IBM Software Group

Building Rational *Collaborative Lifecycle Management (CLM)* **Enabling Continuous Integration for the Enterprise**

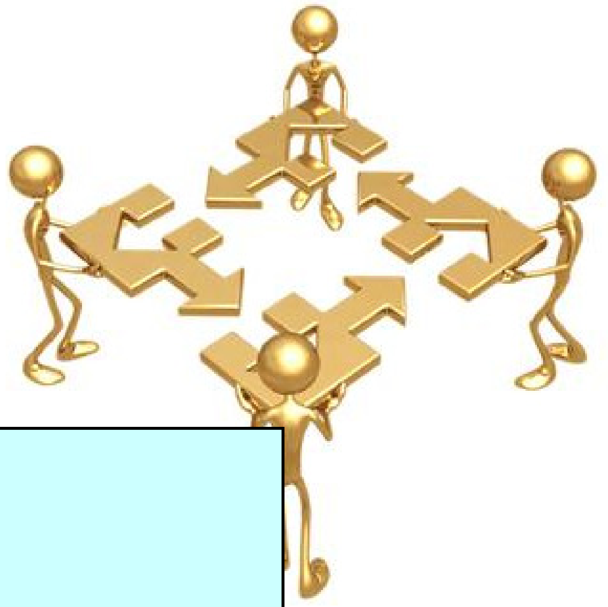
Martha DasSarma, Sonia Dimitrov, Christopher Maguire, Peter Steinfeld

May 4, 2011

Building *Rational Collaborative Application Management*

- On June 14th, 2011, we will ship Rational CLM
- CLM integrates the work of 170 people and 4 product teams
- To ensure that we can ship on June 3rd, we build the integrated product every day
- Normal continuous integration techniques do not scale to projects with many people and many teams
- ***Here's how we do it …***

Topics



- Continuous integration –
  - **Why** it’s good and **how** it’s usually done
- Implementing continuous integration for large teams is hard
- *How we scale continuous integration to the 4 CLM product teams*
- Details of the required build infrastructure
- Details of the individual builds

Why continuous integration is good

- Continuous integration reduces surprises and risk.
    - There are many small integrations throughout the project rather than a few big ones.
    - There's no "death march" at the end of the project when integration is forced.
- Continuous integration promotes communication and cooperation among developers.
- Continuous integration makes project status transparent.
- Schedules are more predictable.
- Continuous integration is good for morale. Everyone sees the project making steady progress.

How continuous integration is usually done

Successful projects practice continuous integration

- The project has a single main stream of development – the *integration stream*
  - Developers deliver to and collaborate in the integration stream
  - Developers can run a personal integration build prior to delivery to validate their changes
- The integration stream builds successfully every day – ***a successful build validates the stream***
- Automated tests run as part of the build
- Clear project status is visible via the build results

But continuous integration for large projects is hard

- 170 people work on CLM
- People are located in many places -- Beaverton, Brunswick, California, China, Florida, France, Haifa, India, Littleton, Madison, Ottawa, Perth, RTP, Saskatoon, Toronto, Virginia, …
- There are 4 product teams – Jazz Foundation, RTC, RRC, and RQM
- Teams work in 5 project areas on 3 separate Jazz servers -- jazzdev, jazzdev02, jazzdev03

•There are too many people to collaborate in one stream

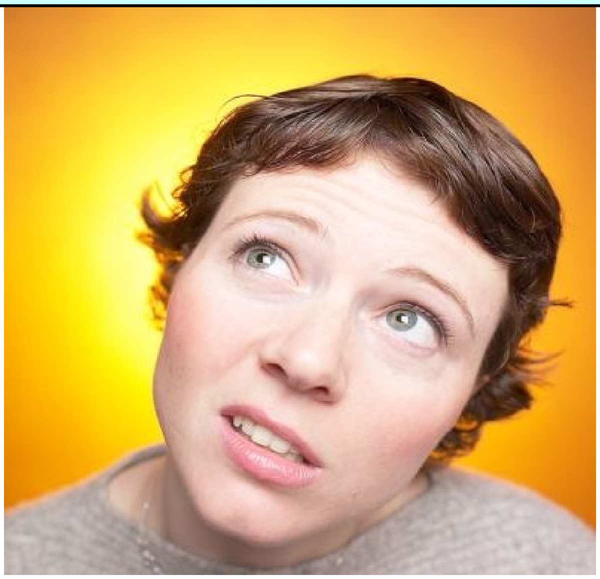
▪Each product team has its own community, processes, and culture

•Multiple Jazz servers make a single integration stream impossible

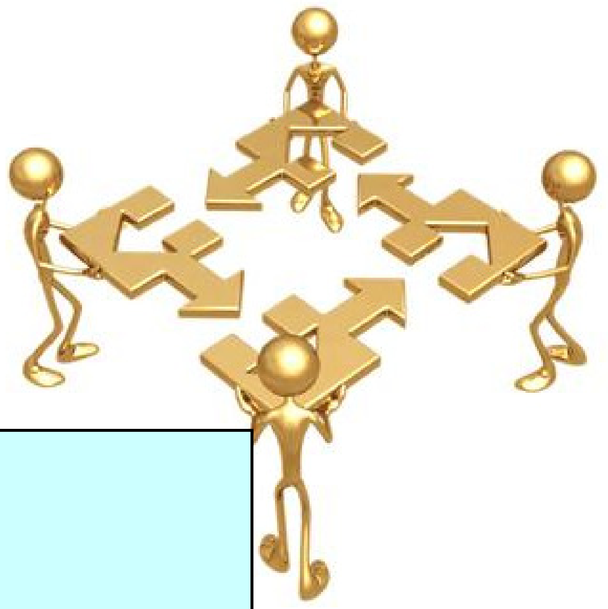
And yet …

- We build CLM twice a day, every day
- **24** of the last **25** builds were green
- A total of **389** CLM builds with **58** failed builds for an **85%** success rate

How are we able to do this?



