COSC 310: Software Engineering

Dr. Bowen Hui
University of British Columbia Okanagan
bowen.hui@ubc.ca

Implementation Considerations

- what are some issues to consider during implementation?

Implementation Considerations

- what are some issues to consider during implementation?
  - on track
  - meets requirements and design
  - tested "properly"
  - versioning
  - duplicate code segments
  - overwriting each other's code

Implementation Issues

- reuse
  - reusing existing components or systems
- configuration management
  - keeping track of different versions of software components being developed
- host-target development
  - development vs production platform

Issue #1: Reuse

- between 1960s - 1990s, new software mostly developed from scratch
- only reuse was functions and libraries
- this approach is costly and time consuming
- new ways of reuse needed

Levels of Reuse

- abstraction level
  - doesn't involve software
  - reuse knowledge of successful designs and abstractions
  - Ex: architectural pattern
- object level
- component level
- system level

Levels of Reuse

- abstraction level
- object level
  - reuse objects from a library
  - Ex: process mail messages in Java - can use objects and methods from a JavaMail library
- component level
- system level

Levels of Reuse

- abstraction level
- object level
- component level
  - components = collections of objects and object classes that operate together to provide related functions and services
  - can adapt or extend another component
  - Ex: use framework to build GUI
- system level

Levels of Reuse

- abstraction level
- object level
- component level
- system level
  - reuse entire application
  - usually involves some configuration
  - most commercial systems are now built this way
  - Ex: Oracle's customer relationship management systems
  - new field emerged: **software customization**

Is it worth it?

- what are some benefits of reuse?
- costs?

Benefits of Reuse

- faster
- fewer development risks
- lower (monetary) costs
- more reliable (existing pieces already tested)

Costs of Reuse

- time to search for opportunities to reuse
- time to search for software that may/may not suit your needs
- additional testing in your environment
- cost of integrating code pieces that may employ different assumptions
- monetary cost of purchasing software
- monetary cost of customization

Issue #2: Configuration Management

- don't interfere with team members' work
- if two people work on the same component, their changes need to be coordinated
- life without configuration management:
  - [member 1] 6am-2pm: code
  - [members 1,2,3] 2pm-3pm: compilers class
  - [members 1,2,3] 3pm-4pm: team meeting
  - [member 2] 4pm-11pm: code
  - [member 3] 11pm-6am: code

true story!

Purpose of Configuration Mgmt

- prescribe process for managing changes during the development
- ensure everyone can access most recent version
- maintains older versions (so can revert back)
- main purpose: support integration process
- applies to both code and documentation

Software Configuration Management (SCM)

- Configuration management is the process of identifying, organizing and controlling modifications to the software artifacts being built by the project team
- The items that comprise of all information produced as a part of the software process are collectively called a software configuration

Sharing the software development work (SCM Scenario)

Generate a list of Active/Inactive Students

Generate correspondence

Printing test exam schedules

Repository

Create/Modify Student Database

Software Configuration

- What are the various work products?
  - computer programs
    - both source and executable forms
  - documents that describe the program
    - targeted at technical practitioners and users
  - data contained within the program or external to it
- As software processes progress, the number of software configuration items (SCI) grows rapidly

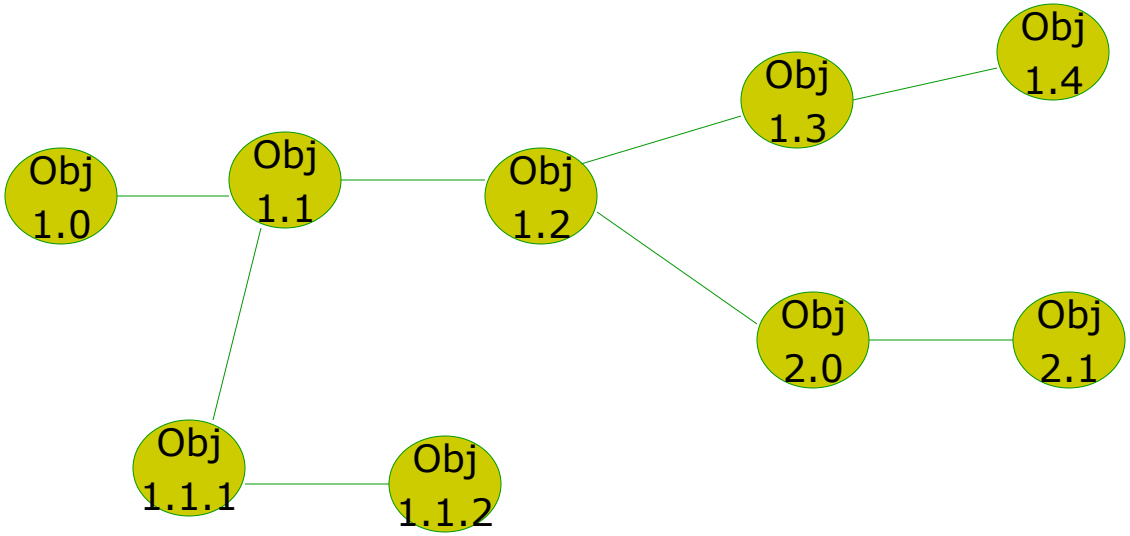
Role of a repository

- Data Integrity
  - Consistency among related objects
- Information Sharing
- Data Integration
- Documentation Standards

