



OpenMake
Software

*Bridging the gap between
development efforts and production results*



Separating Duties to Meet It
Compliance

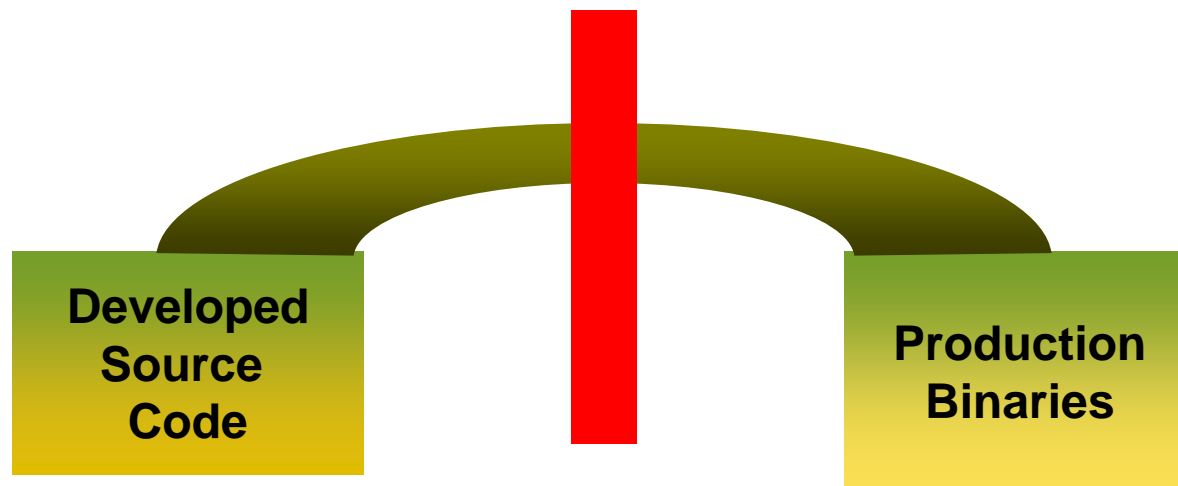
How the Big Boys Do It!

Presented by Tracy Ragan,
COO OpenMake Software



“Throwing it Over the Wall”

Source Code





Separating Duties



Developers

Duties – Write software and perform Unit Test. Check-in Code and request approval for release.

Real World – Developers inform production control that they are ready for a release. They perform the production build, execute a system test and manage the production release.

Production Control

Duties – Perform Builds and installs for QA and Prod environments.

Real World – No Knowledge of the construction of distributed production binaries. Rely on developers for builds and releases. May execute a job the developers have given them – push button without knowing what the button does.





What Separates Duties

- The ability for developers to simply hand off source code to production control.
- The ability for production control to create binaries from “empty” directories using SCM managed code, and perform the install.
- The ability for SCM to match source code to binaries completing the whole picture.



Throwing it Over the Wall

- Lets look at the Big Boys
 - Mainframers. Yes, these applications run far more technical systems than distributed platforms. They have figured it out.
 - Processors – provided by CA Endevor or Serena ChangeMan. Processors removed the reliance on ad hoc compile JCL. All binaries are created the same using a best practice processor. Production Control manages the Processor.
 - *Myth Buster - Mainframe and Unix builds are less complex than C, Java or .Net on the distributed platform.*
 - *Not true- Mainframe builds can call more complex compile and link actions than Java, C or .Net.*