Topics

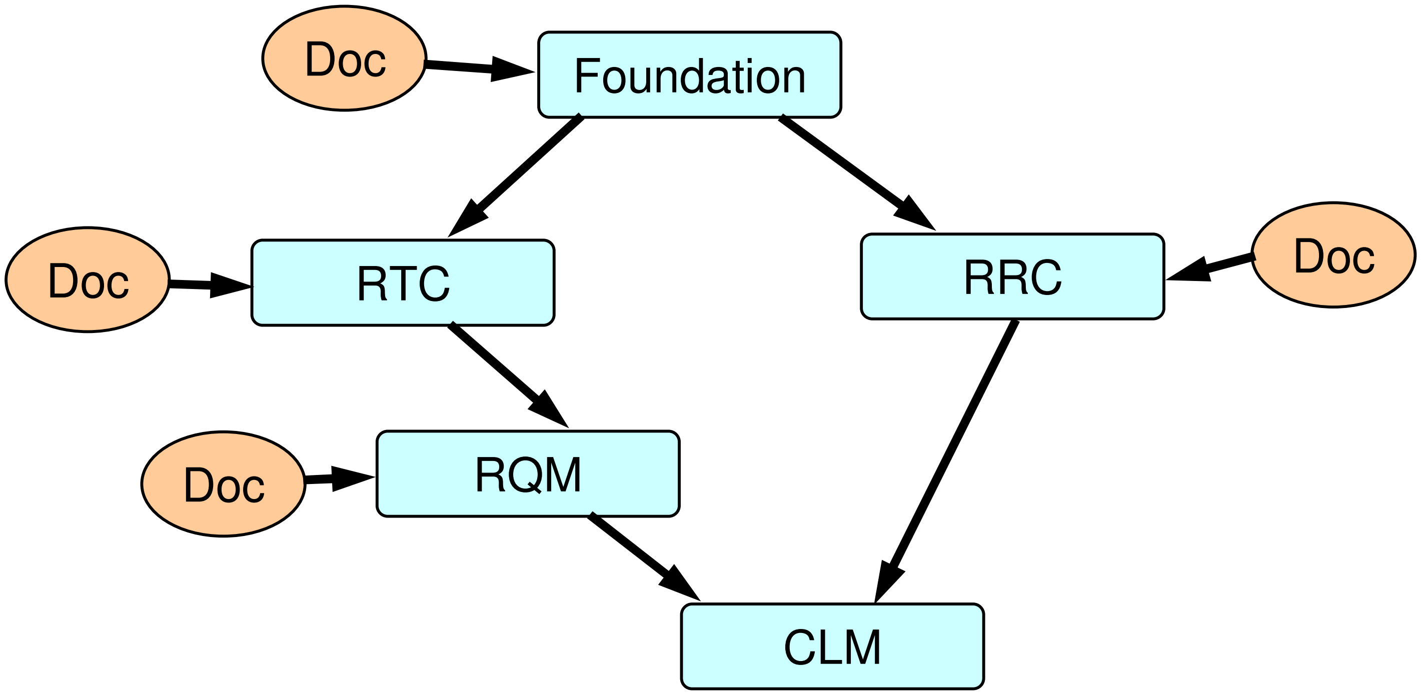


- Continuous integration –
    - Why it's good and how it's usually done
- Implementing continuous integration for large teams is hard
- ***How we scale continuous integration to the 4 CLM product teams***
- Details of the required build infrastructure
- Details of the individual builds

Vocabulary lesson

- CLM – Collaborative Lifecycle Management product
- Foundation – short for Jazz Foundation
- RTC – Rational Team Concert product
- RRC – Rational Requirements Composer product
- RQM – Rational Quality Manager product
- UA – User Assistance, also known as documentation
- FVT – Functional Verification testing – the functionality of a single product
- SVT – Systems verification testing – the interactions between products

Structure of the CLM build



▪Products share from their dependencies – for example, RQM depends on and shares code with RTC and Foundation

▪Dependencies determine build order

Continuous integration in CLM – *how we build it*

Successful projects practice continuous integration

- Each product team maintains a CLM integration stream
  - Builds from the products' CLM integration streams contribute to the CLM build
- The CLM builds run every day, validating the content of the product teams' integration streams
- Development teams can do a test build of the full CLM stack to validate their own stream and build
- No automated tests run as part of the CLM build, but manual testing is done on the CLM build and automated tests run in the product builds
- Green CLM builds are a good indicator that the project is going well

Builds enable continuous integration

- Each build is a single RTC build definition –
  All builds support RTC personal builds
- Each build is simple and complete – it transforms source code into testable artifacts that the customer sees (zips and installable offerings)
- All builds use similar underlying technology – Common Component Build (CCB) tool, shared build engines, scripts to maintain the build farm, …
- All build-to-build communication is via versioned repositories – for example, the Foundation build produces a repository that's input to the RTC build
- Build technology supports automated adoption of latest promoted version of upstream builds
- There are many big builds, requiring a large, powerful build infrastructure

Build rhythm

- All product builds follow the same rhythm
- Daily builds of Foundation with a **98%** success rate
- Daily builds of RTC that adopt the latest good Foundation build with an **84%** success rate
- Daily builds of RRC, RQM, and CLM based on the latest good RTC build and its matching Foundation build with success rates of **95%, 85%,** and **96%**
- Weekly builds with more test pressure
  --Some teams require explicit approvals from team leads
- Milestone builds every 3 weeks that we use to self-host
- All CLM builds are tracked by Track Build Items

"Successful daily builds are the heartbeat of a software project. If you do not have successful daily builds, then you have no heartbeat, and your project is dead!"

Jim McCarthy, Microsoft VC++ product manager

Product team rules of the road

- Each product team build creates its own testable product
- Product teams contribute new features at their own pace
- Product teams determine their own development, test, and delivery process

