very influential article, "*No Silver Bullets – Essence and Accident in Software Engineering*". Its main thrust was that no single tool or methodology suggested a significant improvement in productivity over the next decade. Brooks argued that major gains in the accidental elements of software engineering (high-level languages, powerful development environments) had already been made, and any further improvements in productivity would have to come from addressing software's essential difficulties: the complexity, conformity, changeability, and invisibility inherent to software development.

Knowledge that addresses these difficulties can be thought of as *software engineering principles*. Since the first NATO conference on software engineering in 1968, the software industry has come a long way in identifying the essential knowledge that a software engineer needs in order to develop software effectively.

This essential body of knowledge includes concepts such as requirements analysis and development, design, construction, testing, reviews, quality assurance, software project management, algorithms, metrics, user interface design, and so on.

"As a software development professional, you need knowledge of specific technologies to do your job. But you need knowledge of software engineering principles to do your job well".
Steve McConnell – Editor IEEE Software Magazine.

Something else to think about



Once upon a time, it was possible to create useful software with nothing more than a text editor, file management commands, and a compiler. As hardware and software capabilities have grown, so have the scope and complexity of the business objectives software addresses. These trends have led to a dramatic increase in the number and variety of tasks and tools required to create business software. Modern software development is so complex that it requires all contributors to use several processes and tools for managing the development process itself.

Joe Niski, Burton Group, 2007

Agenda



- Introduction
 - Software engineering history lesson
 - Things go wrong
 - Software engineering principles
- So you're planning on implementing a new OpenVMS-based software solution (or modernizing an old one)
 - Some key considerations
 - Some observations and suggestions
 - The good, the bad, and the ugly
- Some specific comments on developing applications for OpenVMS
- Questions

Some key considerations (there are many)



- Weighing up replacement or modernization...
 - Does the application do the job?
 - Functionality
 - Performance
 - Maintainability
 - Capacity
 - Ability to integrate with other systems
 - TCO
 - Do you actually know how the numbers stack up?
 - Return on investment
 - Availability of skills
 - End-of-life technologies

All of these factors are in one way or another related

What am I seeing (the good)



- Some good does seem to have come from the economic crises...
 - More customers are looking to get more out of their existing applications
 - Modernisation
 - Integration (old with new)
 - Postponing large package implementations
 - Larger vendors are starting to provide a wider range of (cost-effective) solutions
 - Small/medium business as well as enterprise-level solutions
 - More and better cloud solutions
 - Seems to be a little more creativity and ingenuity being exhibited
 - Increased use/adoption/acceptance of Open Source components
 - This can be a complex area
 - Licensing considerations
 - Support
 - Updates

What am I seeing (the not so good)



- Still not enough lateral thinking by many customers
- Too many sheep
 - Too much listening to so called experts
 - Do your own research
 - You employ smart people, right?
 - Approach anything that sounds too good to be true with the scepticism it all too often rightly deserves
 - If you need help, engage the services of genuinely impartial and suitably experienced consultants

What am I seeing (the not so good)



- People have a very bad habit of assuming that things need to be much more complicated than is in fact the case
 - And they go and implement unnecessarily complicated and ridiculously expensive solutions accordingly!
 - Organizations waste huge amounts of money on IT simply because they think that's what they need to in do order achieve anything truly useful
 - The KISS principle always applies
- Analysis paralysis
 - When you decide to do something, get on with it!
 - Accept that things will never be perfect
 - You're better to have something that mostly does what you want than to have nothing (or just a nice pile of pretty documents)

So what should you be doing?



- Consider the points on the previous slide(s)
 - Does the application do what it needs to do, and does it do it well?
 - Is it maintainable?
 - It is cost-effective?

Answers to these and related questions will drive your modernization/replacement strategy...



It's not all about technology...



- When implementing a new system or a major enhancement to an existing system...
 - Do your homework
 - Nail your requirements (functional and non-functional)
 - You can't implement a "vision"!
- Retain business knowledge
 - Hiring in short-term contractors to document and understand your business requirements generally does not work
 - Involve end users (the business) throughout the process
- Ask "stupid questions"
- Involve the right people in negotiations with vendors/suppliers

It's not all about technology...



- There are always risks in any project
- It's a partnership
 - All parties want to see a successful outcome!
 - And when there's a problem, remember "no one is innocent"
- Ensure that proper governance structures are in place
 - Projects invariably do not fail for technology reasons (unless it's the wrong solution in the first place)
- Let the vendor get on with the job
 - If you know better, why did you engage the supplier to do the work in the first place?

Agenda



- Introduction
 - Software engineering history lesson
 - Things go wrong
 - Software engineering principles
- So you're planning on implementing a new OpenVMS-based software solution (or modernizing an old one)
 - Some key considerations
 - Some observations and suggestions
 - The good, the bad, and the ugly
- Some specific comments on developing applications for OpenVMS
- Questions

General comments



- Good tools are important, but not as important as good people!
- But if we must talk about tools...
 - Use flexible tools
 - Eliminate duplicate tools
 - Ensure developers know how to use the tools
 - Invest in some training if necessary
 - Use tools in a consistent manner
- What sort of tools are we talking about here?
 - Compilers
 - Build tools
 - Source code management
 - Testing tools
 - IDE's
 - ...