Requirements of an SCM Process

- Five Principles tasks:
  - Identification of Software Configuration Items
  - Version Control
  - Change Control
  - Configuration auditing
  - Reporting

Ex: Evolution Graph - Version Control



Object Description:

SCI type (e.g., document, program and data)

a project identifier

Change and/or version information

Change Control (internal)

- Access Control
  - Governs which software engineers/developers have the authority to access and modify a particular configuration object
- Synchronization Control
  - Helps to ensure that parallel changes performed by two different people do not overwrite one another
- These two controls apply to only baseline SCIs

Centralized Model

- Traditional revision control systems use a centralized model, where all the revision control functions are performed on a shared server.
- If two developers try to change the same file at the same time, without some method of managing access the developers may end up overwriting each other's work.
- Solutions:
  - file locking (check-in/lock; check-out/unlock)
  - version merging (e.g., used in CVS)
- Alternative model: Distributed revision control

Fundamental Activities

- version management
  - keeps track of different versions of software components
  - include facilities to coordinate development by several programmers
- system integration
  - help define what versions of components are used to create each system version
  - this description is used to compile and link required components automatically
- problem tracking
  - allow users to report bugs
  - allow developers to see who's working on these problems and when they are fixed

Examples of CASE Tools

- integrated tool to support all three activities:
  - ClearCase

- separate tools:
  - Subversion - version management
  - make - system integration
  - Bugzilla - problem tracking

Host-Target Development: Motivating Example

- You joined Amazon as a programmer. In the first week, you are asked to change an online payment feature. From your computer, you code, test, deploy. In the next minute, millions of people are using the site and they cannot purchase items anymore.

- What went wrong?

Main Message

- don't ever make changes to a working production environment without ensuring the changes are correct and well-tested!

- always ask yourself:
  - "If I made this proposed change and the system became totally unusable for all my users during normal business hours, would that be okay?"