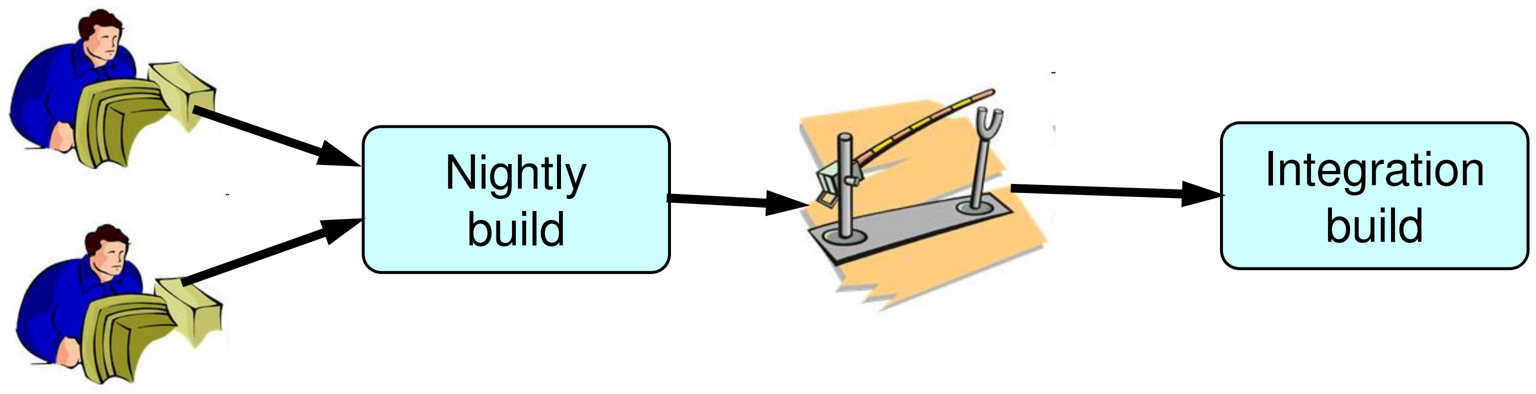
<u>But</u>

- Teams must continuously adopt new versions of upstream builds
- ***<u>Each team must maintain a build/stream for building CLM</u>***

Product team rules of the road – *maintain a stable CLM stream*

- Each product team build maintains a stable CLM integration stream and build, but ***how??***
- All product teams are too large for developers to deliver directly to the stable CLM integration stream, thus …
- Each product team maintains a stable CLM integration stream using one of two strategies –
  - Nightly builds (RRC and RQM)
  - Component team builds (Foundation and RTC)

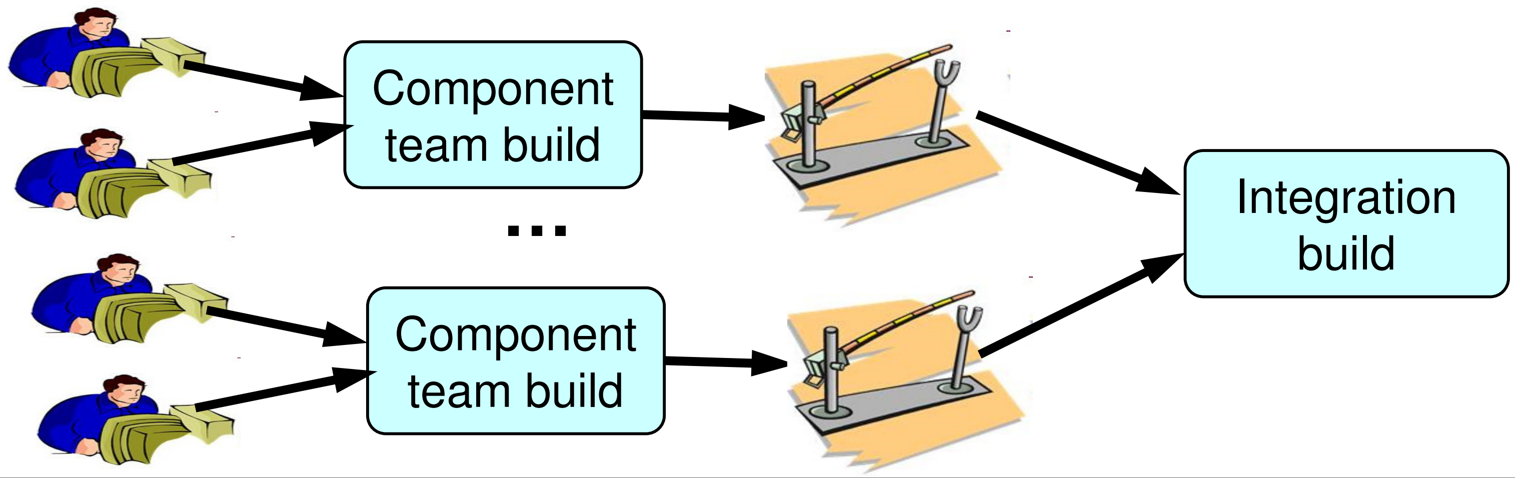
Nightly builds (RRC and RQM) – *maintaining a stable CLM stream*



▪Developers deliver to the nightly stream and build

▪The development manager decides when to deliver from the nightly stream to the integration stream and build

Component team builds (Foundation and RTC) – *maintaining a stable CLM stream*



- Product team is subdivided into component teams
- Each component team has its own stream and one or more builds from that stream
- Component team builds are small, fast, and run JUnits
- Component stream content includes code, tests, and build configuration files – all are controlled in SCM and delivered simultaneously by the component team
- Each component team has a rotating release engineer role, responsible for integrating with the integration stream and build
- Foundation has **11** component team builds, RTC has **24**

Validating CLM and product team builds

▪Build breakages are monitored by the build's release engineer

▪Everyone collaborates to fix broken builds – release engineers and developers

▪Green builds are automatically promoted for use by consumers – developers, testers, and downstream builds

▪The whole CLM development organization self-hosts on milestone builds

▪Types of validation tests –

- ▪Scans run in build (Eclipse version numbers, translatability – CHKPII, copyright validation) – failure does not prevent automatic promotion
- ▪JUnit tests run by developers and in the build – failures in the build prevent automatic promotion to downstream consumers
- ▪Functional Verification Testing (FVT) – done post-build
- ▪Systems Verification Testing (SVT) – done post-build
- ▪Security testing (AppScan) – done post-build

How Martha builds CLM

- Martha DasSarma is the CLM release engineer
- Starting point – green Foundation and RTC builds
  - Foundation builds daily
  - RTC builds daily and adopting the latest good Foundation
- Martha builds RRC using the same Foundation as RTC
- Martha builds RQM using the same RTC and Foundation
- Martha builds CLM
- End-to-end build time – 10 hours
- *The end-to-end build is not automated –*
  6 separate build requests are required

This works because each product team maintains a stable CLM stream

How Martha builds CLM – many people can do this!

