IBM Software Group

Global Development & Delivery

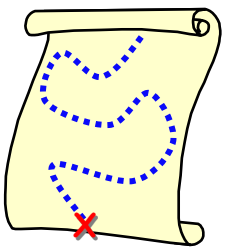
The politics, patterns and process of implementing distributed software configuration management

Kevin A. Lee – kevin.lee@uk.ibm.com
Solution Architect, IBM Software Services

IBM Software Group

Agenda

- **What is Global Development and Delivery?**
- **Patterns**
- **References**



© 2007 IBM Corporation

IBM Software Group

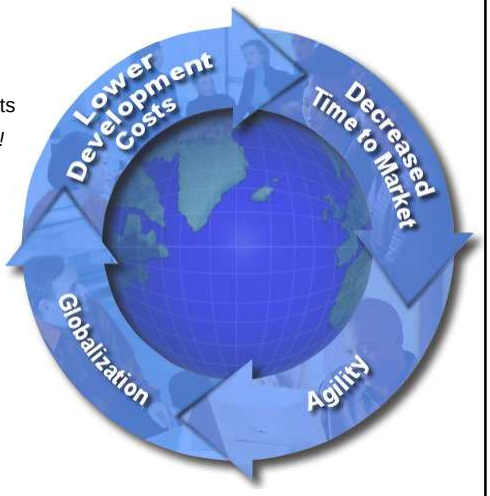
What is Global Development and Delivery (GDD)?

- Any scenario in software development that is carried out by teams across geographies.
  - Includes entire software lifecycle: design, implementation, testing, deployment, maintenance and support.
  - Made possible by the high bandwidth networks of today and the availability of skilled engineers worldwide.
- The *Delivery* of applications to Test and Production environments is an inherent part of any GDD solution lifecycle.
- **Effective** *Configuration, Change and Release Management* enables distributed development team **collaboration** and **governance** and is therefore critical to GDD.

IBM Software Group

Why GDD?

- **Lower Development Costs**
  - Lower labor rates
  - Reduce long term maintenance costs
  - *The original but not the only reason!*
- **Decreased Time to Market**
  - Follow the sun development
  - Access to global *skilled* resources
- **Globalization**
  - Product localization
  - Mergers and acquisitions
- **Agility**
  - Access to scarce resources
  - Virtual teaming



4

GDD Enablers

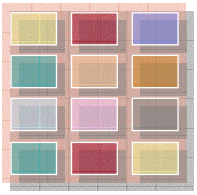
- **Collaboration**
  - the process by which teams accomplish work.
- **Governance**
  - the framework which allows them to collaborate
    - *Static* Governance – institution of chains of responsibility, authority and communication to empower people.
    - *Dynamic* Governance - institution of measurement, policy, and control mechanisms to enable people to carry out their roles and responsibilities.
  - Governance can be an enabler for collaboration, however too much or unnecessary governance can stifle collaboration.
- **Automation**
  - the use of integrated tools and technologies to enable effective collaboration and make governance palatable.

GDD Inhibitors

- **People**
  - *Language difficulties* – surely everyone speaks perfect English!
  - *Cultural difficulties* – you say tomato …
- **Process**
  - *Predictability* - distributed teams *need* to implement the same process.
  - *Sustainability* - local teams *need* to develop for future distribution.
  - *Compliance* – mandates can significantly increase the amount of governance.
  - *Measurement and Metrics* – agreeing, automating and interpreting them can be major cause of confusion.
  - *Knowledge Management* – transfer of knowledge and training of distributed teams can be a major cost.
- **Technology**
  - *Suitability* - for distributed development environment.
  - *Capability* - to increase collaboration.
  - *Integration* - into the larger scheme for traceability and automation (ALM)

GDD Implementation Patterns

- **Resourcing**
- **Infrastructure**
  - Logical and Physical
- **Governance**
- **Collaboration**