Throwing it Over the Wall

- UNIX Environments

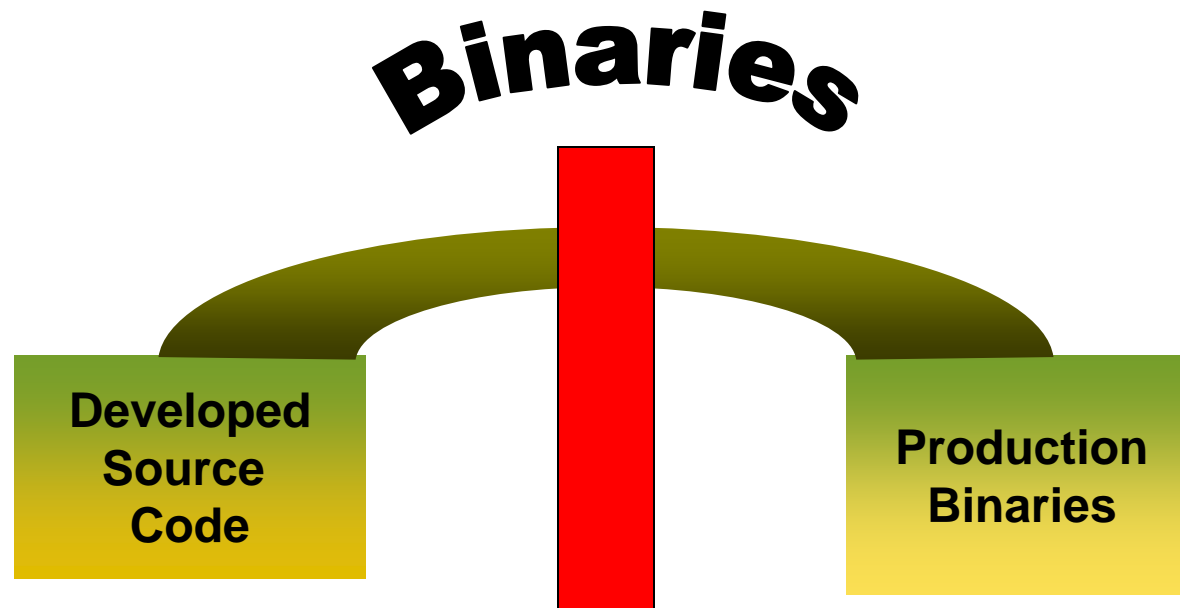
- Although not as standard as the Mainframe, ClearCase and ClearMake made a substantial change in UNIX development processes.

- Like Processors, Unix Administrators embraced the coding of a single large ClearMake makefile, integrated with ClearCase to create a very similar environment to mainframe processors.
 - The standardizing of the ClearMake scripts made it possible for Unix Administrators to take control of the ClearMake scripts, versus relying on the developers to provide the expertise.



What is Missing in the Distributed Development Processes?

A repeatable Build Process that allows Source code to be “thrown over the wall” and compiled by Production Control.