- All products use the same version of RTC for development, thus …
- All builds can be controlled from a single RTC client
- Anyone on the release engineering and many people on the product teams product team can do a full stack build – either production or personal builds
  - Product teams can test changes that affect their consumers by doing a full CLM stack build
  - The release engineering team can go on vacation

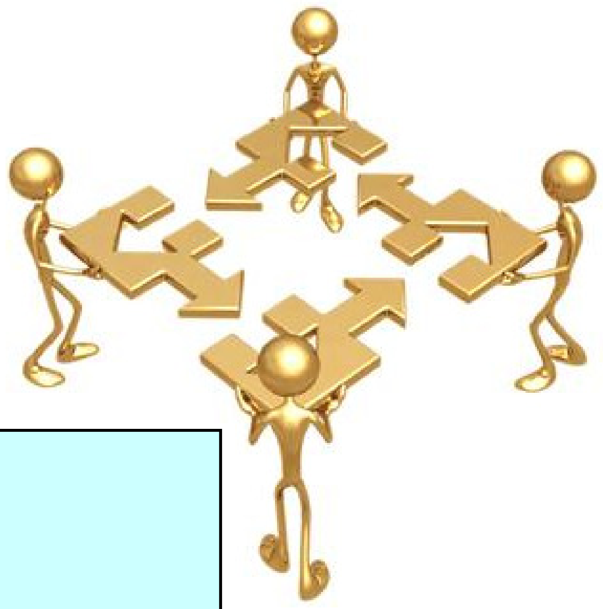
Product teams can validate their CLM streams by doing a CLM build

Test build of CLM – *RQM use case*

How the RQM product team can verify that they won't break the CLM build

- The RQM team does a personal build of the RQM build
  Build time – 65 minutes
- The RQM team adjusts the configuration of the CLM build to use the RQM personal build as input
- The RQM team does a personal build of CLM
  Build time – 75 minutes
- If necessary, the RQM team installs and tests the resulting CLM
- If all is good, the RQM team delivers from the personal stream to the RQM CLM stream
- The next RQM integration build will contain the same changes

Topics



- Continuous integration –
  - Why it’s good and how it’s usually done
- Implementing continuous integration for large teams is hard
- *How we scale continuous integration to the 4 CLM product teams*
- **Details of the required build infrastructure**
- Details of the individual builds

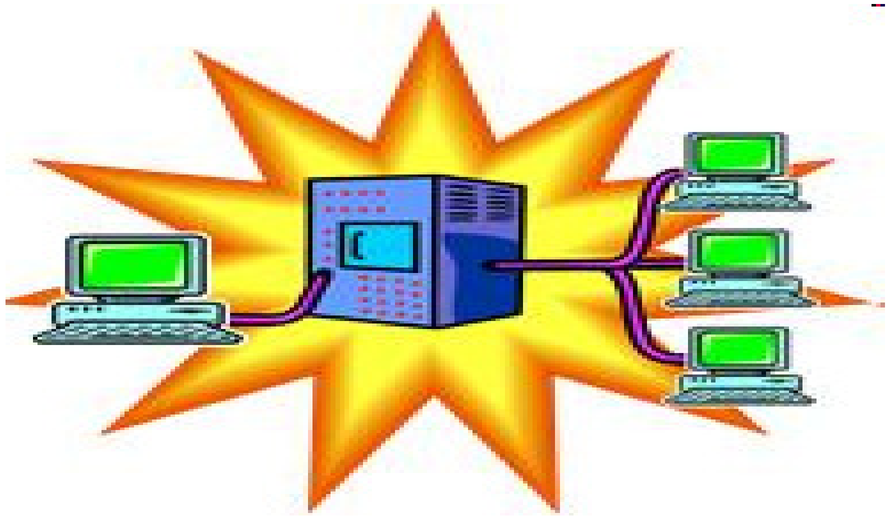
Infrastructure to support CLM builds

On March 2nd 2011, 222 CLM-related builds ran over 200,000 tests!

- ***Large scale development efforts need a powerful infrastructure***
- Jazz servers for development, build, test, and project management
- Release engineering team to keep the builds running smoothly
- Shared, standard build tooling
- Build farm that can handle peak build volume

Rational Team Concert server infrastructure -- providing an RTC development

- **3** Jazz servers – jazzdev, jazzdev02, and jazzdev03
- **6** Linux machines and **5** database servers (shared with the jazzop* servers)
- **4** engineers maintaining the infrastructure and managing self-hosts
- **1500** GB of data (controlled in RTC source code management)

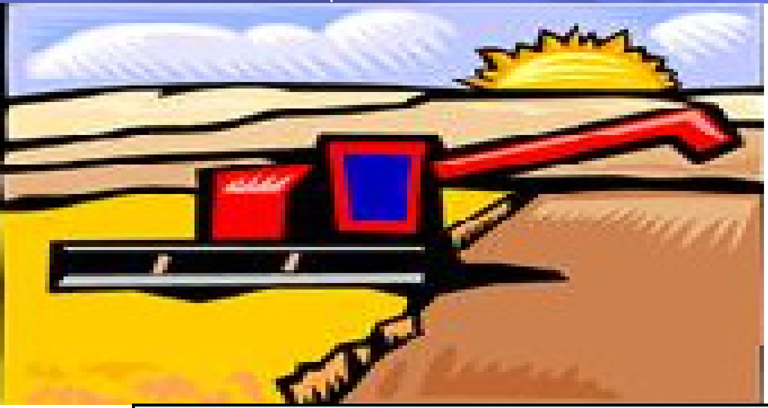


The CLM release engineering team – watching over the five CLM builds

- ***Release engineers are the guardians and keepers of the builds***
- **6** release engineers in **4** time zones and **5** locations – Bangalore, Littleton, Ottawa, Guadalajara, Beaverton
- Release engineers share build tools, build accounts, and scripts to maintain the build farm
- In general, one release engineer is responsible for each product
- Release engineers back each other up