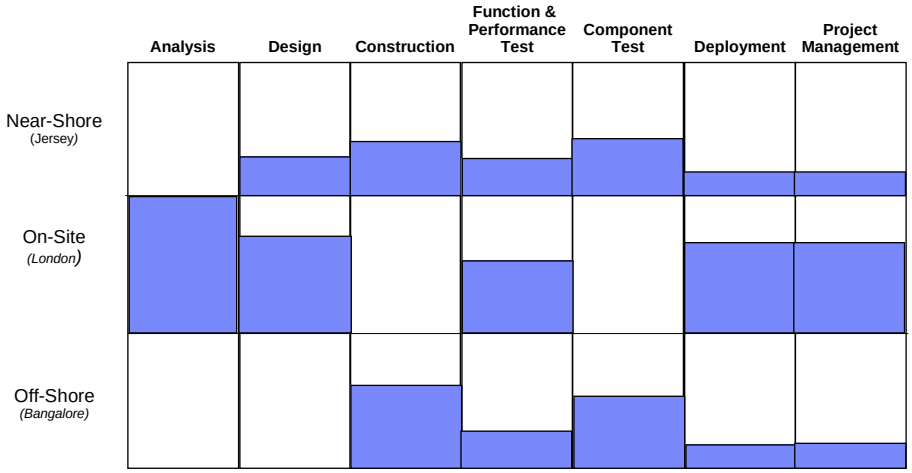
- **pattern names are underlined**
  - but some names are not very well chosen – any help is appreciated
- **and we will pay specific interest to those related to CM ☺**

Resourcing Patterns

- **Organisational Based**
  - **Outsourcing** – sharing organisational control with a different company
  - **Insourcing** – re-ownership of previously outsourced services
- **Project Based**
  - **On-site** – internal or outsourced staff
  - **Off-shore** – direct ownership of resources in a foreign country, in the form of a subsidiary or joint partnership
  - **Near-shore** – satellite office usually in a neighbouring country or in the same region
  - **Combined?** – (potentially) all of the above, maybe this should be called eXtreme Sourcing (XS) – increasingly the “normal” way of working…

IBM Software Group

Resourcing Patterns: *Combined Example*



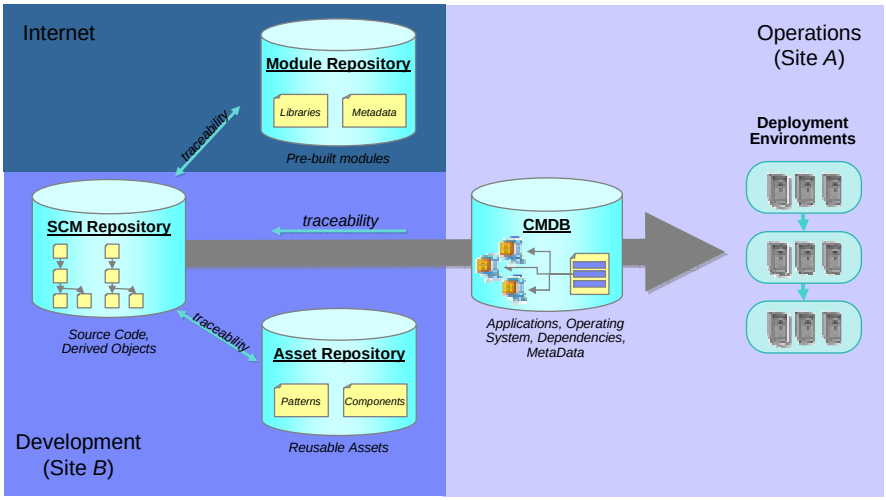
IBM Software Group

Logical CM Infrastructure Patterns

- **Most customers have a number of different repositories - more than they realise - each of these captures CIs at different levels of granularity:**
  - **SCM Repository** – typically source code, usually developed in parallel.
  - **Reuseable Asset Repository** – development time "reuseable" assets, e.g. patterns, models, solutions to common problems.
  - **Module Repository** – published, pre-built modules, e.g. Java libraries (JSR-277/Maven), Perl Modules (CPAN).
  - **Service Registy** – published searchable web-services and specifications.
  - **CMDB** – production assets (applications, operating systems, hardware etc) and their relationships
- **It is a big challenge to trace and management change across and between these different repositories.**
- **These repositories could also be hosted in different environments, different regions and have different security implications, e.g. Module Repository accessible by developers, CMDB only accessible by Operations, Service Registry accessible over the Internet!**
- **You must Plan, Document and Trace for every type of CI!!!**

