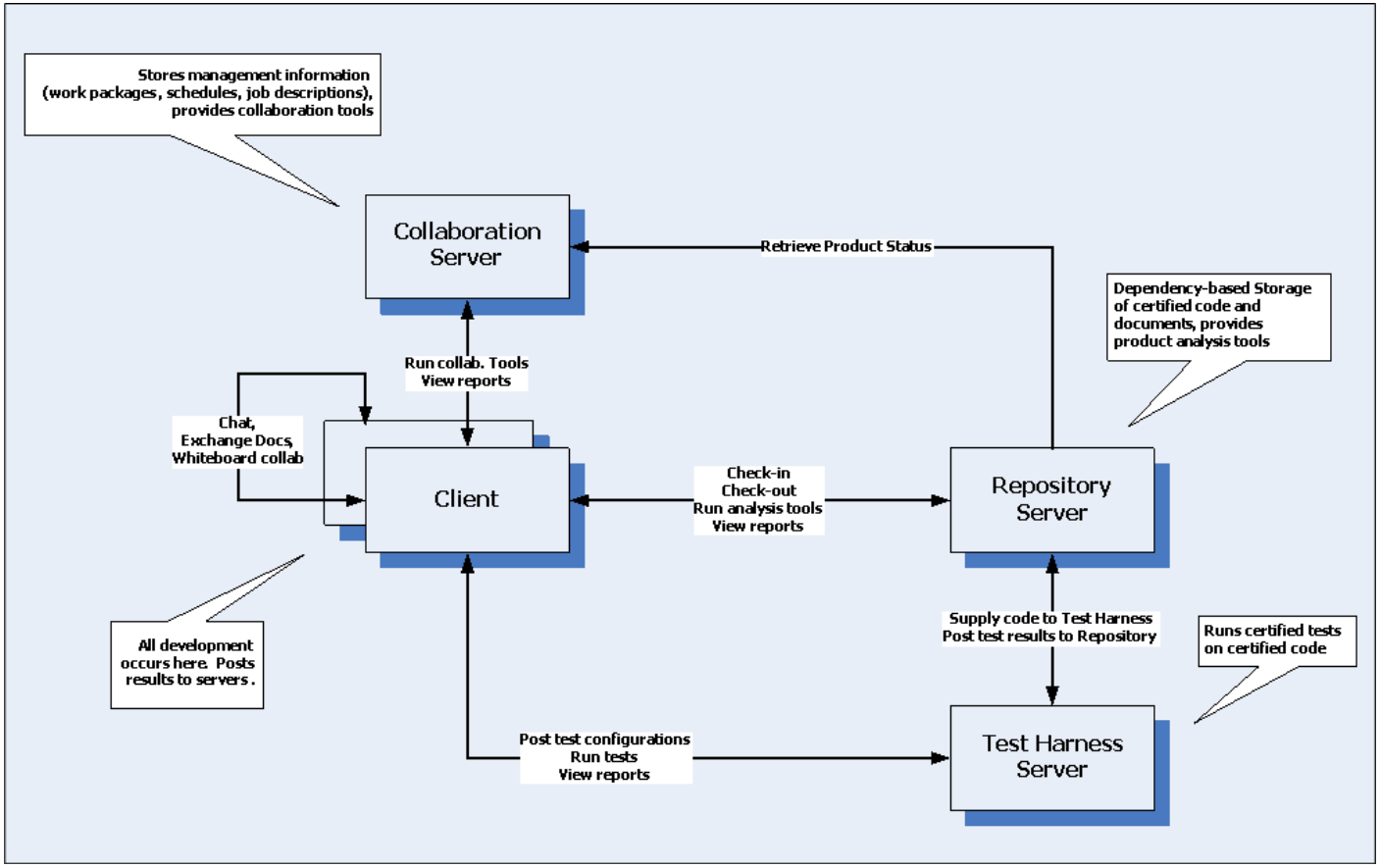
# Enterprise lecture

Jim Fawcett

CSE681 – Software Modeling and Analysis

Spring 2010

- VCRTS based example: Collaboration, Repository, Testharness Servers
  - Interesting questions: collaboration agents? Rule-based process? Rule-based testing?