Shared build tools

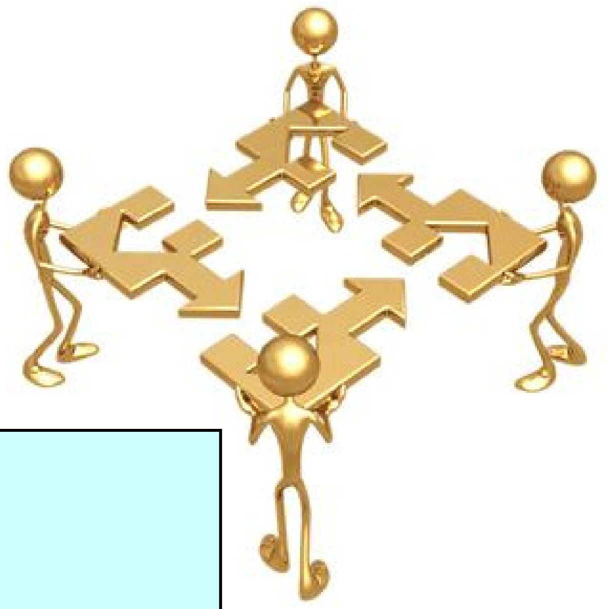
- **Release engineering scripts**
  - Manage the build farm
    - Set up new machines
    - Monitor disk usage
    - Update Jazz build engines
  - Deploy to test systems
- **Common Component Build (CCB) tool**
  - Support for code sharing through componentizing builds
  - Build-to-build communication via Installation Manager repositories
  - Shared configuration files for component team and production builds
  - Dependency handling (suppliers) supporting --
    - Automated adoption of the latest good upstream builds
    - Support for test builds of the full stack



Build farm – ready when a build is needed

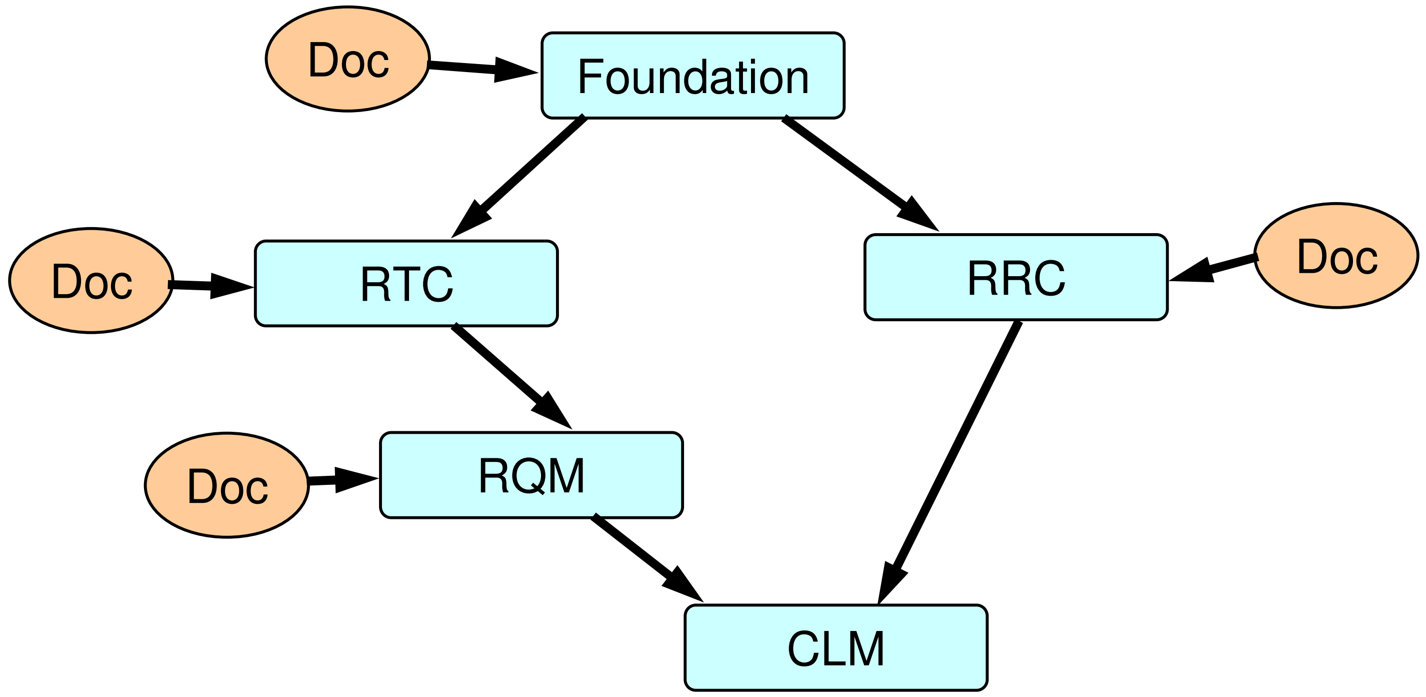
- ***The build farm must support peak build volume***
- **74** Jazz build engines on **55** physical machines
  - Foundation/RTC/CLM -- **59** Jazz build engines on **43** physical machines
  - RQM -- **8** Jazz build engines on **5** physical machines
  - RRC -- **4** Jazz build engines on **4** physical machines
  - UA -- **3** Jazz build engines on **3** physical machines
- **7000** GB served up by **3** disk servers
  - **2000** GB RAID 0 for personal builds
  - **5000** GB RAID 1 for production builds
- **2** engineers maintaining the build farm's machinery