IBM Software Group

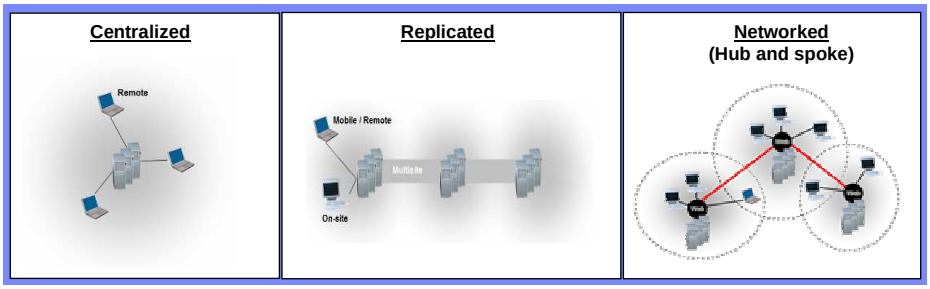
Logical CM Infrastructure Patterns: *Example*



IBM Software Group

Physical CM Infrastructure Patterns

- **Physical Infrastructure Requirements depend on a number of factors:**
  - Mix of trusted and untrusted sources (outsourcers); impacts security requirements
  - Amount of distribution; impacts collaboration required.
  - Desired TCO vs Perceived Risk vs Requested Site Independence
  - General trend for our customer is towards Centralized or Networked topologies.



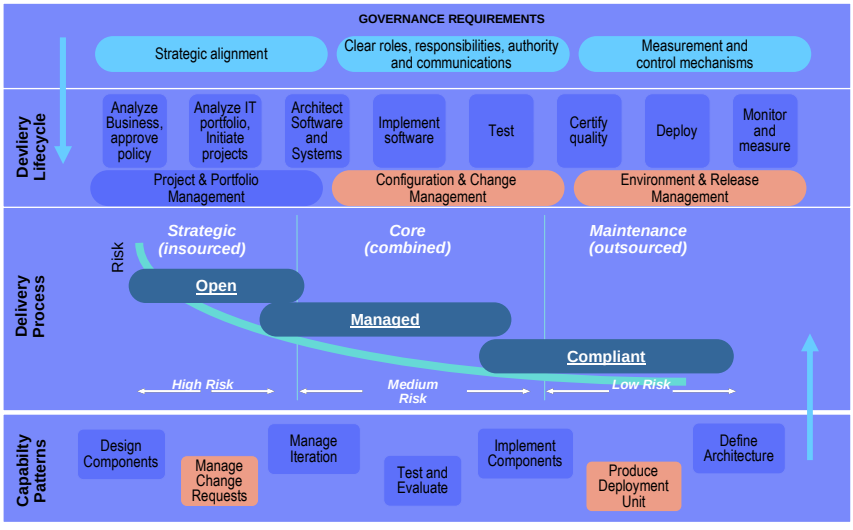
IBM Software Group

Governance Patterns

- **Enterprise organisations typically have different governance requirements, maybe event project by project.**
- **The individual process that a "project" will use depends to a large degree on these requirements, we typically see three patterns:**
  - **Open Development Process** – Agile like implementation for small sized and high risk projects.
  - **Managed Development Process** – (project managed) implementation for majority of core projects.
  - **Compliant Development Process or method?** – "high-ceremony" implementation for projects with compliance mandates.
- **Every development process must have CM!**
  - **Open** usually implements some form of Agile SCM (see references).
  - **Compliant** implements variation of Three Points of Control pattern.
- **Every development process must fit in with the organisations Change Request process!**