"Only those who dare to fail greatly can ever achieve greatly", Robert F. Kennedy, 1926 - 1968

- Goals
  - VCRTS supports work scheduling and collaboration (Collab), maintenance of the baseline (Repos), testing (TestH), and development (clients).
    - Everything of value concerning definition of work and scheduling goes on Collab
      - Collab has tools like whiteboard, scheduler with notifications, web cams, etc.
    - Everything of product value goes on Repos
      - Repos provides access to versioned components (a package and all its dependencies)
    - Everything of product value is extracted from Repos and tested on TestH
      - Test results generated by TestH are certified and stored on Repos, linked to component version
  - VCRTS has virtually instant access to all important project data, with the help of some tools like QATS
  - VCRTS is composed of virtual servers, e.g., server is not necessarily a machine boundary and can easily be moved.
    - has an interface and abstract class that defines what a virtual server is
    - supports cloning onto any machine with a receiver installed, e.g., has copy constructor
      - performance prototype would be interesting
    - subset of these mechanics used to synchronize servers
    - does all communication through http or tcp
  - VCRTS can be delivered to customer as maintenance facility
    - may get a subset of what is available to the development team

- Each project has its own VCRTS
  - VCRTS supports cloning servers which may include only part of the original state
    - rule-based cloning?
  - each team can have its own VCRTS
  - but only project VCRTS has certified product: code, documentation, ...
  - company has company-wide VCRTS to hold and distribute reusable code
- VCRTS supports rule-based management of Project and Product

- Organizing principles
  - **continuous integration and test via built in test functionality**
    - supports topdown testing - must be batchable with no human intervention
      - each package supports rTest (regression level) and uTest (unit level)
      - run top test driver for current baseline system – regression level test
      - top test driver calls regression test driver in all packages it calls
      - recursively works toward leaf elements
      - need to mark visited packages so don't loop forever on mutual dependencies
      - if any fail, start that subtree with more detailed uTest tests.
      - no logging for regression accept at the top test level
      - needs internal consistency checks, ala Design By Contract (DBC).
        - Will parser like structure help here? Maybe - can add test rules at any time without rebuilding rest of system.
      - How do we specify these checks? Do we need to?, or just let test evolution have its way!
      - developer can build a test driver at any component node. Checkin should fail if drivers are not supplied or do not build.
      - test drivers should be verified with (sensible) mutation testing
    - must be able to support class, package, subsystem, and system invariants
    - each package provides a function that implements ITest, will normally be built as a DLL

- **documentation and builds are encapsulated.**
  - Repository components are linked with XML manifests containing metadata
  - Each package documents itself only, with abstract treatment of those packages it calls.
    - spec, design, test document
    - perhaps design and test documentation can be (nearly) generated automatically by recursing through metadata
  - Each package provides a build script that use its callees build scripts recursively
  - builds: Static vs dynamic linking can be set with configuration in metadata
    - supports packaging that adapts to evolving baseline
  - storage: Test drivers link to their called code (obviously), test data links to test driver
  - each package has version stamp it supplies to test data.
- **Work Scheduling (Collab)**
  - holds partitioned schedule, reviews, work packages, collaboration products
  - authority roles based on work package
  - Work package structure linked, similar to component model on Repos
  - Each leaf node work package describes the work of one individual with tangible completion condition
- **Repository holds components** (pull model for access with file caching)
  - repository holds all versions in linked structure so earlier system is there all linked up.
  - linkages defined by XML manifests with metadata
    - manifest has brief description of component
    - link to more detailed documentation (for just top level)
    - links to files or lower level manifests
  - could we use change sets to reduce storage and still quickly spawn an old version?
  - baseline test only from repository

- VCRTS
  - servers are virtual with replication
  - all communication message-passing is via web.
    - Performance issue?
  - notifications are ordinary messages
  - anyone can have a repository, testbed, and collab server
  - only project VCRTS holds certified products

## - Layering

- message-passing communication service
  - notification based on message passing
- virtual server service
- batch spiral testing service
- linkage scanning service
- display service

## - Example of prototype code

- file transfer via pull model with file caching
- repository navigation via component manifests