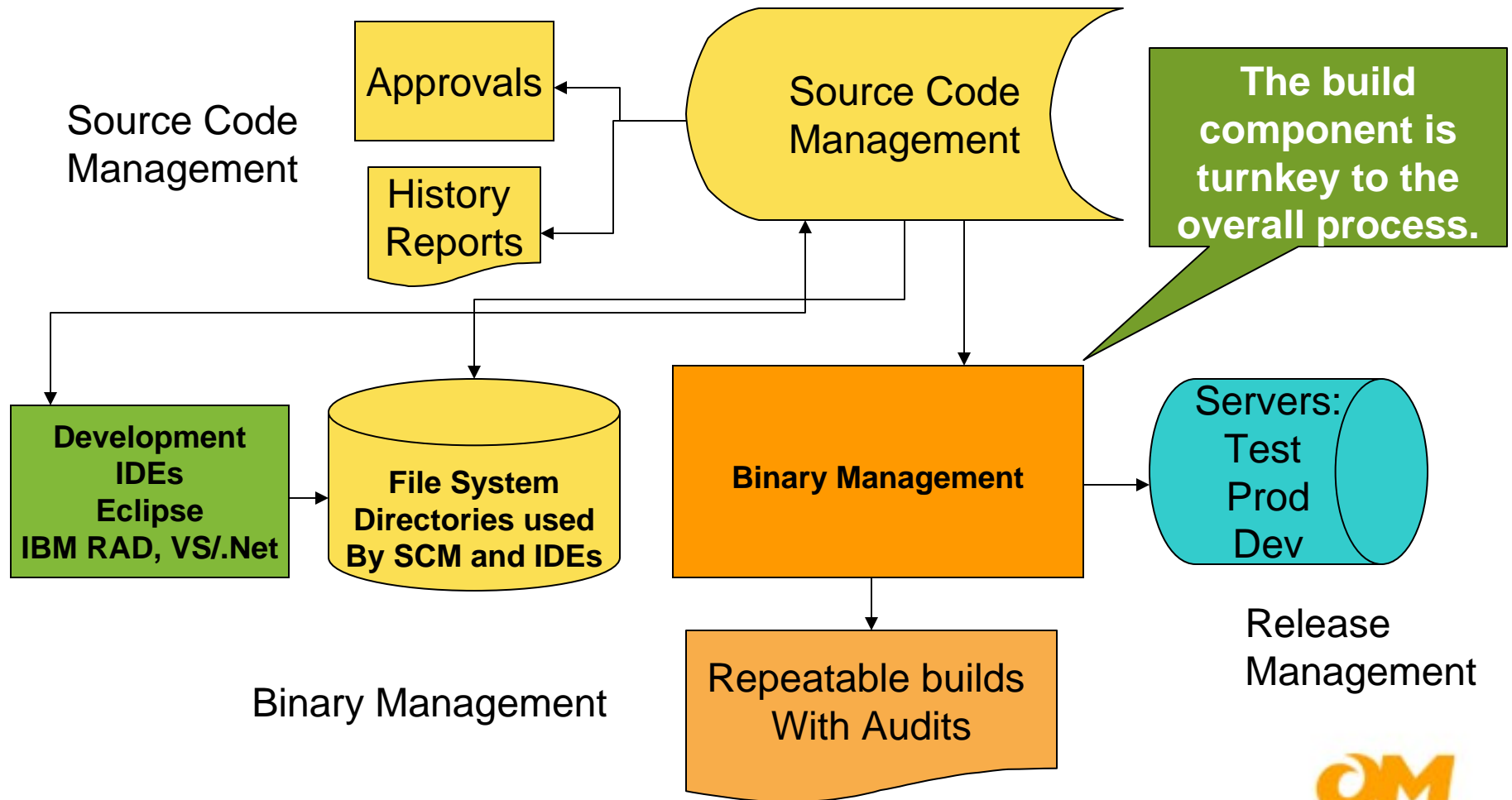
What's missing? – Binary Management

- Binary Management is the process of standardizing on the creation of binaries, in a 100% repeatable method, regardless of operating system or language.
- Mainframe binary management works the same way regardless if you are writing a COBOL or C program.



End to End ALM

Where Binary Management Fits





Achieving Binary Management

- Reduce your dependency on ad hoc scripts
- Never take binaries from a developer
- Identify the files used in the build, but not managed in SCM
- Create an audit trail by managing the dependencies and exposing the artifacts.
- Support build avoidance or incremental builds with your build process.



Achieving Binary Management

- Reduce your dependency on ad hoc build scripts and point and click processes.
 - Consolidate your build scripts.
 - Make the scripts re-usable.
 - Get out of the IDE.
 - Take a tip from the Big Boys
 - Mainframe processors are 100% re-usable.
 - Legacy Unix Makefiles are not recursive. Instead of 100s of little makefiles, they use one large standard makefile.



Polling Questions

- Who manages your production build scripts on the distributed platform?
 - The Developers
 - A Developer dedicated to builds
 - Production Control
 - Change Management



Achieving Binary Management

- Never take binaries from a developer or execute the developer's script.
 - Developers are already under the gun to produce. The “production build” is best managed outside of the development team.
 - If the Development team has a dedicated “build meister”, than this individual could be a “trusted partner” of the production control team.



Achieving Binary Management

- Never take binaries from a developer or execute the developer's script.
 - Developer's scripts are written to run on a particular development machine with a particular configuration. Production scripts should support the production environment.
 - Your builds should support a dynamic process that allows you to build in multiple locations with different configurations.



Polling Question

- How many different locations can your production builds be executed?
 - Just on our dedicated build servers
 - On both our development and pre-prod build server
 - Anywhere at anytime as long as the compilers are installed



Achieving Binary Management

- Identify the files used in the build, but not managed in SCM
 - If you have a code approval process, than you should manage a production build process that ensures that the scripts are pointing to the approved code. Developer's scripts support development, not production.
 - Developer's use compile and link flags for development usage i.e., debug on, optimization off. Production scripts should be just the opposite.



Achieving Binary Management

- Create an Audit Trail by discovering the dependencies – Foot printing!!!!
 - Your build should produce a report, or a foot print, that exposes every artifact used in the build.
 - A BOM is not enough. Just listing the files that were checked out prior to the build is not enough. The scripts could use files from other locations. And the files may not come from SCM so they will not show up in the BOM Report.