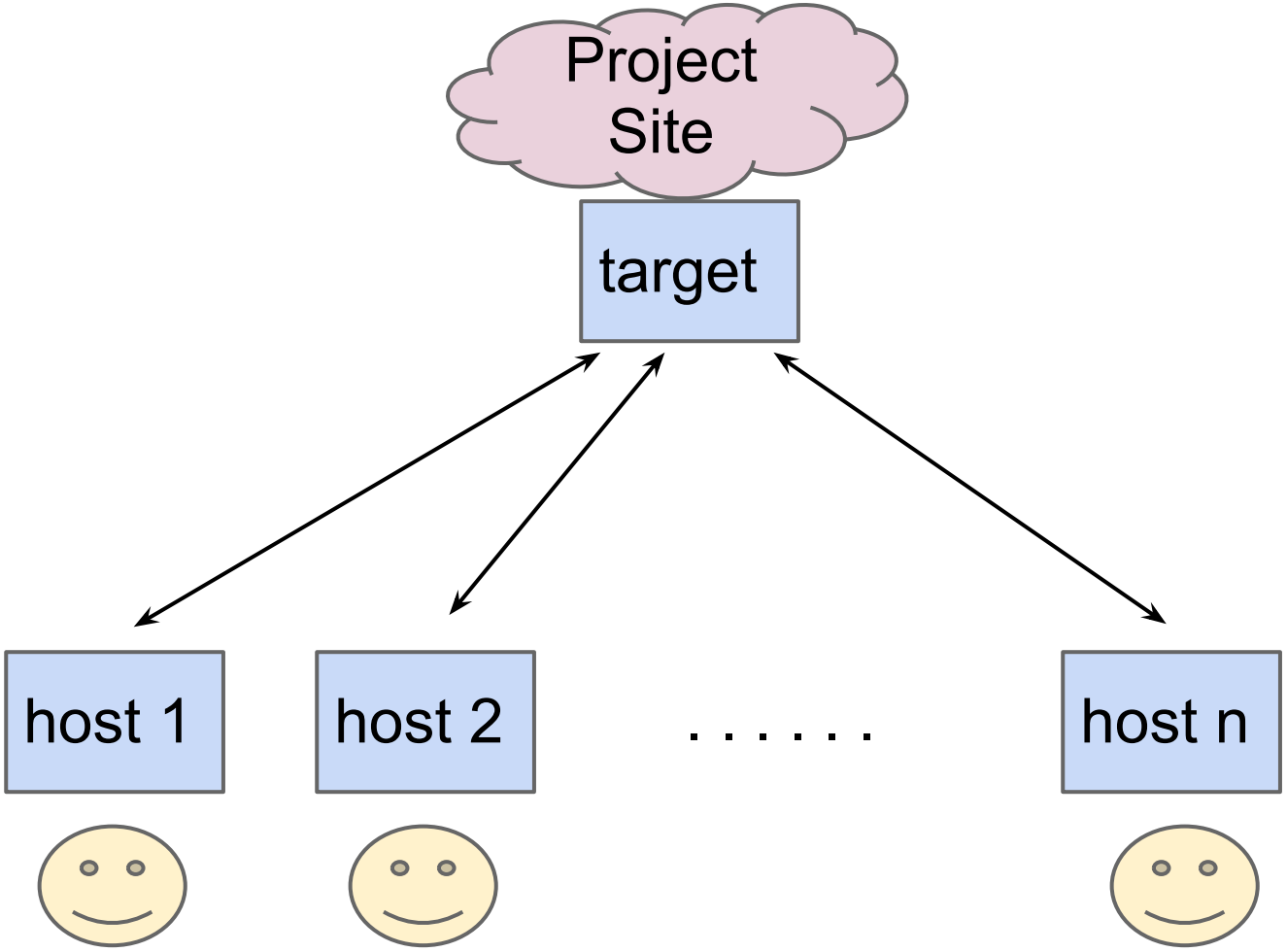
Issue #3: Host-Target Development

- industrial development is based on a host-target model
  - software developed on one computer (host)
  - but runs on a separate machine (target)
- e.g., website project
  - host = your computer (localhost:3000)
  - <upload your code>
  - target = web server (www.xyz.ca)
- each machine may differ in:
  - installed OS
  - DBMS
  - IDE

"deploy"

platform

Minimum Host-Target Model Visually



Developers each work on a separate host

Implications on Testing

- sometimes the development and execution platforms are the same
  - can develop and test on the same machine