IBM Software Group

Governance Patterns: *Process Selection Example*



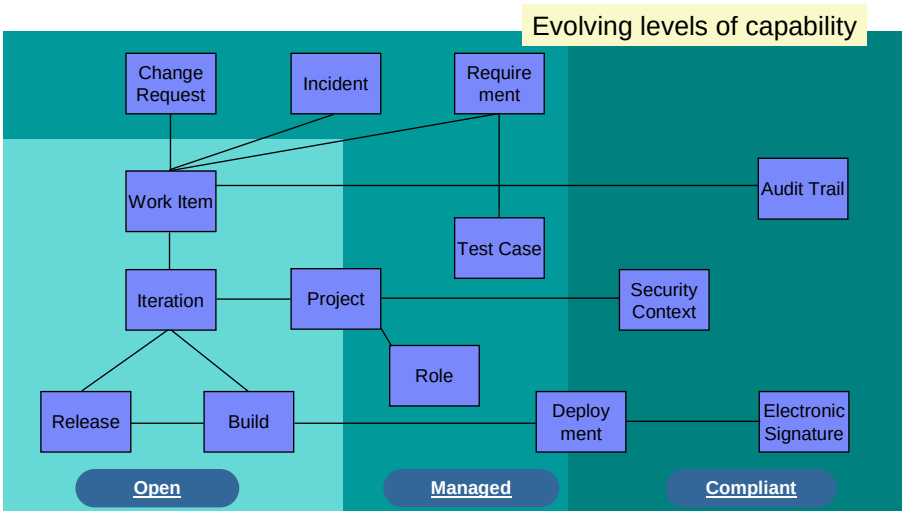
IBM Software Group

Governance Patterns: *Three Points of Control*

- **There are three key events where Control Points exist in a Change and Release Management workflow to allow or prohibit continuation.**
  - **Control point #1 - Deliver Change** is managed by the "*Lead Developer"* and provides a gate where new changes are approved for delivery to a shared team integration stream.
  - **Control point #2 – Register Derived Objects** is managed by the "*Build Administrator"* and provides a gate where build objects are added to version control and registered as "QA Ready". In order for the build to pass this gate the source code must be under version control and the build must have occurred in a controlled environment.
  - **Control point #3 – Deploy Objects** is managed by the "*Release Administrator"* and provides a gate that prohibits deployment of a baseline to QA and/or Production if it is not registered as "QA Ready" or "Prod Ready", respectively.

IBM Software Group

Governance Patterns – *Change Request Process*



IBM Software Group

Governance Patterns: Metrics and Measures

- ***Measures***
  - concrete or objective attributes (SLOC/Function Points)
- ***Metrics***
  - more abstract, higher-level, or somewhat subjective attributes (Robustness/Quality Feel)
- **Measures help separate management issues from development environment "ideology".**
- **Any development effort must have a Measurement Plan.**
- **Measures are best captured automatically at appropriate points, e.g. daily/nightly, at each build, at each milestone.**
- **Implementing GDD without a Measurement Plan is development suicide ☠**

IBM Software Group

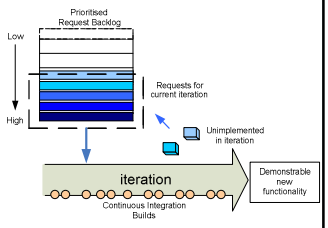
Governance Patterns: *Sample Measurement Plan*

- **Quality/Architectural Stability**
  - **Background**: Defect distribution is already captured to some extent.
  - **Goal**: Understand quality of delivered system
  - **Measure**: Defects in production two weeks after deployment of system per subsystem (coupling to architecture)
  - **Goal**: Understand quality of system before system test
  - **Measure**: System test resources used per subsystem and per project.
- **Estimation Variance**
  - **Background**: Want to get better control of methods for estimation. Estimates and actuals are captured, but in some areas schedule over-runs are hidden in maintenance. Also, there is a need to emphasize that planned values always have variance associated with them – variances that represent risk in a project, and that should become lower as the project progresses. Estimation should also be put in relation to a project's 'genre' – there is however still some work to be done to define the parameters that define genres.
  - **Goal**: Understand true estimates vs. actuals – how does it look for the various project 'genres'.
  - **Measure**: Planned vs. actual deliveries, and how they evolve as project executes.
  - **Goal**: Emphasize the importance of variances in estimates
  - **Measure**: Estimated variance to project cost/time and how it evolves as project progresses.
- **Change Management**
  - **Background**: Need to better understand the cycle times for change requests and increase the understanding of the impact to project schedules of change requests.
  - **Goal**: Understand change frequency and change cycle times for respective project 'types'
  - **Measure**: Cycle time for a change request.
- **…**

