# Enterprise Architecture

Jim Fawcett

CSE681 – Software Modeling and Analysis

Fall 2015

# References

- Beautiful Architecture, O'Reilly, 2009
  - Will become required reading for this course
  - Interesting examples:
    - Massive Multi-player on-line games
    - Facebook data management
    - Surfing business data like surfing web pages using REST
- Beyond Software Architectue, Luke Hohmann, Addison-Wesley, 2003
  - Places software architecture in the context of a business process (concept, plan, marketing, development, deployment, extension, security)

# Need for Architecture

- Driven by:
  - System complexity
  - Performance requirements
  - Need for flexibility
    - To accommodate changing business objectives
    - To replace aging technologies
    - To cooperate with other applications
  - Security
    - Visibility and mutability based on authorization
    - May need audits to prove compliance with regulations

# Needs driven by Size

- Longevity
  - Enterprise system are expensive to build and deploy.
  - We want them to last a long time to get significant return on investment.
- Stability:
  - We want the core system to remain stable as development proceeds and later as maintenance adds new features

# First Questions (BA)

- The first questions an architect asks are not about functionality:
  - Who are the stakeholders?
  - On what platform will the system be built?
  - How many concurrent users?
    - Load model
    - latency
  - How secure does the system need to be?
    - Intranet or internet?
    - How sensitive is the information?
  - How scalable must the system be?
    - Orders of magnitude?

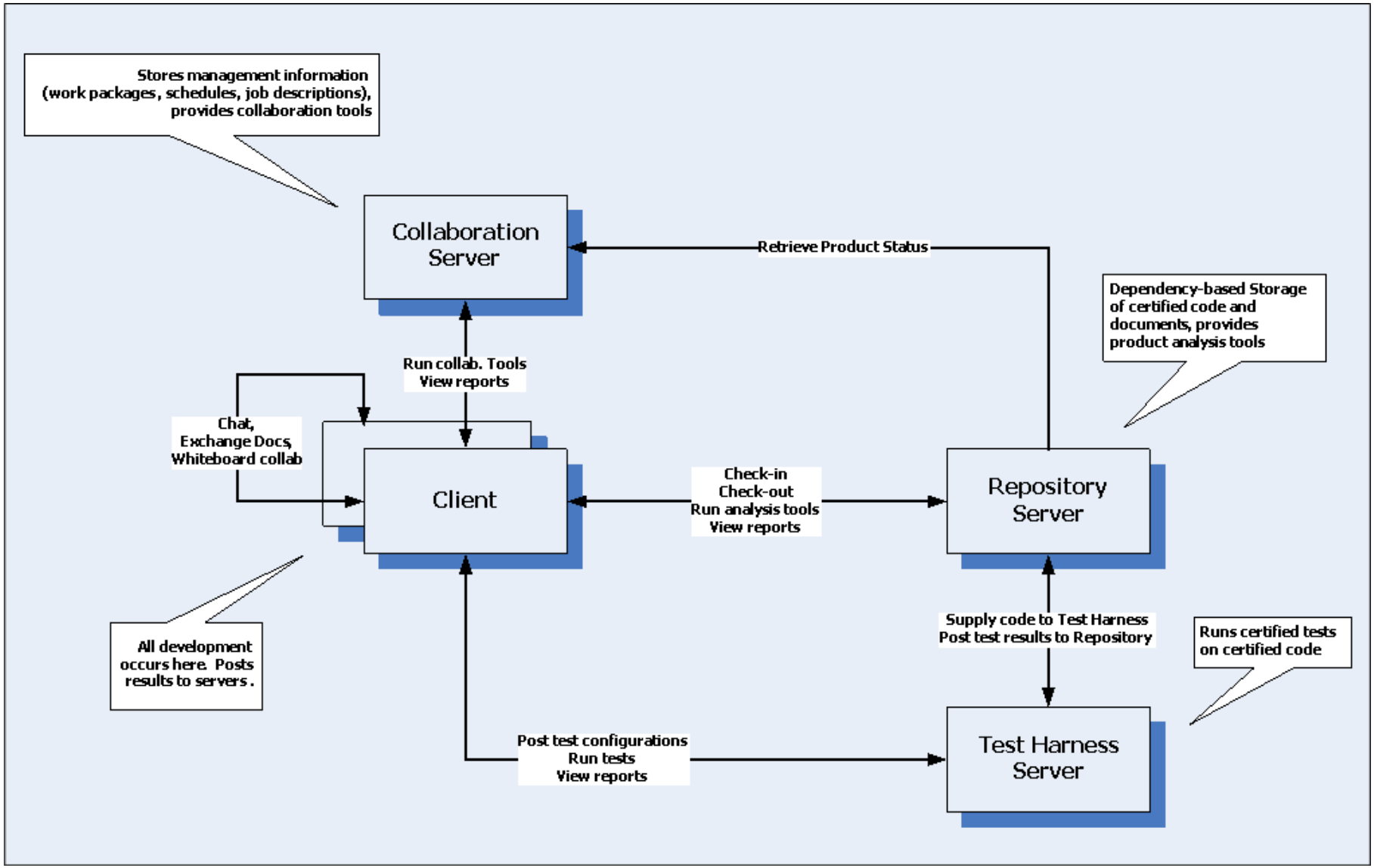
# Goals of an Architecture (BA)