- more commonly, the platforms are different, so to test, you need to either:
  - move software to execution platform
  - run simulator on dev machine

General Setup: 4 Tiers

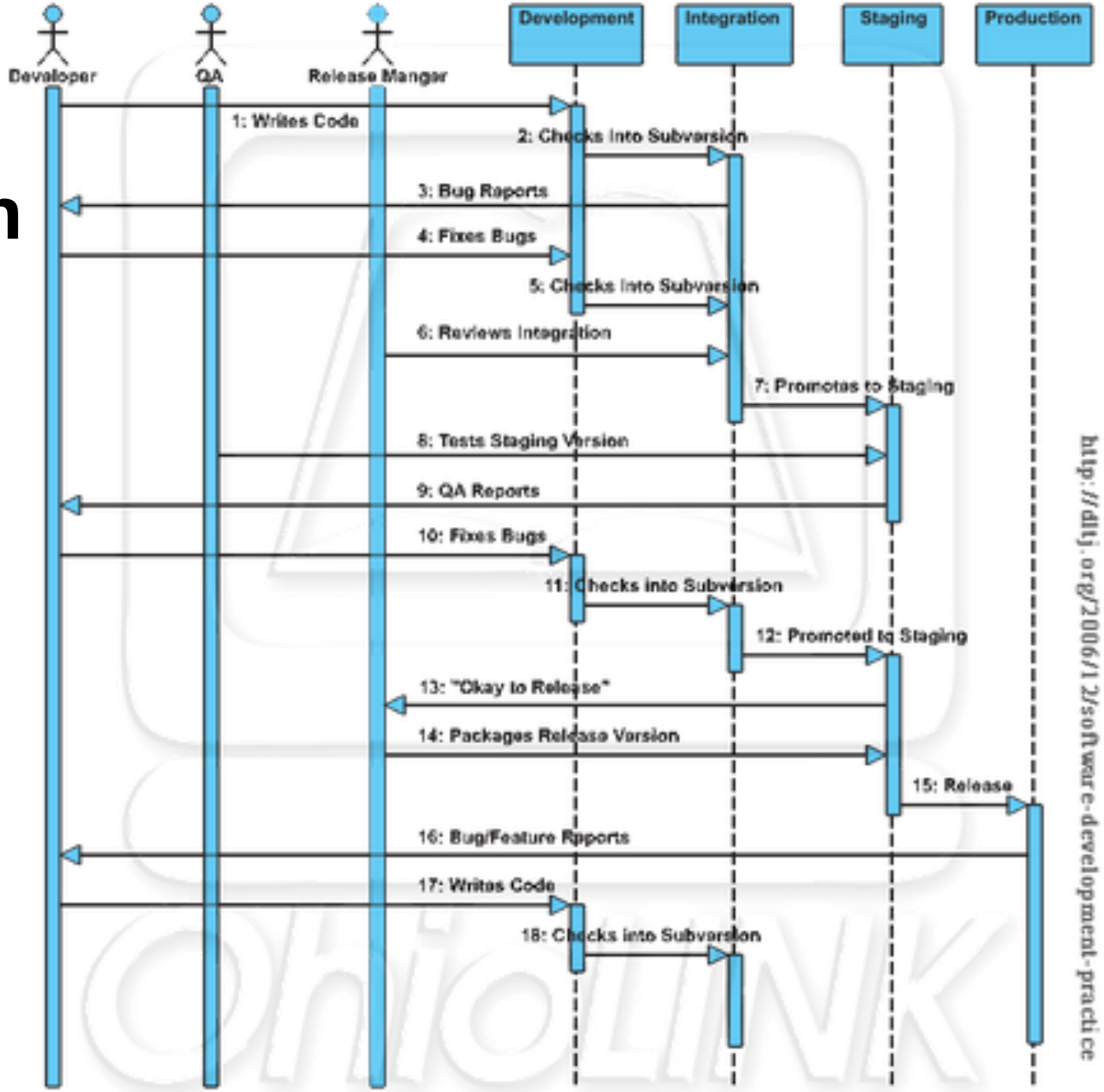
- development
  - working platform for individual developers
  - radical changes won't affect rest of the team
- integration
  - common platform where all the developers to commit code changes
  - goal: validate entire project
- staging
  - aim is to use platform identical to that of production
  - simulates production environment as closely as possible
  - often used for demo or training purposes
- production
  - final platform that all the users see