IBM Software Group

Collaboration Patterns: *Continuous Integration*

- **An Agile Development practice that focuses on frequently integrating small code changes:**
  - not restricted to Agile Development projects or teams.
  - in sprit simply:
    - an automated process for *building* AND *testing* all assets,
    - that can be run many times a day,
    - is self sufficient,
    - and creates a (potentially) executable application.
  - The refinement of a common software development best practice: the daily build and smoke test.
- **Excellent communication mechanism, establishes progress, embeds quality (suitability for testing) and enables controlled handovers.**
- **An ideal opportunity for automating the capture of measures:**
  - total/cumulative SLOC/function points etc.
  - although per build measures do not always tell the bigger picture and should be reinforced by other means.



IBM Software Group

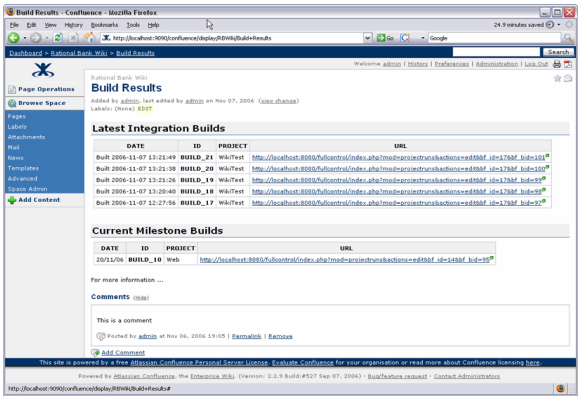
Collaboration Patterns: *Social Software*

- **Web 2.0/Social Software = the second generation of Web based services with a focus on community collaboration:**
  - **Weblogs** – user generated "journal" style internet pages.
  - **Wikis** – community generated inter/intranet web pages.
  - **RSS feeds** – syndication of frequently changeable web content that users can subscribe to.
  - **Instant messaging** – real-time "chat" based communication.
  - **Social bookmarking** – user generated bookmarks, shareable by others.
- **All of these services can be of significant benefit for software development team collaboration in distributed (GDD) environments.**
- **On every project I have worked on we have created a Wiki and used Instant messaging feverishly (if allowed)!**
- **If you are lucky your Wiki can also be integrated with your CM or Change Control System (hyperlink to version, changes, builds etc).**

IBM Software Group

Collaboration Patterns: *Social Software Example*

- **Automating the updation of a *Wiki page with the build results:**



*Confluence Enterprise Wiki Software

IBM Software Group

Collaboration Patterns: Jazz

- **Jazz is an open, scalable, extensible team collaboration technology for seamlessly integrating work across the development (both systems development and software development) lifecycle**
- **Jazz:**
  - enables development teams to collaborate in real time in the context of the work that they are doing
  - enables projects to be run more effectively by providing accurate real-time project health information drawn directly from actual project work
  - manages artifacts across the development lifecycle
  - Shifts thinking from “Individual first” to “Team first” to “Collaboration First”
  - will be part **open-source** and part **open-commercial**
  - won’t be available for a while (tech preview 2H07, release 08)

References

- **Jazz community site**
  https://jazz.net/pub/index.jsp
- **SCM Patterns for Agility**
  http://www.scmpatterns.com/
- **Continuous Integration (according to Martin Fowler)**
  http://www.martinfowler.com/articles/continuousIntegration.html
- **CM Crossroads**
  http://www.cmcrossroads.com
- **The Buildmeister (includes update Wiki script)**
  http://www.buildmeister.com
- **List of collaboration software**
  http://en.wikipedia.org/wiki/List_of_collaborative_software