A few more general comments



- Source code control systems
 - Use one... and use it properly!
 - CMS is okay if working only on OpenVMS
 - Mercurial, CVS, or SVN otherwise
 - Things like SVN and Mercurial work well with NetBeans and NXTWare
 - I still quite like CVS...
- Build tools
 - DCL
 - MMS/MMK
 - Other
 - ANT (if you're working in Java), or some custom built tool
 - But whatever you have, make sure that your build procedure can rebuild the entire application and run it regularly!

Integrated development environments

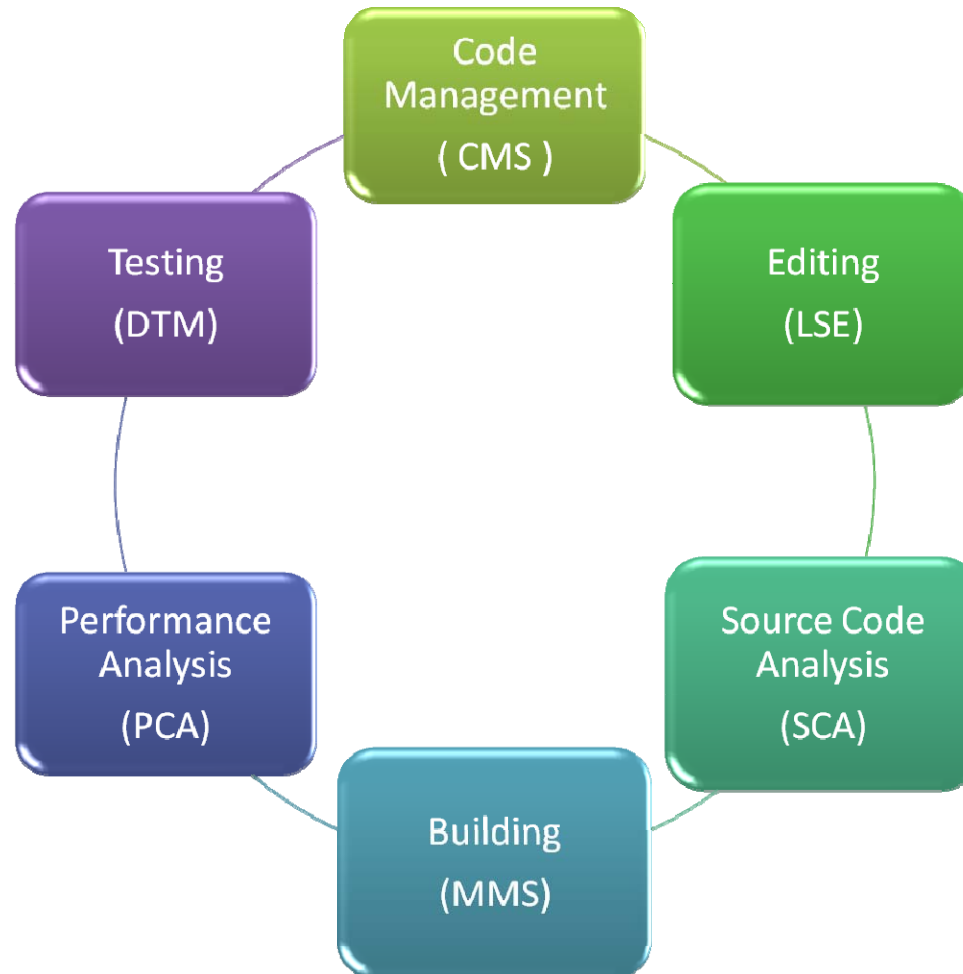


- DECset
 - The traditional IDE for OpenVMS
 - More than adequate for most projects
 - Can struggle a bit with some newer software development technologies and with distributed, heterogeneous development environments
 - Assumes a good level of OpenVMS knowledge
 - And frankly if you're developing software for OpenVMS then your team should have such knowledge!
- Modern Integrated Development Environments (IDE's)
 - Distributed NetBeans
 - NXTWare Remote from eCube Systems (Eclipse-based)
 - Can help to address limitations of DECset
 - Can enable developers with limited OpenVMS skills to develop applications
 - But frankly, if you're developing software for OpenVMS then your team should have good OpenVMS skills!