Tier Resources

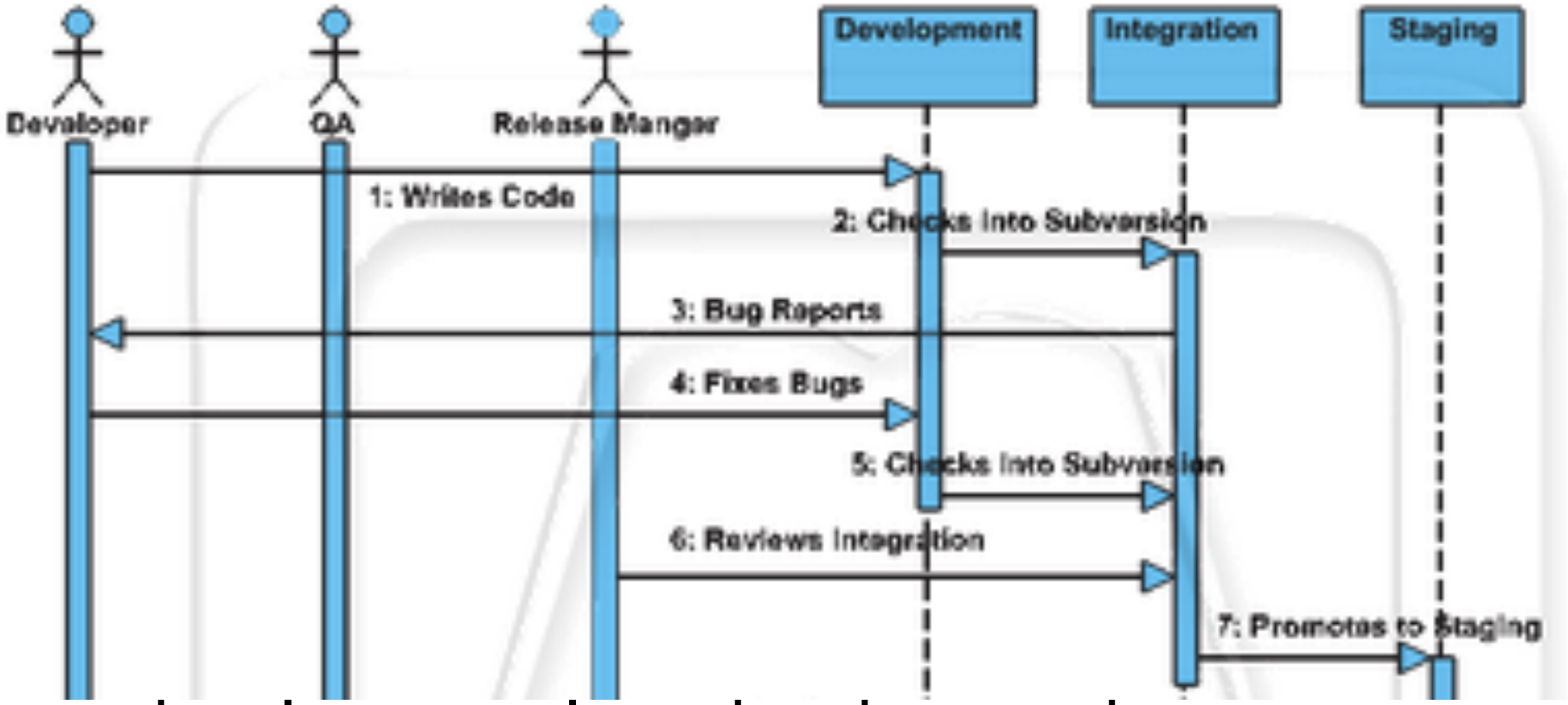
- development and integration:
  - independent copy of DB
  - limited subset of data
  - useful for **boundary testing**