Topics

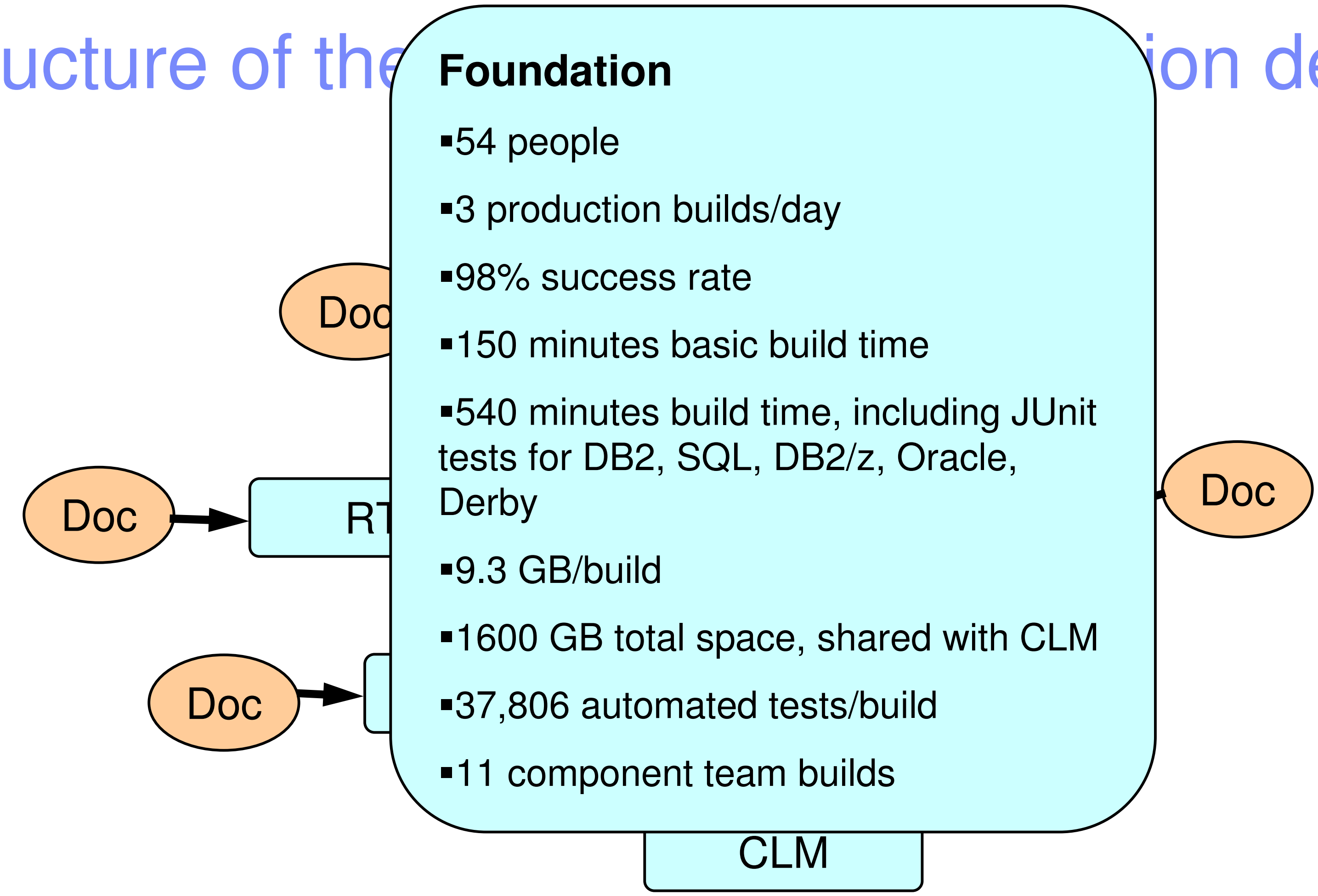


- Continuous integration –
  - Why it’s good and how it’s usually done
- Implementing continuous integration for large teams is hard
- *How we scale continuous integration to the 4 CLM product teams*
- Details of the required build infrastructure
- **Details of the individual builds**