Integrated development environments



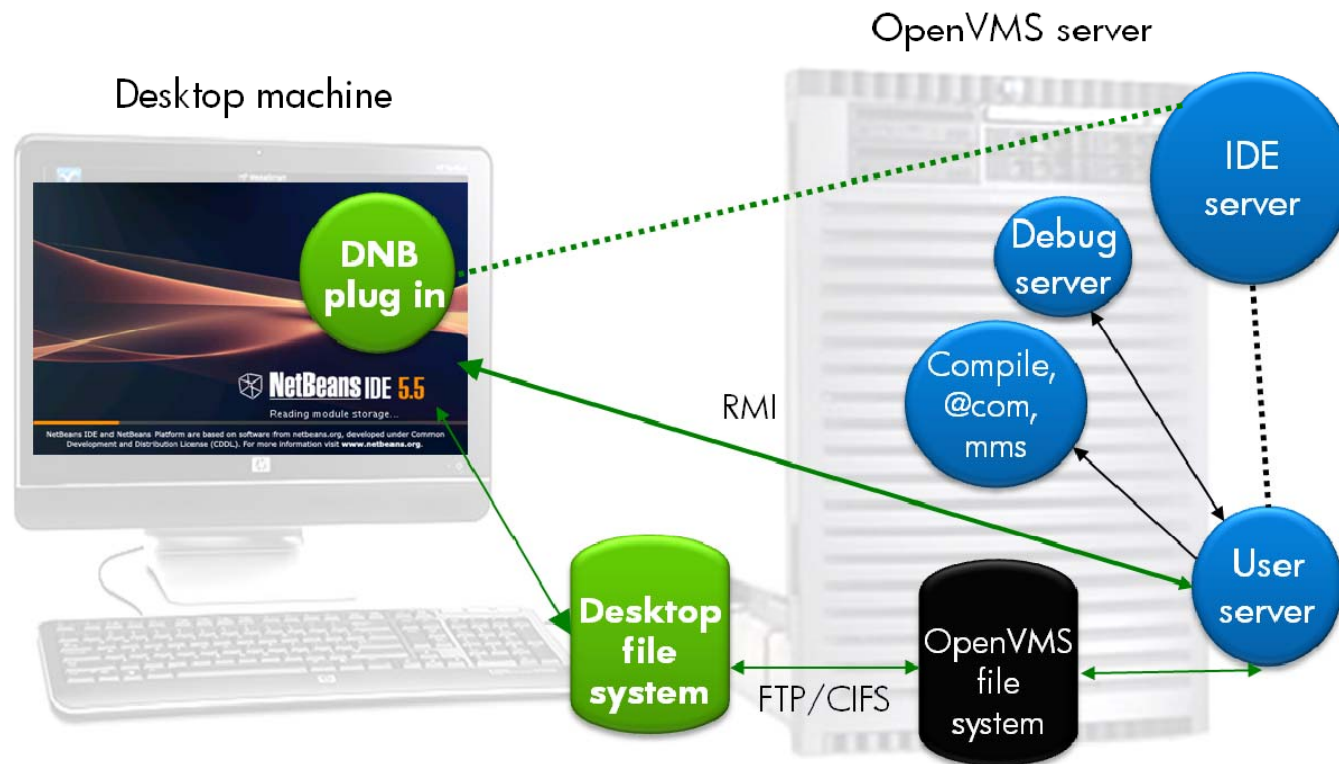
- DECset – the traditional OpenVMS IDE



Integrated development environments



- Distributed NetBeans...



- Editing with syntax highlighting
- Remote compilation/execution
- Error navigation
- Remote debugging

GNV



- Unix-like environment
 - Bash Shell
 - Many UNIX commands (ls, cp, su, ...)
 - Utilities (tar, make, vi, gcc, ...)
 - Utilities "tweaked" for OpenVMS (mnt, umnt, ...)
 - Latest sources available on <http://sourceforge.net/projects/gnv/>
 - Available for both OpenVMS Alpha as well as IA64
 - Future plans to support for more commands and utilities and to improve what's already there
- Can be useful if you have a number of UNIX/Linux developers on your team
 - But try to use it as a vehicle to introduce them to OpenVMS
 - Try to avoid using the GNV environment long-term
- Can be useful for porting
 - Again, try to avoid using the environment long-term



A few fundamentals...



- Compile code with `/list/machine`
- Link `/map/full`
- Be aware of what is provided by OpenVMS
 - OpenVMS RTL routines
 - CRTL (or other language RTL)
 - System services
 - Utility API's
 - Object libraries, shareable images
 - Message files
 - ...
- Understand compiler and linker qualifiers
- If working in a mixed-language environment be sure to understand differences in things like passing mechanisms, data types, structure field alignment, and so on

A few more fundamentals...



- When testing...
 - Have things properly set up to capture process dumps (they can be really useful)
 - Use something like T4 to record system data, and review such data
 - Keep an eye on things like:
 - Alignment faults (nasty, bad, evil things)
 - Excessive logical name translation
 - ...
- Judicious and proper use of logical names
- When implementing API's, consider compiling with `/POINTER_SIZE=64`
 - Can require some code changes, but generally minimal
- Don't ignore or disable compiler warnings
 - Try to understand what the compiler is complaining about and address it

Agenda



- Introduction
 - Software engineering history lesson
 - Things go wrong
 - Software engineering principles
- So you're planning on implementing a new OpenVMS-based software solution (or modernizing an old one)
 - Some key considerations
 - Some observations and suggestions
 - The good, the bad, and the ugly
- Some specific comments on developing applications for OpenVMS
- Questions

Summary



- Some random comments to wrap up...
 - Good people and proper processes are more important than technology
 - Invest in training
 - Retain knowledge
 - Understand your requirements
 - Know how to use your tools
 - And ensure that you have all the necessary tools
 - Don't make things more complicated than they need to be
 - Have fun!

Questions?





i n v e n t