- staging:
  - (ideally) identical configuration of production platform - software, DBMS, hardware
  - independent copy of DB
  - supports QA testing
  - performance testing provides accurate forecast of capacity

Example Interaction

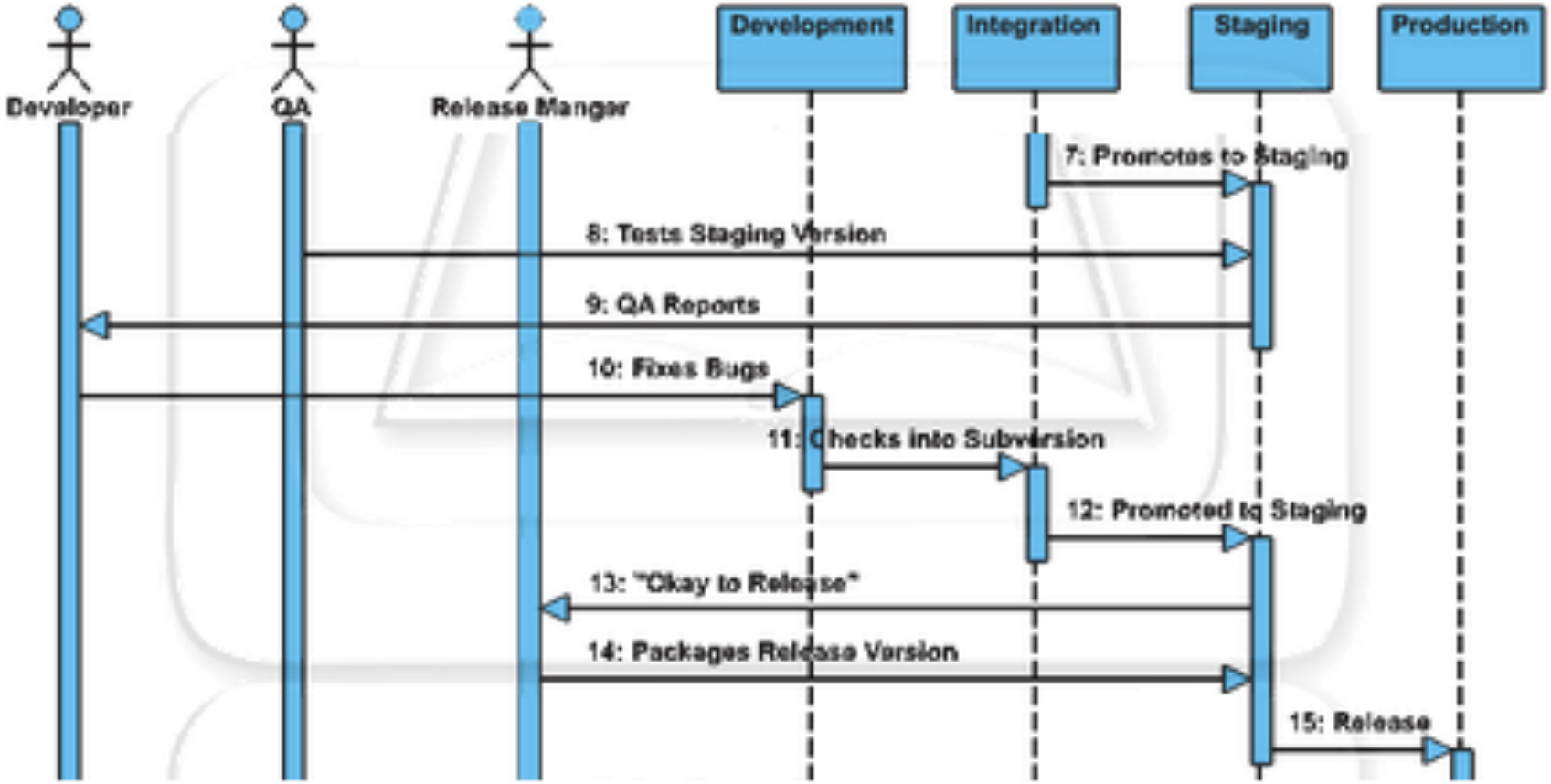


Development



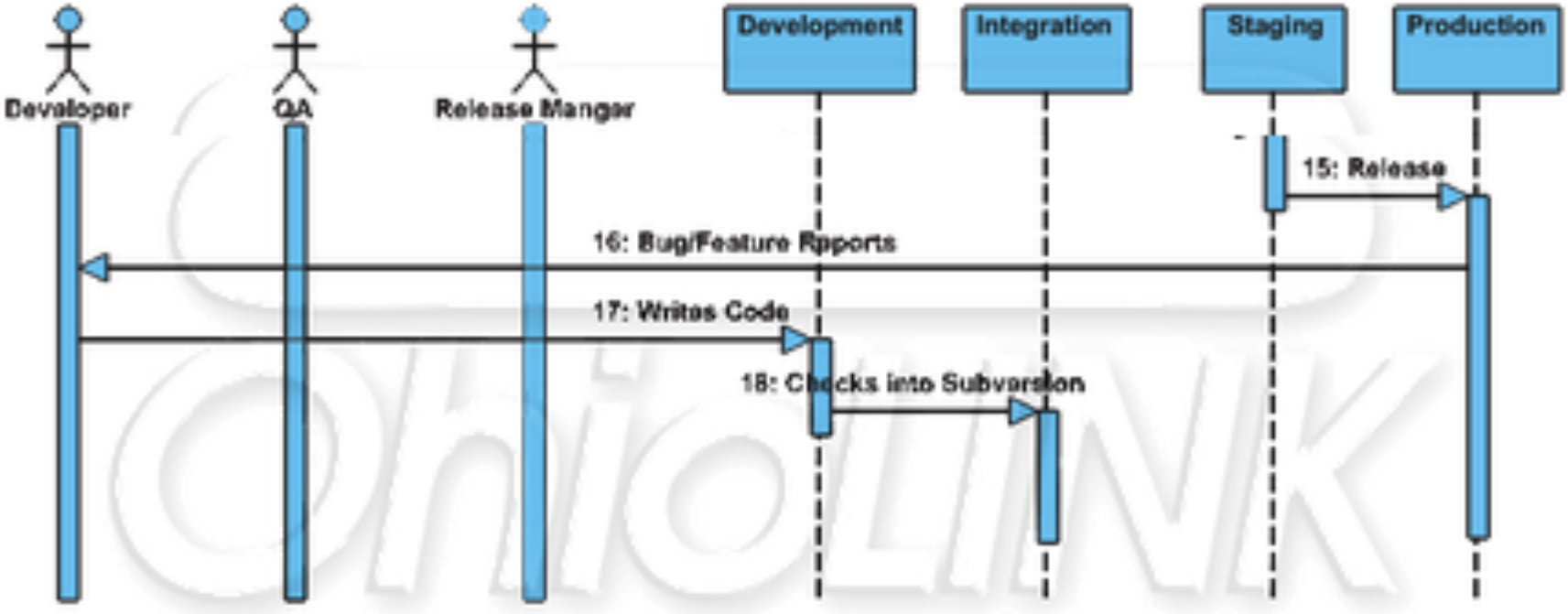
- developers only make changes to Development and Integration environments