Achieving Binary Management

Build Audit Report for metalworks.jar

Project Variables:

Built on SONYLAN by steve at 04/18/2006 08:47:59

Environment Variables:

```
APPL=METALWORKS
CFG=RELEASE
COMPUTERNAME=SONYLAN
JAVA_HOME=D:\Program Files\Java\jdk1.5.0_06
OPENMAKE_SERVER=http://sonylan:58080/opermake
OS=Windows_NT
Os2LibPath=D:\WINNT\system32\os2\dll;
PERLLIB=D:\Program Files\opermake641 client\perl\lib
PROJECTVPATH=.;
Path=D:\PROGRA~1\Java\JRE15~1.0_0\bin;D:\Program Files\opermake641 client\bin;D:\Program Files\opermake641 client\perl\bin;D:\WINNT\
ProgramFiles=D:\Program Files
Pwd=D:\ClearCaseViewsDev1\dev1_dev\metalworks_sources\InitialComponent\Metalworks
REFDIR=D:\Program Files\opermake641 client\examples\ref
STAGE=QA
USERDOMAIN=SONYDOMAIN
USERNAME=Steve
VPATH=.;$(REFDIR)/metalworks/qa/src;$(REFDIR)/metalworks/qa;$(REFDIR)/metalworks/release/src;$(REFDIR)/metalworks/release;$(JAVA_HOME
```

Dependencies:

Version	Date	Time	Size	Target	Dependencies
	04/18/2006	08:48:12	9	metalworks.javac	
	04/18/2006	08:48:01	52	metalworks.classpath	
rt.jar is not in a VOB	11/10/2005	14:19:19	37757974	D:\Program Files\Java\jdk1.5.0_0	
AquaMetalTheme.java@@\main\dev1_dev\2	03/13/2006	17:33:40	2428	D:\ClearCaseViewsDev1\dev1_dev\m	
BigContrastMetalTheme.java@@\main\dev1_dev\2	03/13/2006	17:33:46	4220	D:\ClearCaseViewsDev1\dev1_dev\m	
ContrastMetalTheme.java@@\main\dev1_dev\2	03/13/2006	17:33:49	4642	D:\ClearCaseViewsDev1\dev1_dev\m	
DemoMetalTheme.java@@\main\dev1_dev\2	03/13/2006	17:33:51	3628	D:\ClearCaseViewsDev1\dev1_dev\m	
GreenMetalTheme.java@@\main\dev1_dev\2	03/13/2006	17:33:53	2399	D:\ClearCaseViewsDev1\dev1_dev\m	
KhakiMetalTheme.java@@\main\dev1_dev\2	03/13/2006	17:33:55	2837	D:\ClearCaseViewsDev1\dev1_dev\m	
MetalThemeMenu.java@@\main\dev1_dev\2	03/13/2006	17:33:57	2963	D:\ClearCaseViewsDev1\dev1_dev\m	
Metalworks.java@@\main\dev1_dev\2	03/13/2006	17:33:58	2975	D:\ClearCaseViewsDev1\dev1_dev\m	
MetalworksDocumentFrame.java@@\main\dev1_dev\2	03/13/2006	17:34:00	6129	D:\ClearCaseViewsDev1\dev1_dev\m	
MetalworksFrame.java@@\main\dev1_dev\2	03/13/2006	17:34:02	9616	D:\ClearCaseViewsDev1\dev1_dev\m	
MetalworksHelp.java@@\main\dev1_dev\2	03/13/2006	17:34:03	4909	D:\ClearCaseViewsDev1\dev1_dev\m	



Open Make
Software



Polling Question

- What do you use to show a footprint for matching source to binaries?
 - We rely on the SCM Check-out report - Bill of Material.
 - Custom Script tracing compiler calls.
 - We don't do a build footprint.



Achieving Binary Management

- Build Incrementally using Build Avoidance
 - Speed up your builds using Build Avoidance. A production fix should not require a complete re-build of the system.
 - Emergency builds should be possible. Build only what has changed.
 - Incremental builds supports lean methodology where eliminating redundancies and improving speed is the ultimate goal.



Conclusion

- Do not discount the processes that have supported the mainframe and legacy environments.
 - Turning over builds to the Production Control team or Systems Admins has served these environments well.
 - It can work for the distributed teams regardless of what you are developing in.



Conclusion

- Big Benefits that Big Boys have enjoyed.
 - Reduced development cost by eliminating hundreds of JCL components and Makefiles.
 - Improved development process by minimizing “broken builds”.
 - Reduced risk to production by allowing incremental emergency fixes.
 - Improved releases by providing production control the knowledge they need to predict the success of a release.



Polling Question

- What is your biggest build challenge?
 - Managing dependencies and reusable components.
 - Tracing build configurations and compile options.
 - Consistent Continuous Integration Builds.
 - Accurate and repeatable builds that do not break.
 - Consistent builds inside and outside IDEs



- Questions or Comments from the group?

Thank you for joining and happy building!