- Build systems that:
  - Satisfy project goals
  - Are:
    - friendly and responsive to the user
    - free of critical errors
    - maintainable
    - easy to install
    - reliable
  - Communicate in standard ways

# Architecture Quality Factors

- Usability
  - Free of critical errors
  - Metaphors
  - Performance
- Security
  - Safe
  - Traceable
- Scalability
  - Add more functions
  - Add more users
  - Add more data
- Maintainability
  - Changeable
  - Stable

# Software Collaboration Federation



# Usability

Software Collaboration Federation (SCF)

- Users don’t wait for anything
  - All tasks are asynchronous
- Simple models
  - Everything is immutable
  - Only control is check-in
  - Concurrency models are simple and hidden from user
- All work is task driven:
  - Check-in products
  - Test code products
  - Analyze results

# SCF Security

- Use platform-based authentication
  - Asp.Net like authentication and roles
- Use secure communication
  - WSHttp encrypts transmissions
- Products contain encrypted hash to prevent tampering
- Message logs support traceability if needed

# SCF Scalability

- Test Harness (prime load – test execution)
  - Concurrent test suites use available cores
  - File caching avoids unnecessary network traffic
  - Tests are independent so very little is required to support load-balancing of multiple Test Harness servers
- Repository (prime load – builds)
  - Check-ins are independent so each can run on own thread, using available cores for building
  - File caching avoids unnecessary network traffic
  - Fine-grained availability is not important, so can host on multiple servers, synchronized at night.

# SCF Maintainability

- Test Harness:
  - Functions are configuration, testing, reporting, notification, and communication
    - Test development is a client activity
    - Notification can be supported by Test Harness but implemented by tests, e.g., tests use T.H. notification facilities to report to client.
  - These are all cohesive, easily encapsulated, easy to change without breaking other parts
  - Only configuration and reporting are likely to change
  - Functions are conceptually simple, and so, likely to be stable and free of critical errors

# SCF Maintainability

- Repository:
  - Functions are check-in, versioning, metadata management, building, publishing, and communication.
  - Check-in and versioning will use different policies for Project, Company, and Developer Repositories.
    - Should make these rule-based. How?
  - These are all cohesive, easily encapsulated, and independent, so easy to change without breaking other parts.
  - Check-in and Versioning are conceptually the most complicated parts of SCF, and so will need a lot of attention.

# SCF Maintainability

- Client:
  - Functions are check-in, building tests and Test Suites, reviewing results and logs, processing notifications, and communication.
  - Users will want to configure Client UI to support their own work activities.
    - Could make part of the UI a web portal like construct that will allow users to paste gadgets into portal regions, e.g., team test history for month, work calendar.
    - Creating and using Queries into test data need to be flexible, language driven, and invocable by name and scheduled.
  - Clients will be central to running SCF, so must have core functionality running early.

# End of Presentation