QA Testing and Stage Advancement



- QA occurs in Staging
- Release Manager okays deploy to next stage

Ongoing Maintenance



- bug fixes after deployment
- process repeats

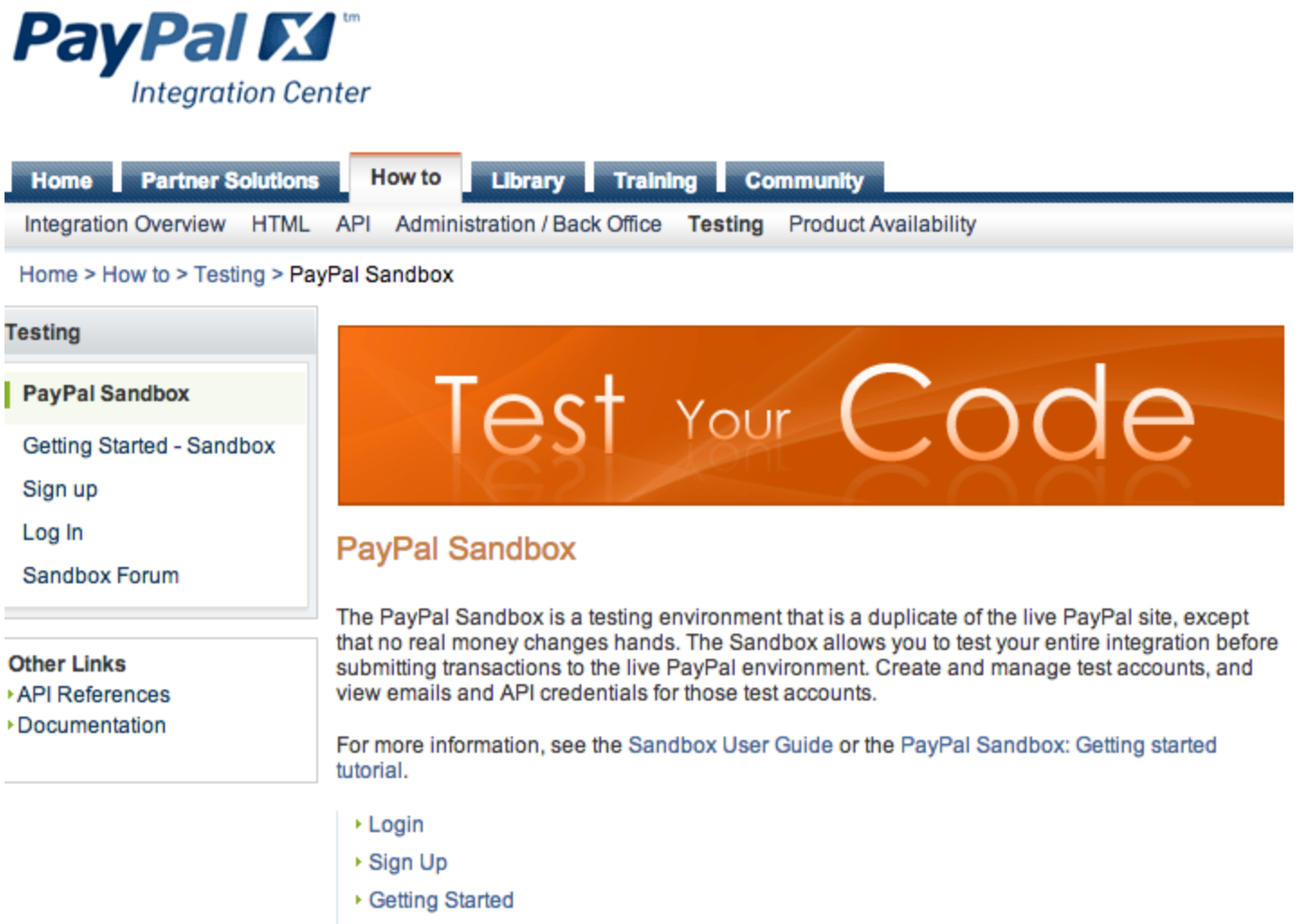
Multiple Versions

- getting to Production:
  - multiple versions in Staging
- getting to Staging:
  - multiple versions in Integration
- getting to Integration:
  - multiple versions in Development

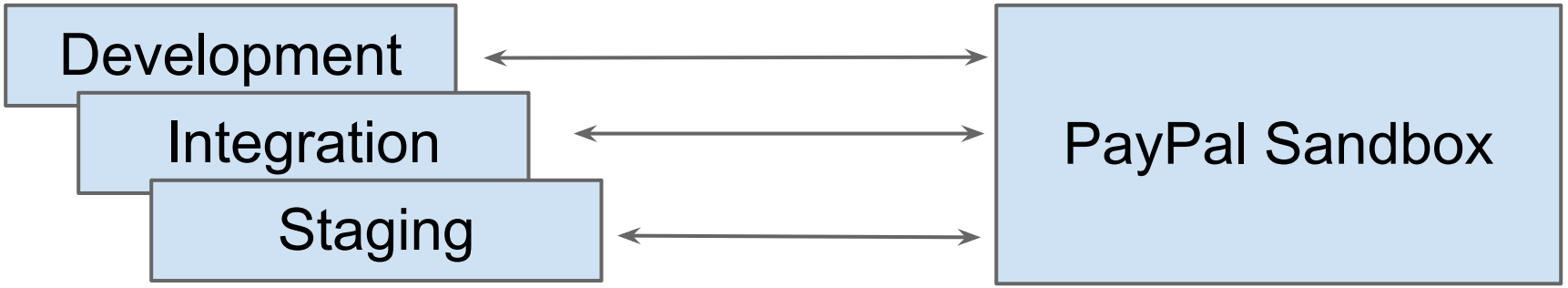
- sometimes Development and Integration are combined

- how to manage all this?