Structure of the CLM build -- *product specific details*



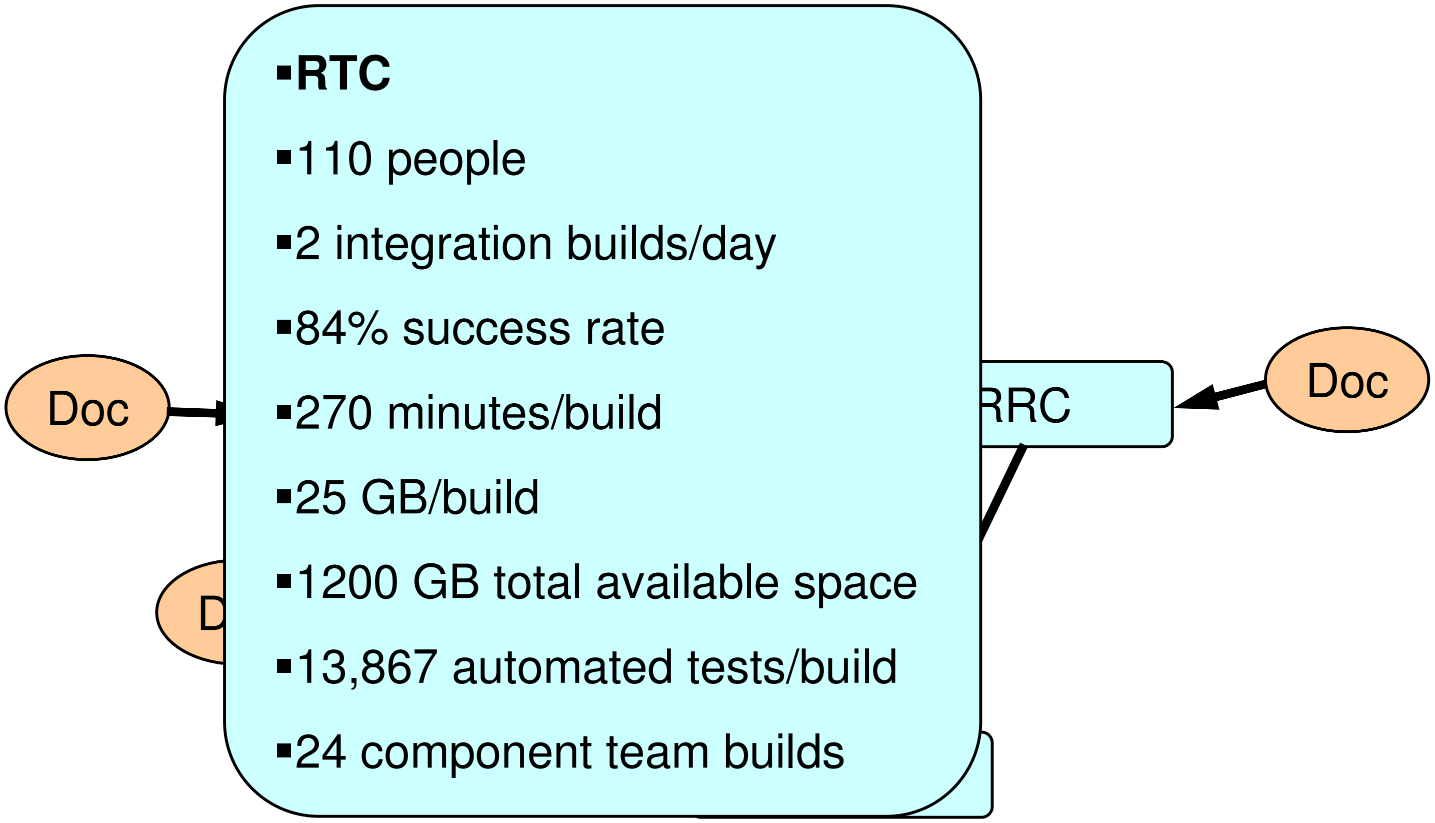
Structure of the ion details



Foundation

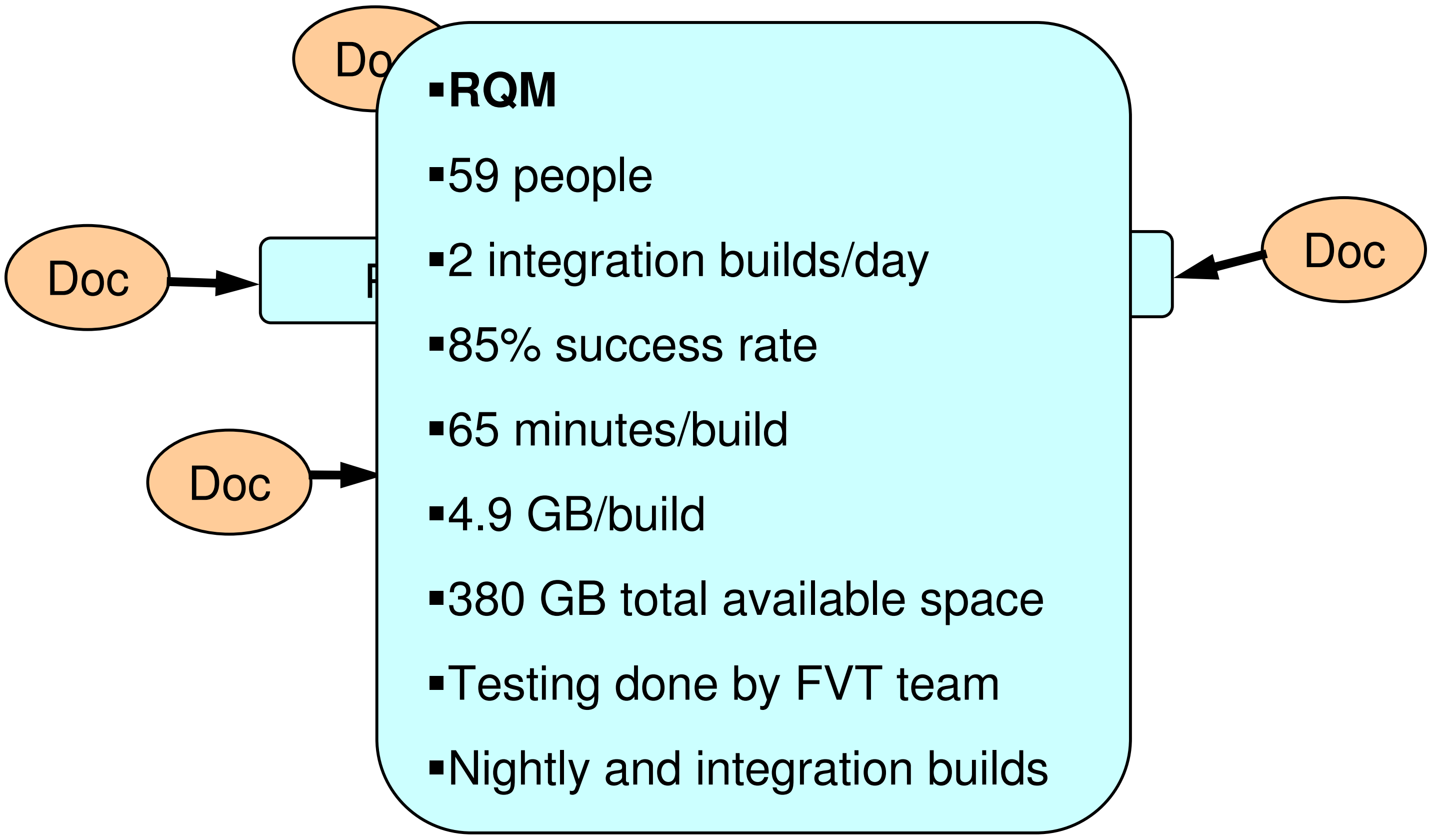
- ▪54 people
- ▪3 production builds/day
- ▪98% success rate
- ▪150 minutes basic build time
- ▪540 minutes build time, including JUnit tests for DB2, SQL, DB2/z, Oracle, Derby
- ▪9.3 GB/build
- ▪1600 GB total space, shared with CLM
- ▪37,806 automated tests/build
- ▪11 component team builds

Structure of the CLM build – RTC details



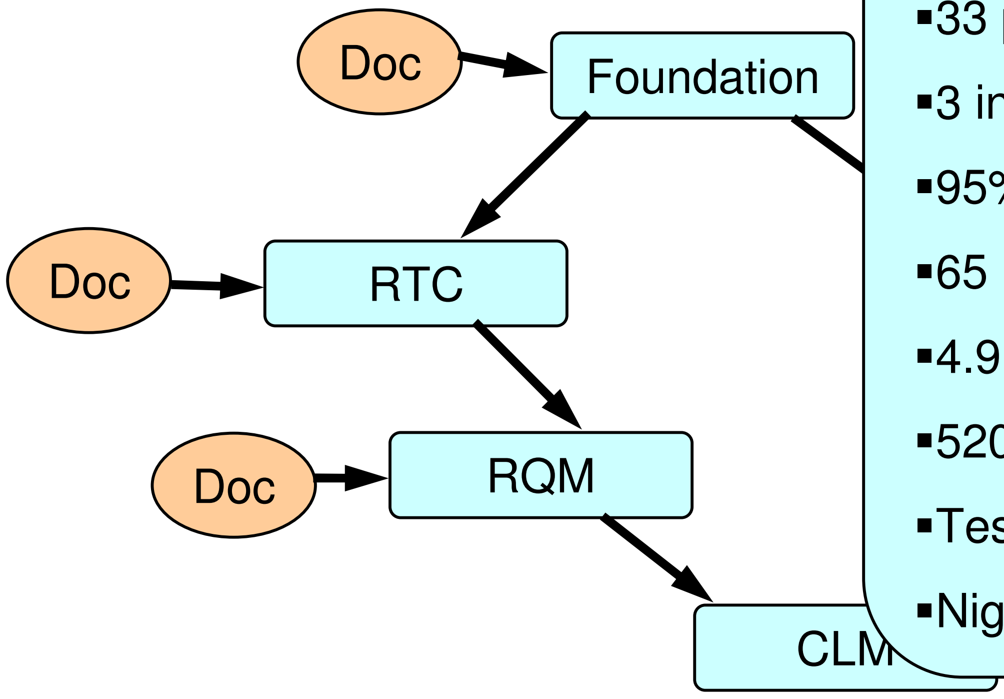
- **RTC**
- 110 people
- 2 integration builds/day
- 84% success rate
- 270 minutes/build
- 25 GB/build
- 1200 GB total available space
- 13,867 automated tests/build
- 24 component team builds

Structure of the CLM build – RQM details



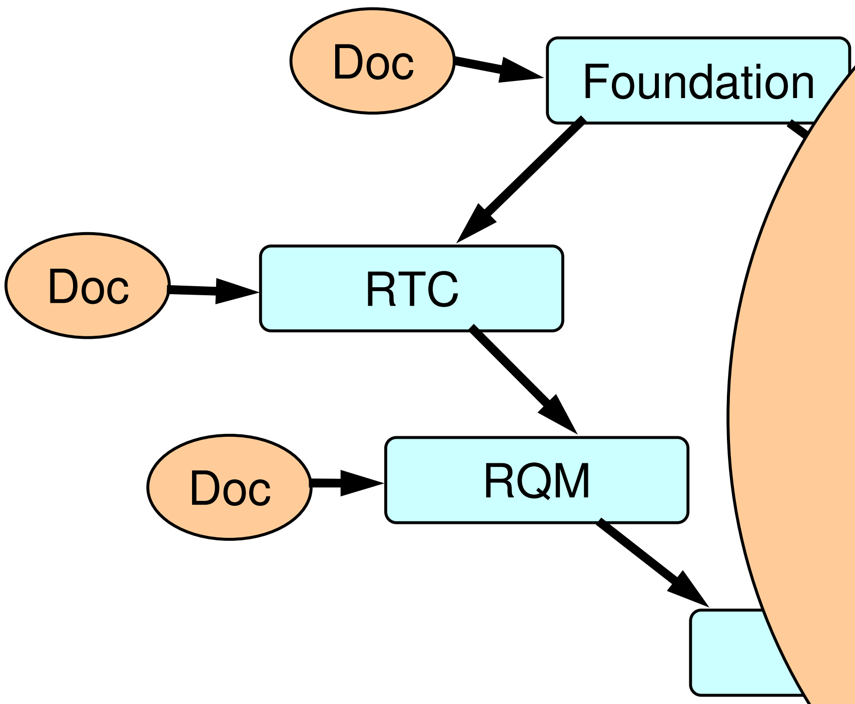
- **RQM**
- 59 people
- 2 integration builds/day
- 85% success rate
- 65 minutes/build
- 4.9 GB/build
- 380 GB total available space
- Testing done by FVT team
- Nightly and integration builds

Structure of the CLM build – RRC details



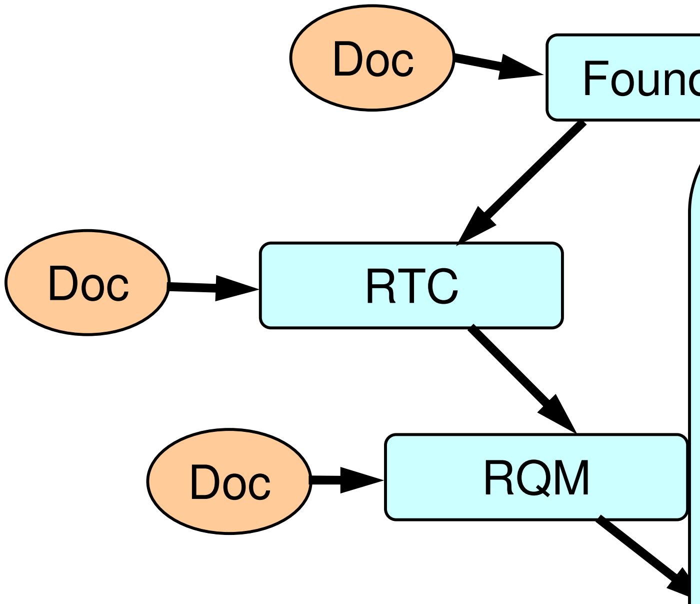
- **RRC**
- 33 people
- 3 integration builds/day
- 95% success rate
- 65 minutes/build
- 4.9 GB/build
- 520 GB total available space
- Testing done by FVT team
- Nightly and integration builds

Structure of the CLM build – UA details



- **Documentation** (for all products)
- 21 people
- 9 integration builds/day for
  7 different builds
- 98% success rate
- 15 to 200 minutes/build
- 160 MB/build
- 1400 GB total available space
- 0 automated tests/build
- Only integration builds

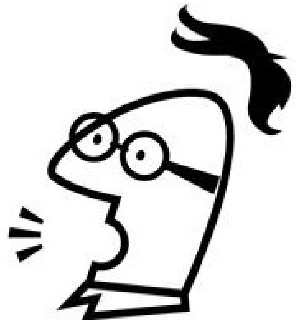
Structure of the CLM build – CLM details



- **CLM**
- 12 people
- 2 integration builds/day
- 96% success rate
- 75 minutes/build
- 16 GB/build
- 1600 GB total available space, shared with Foundation
- 0 automated tests/build
- Only integration builds

Summary statistics

- **2** production CLM builds/day
- **170** people in **17+** locations
- over **200** builds/day, personal and integration
  - **222** builds run on March 2nd, 2011
- over **200,000** automated tests per day
  - **226,522** in non-component team builds on April 26th, 2011
- **48** build definitions
- **74** build engines
- **8500** GB disk space
- **3** Jazz servers



Questions?