Example: PayPal Sandbox



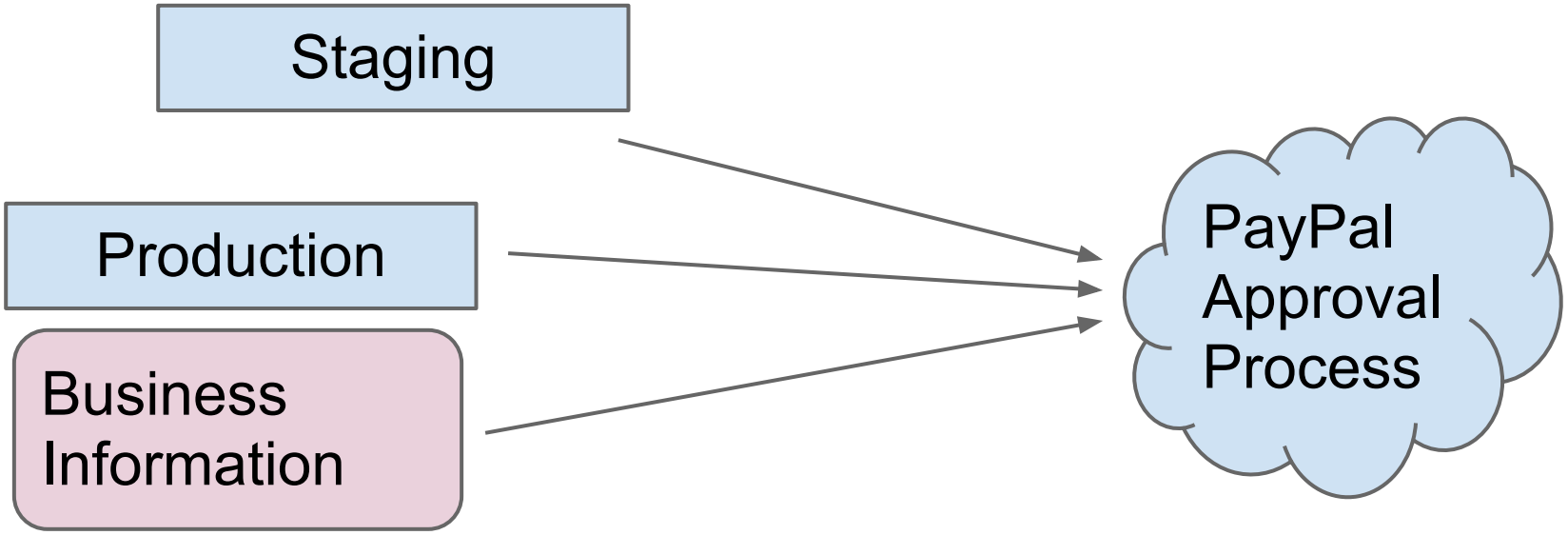
Integration with PayPal



Step 1:

- get your system in the Staging environment to work with PayPal sandbox
  - Development / Integration environments also need to work with PayPal sandbox
  - process as usual, but stop at Staging

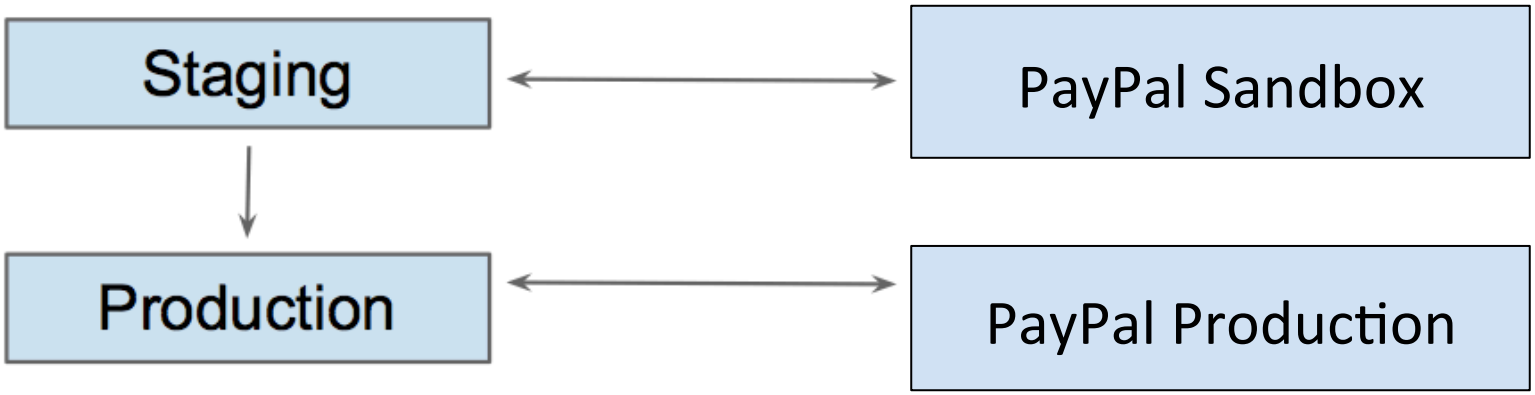
Integration with PayPal



Step 2:

- submit for approval

Integrating with PayPal



Step 3:

- deploy changes to your Production environment
  - promote to Production (accounts etc. configured for PayPay Production)

References

- http://dltj.org/article/software-development-practice/
- http://www.razorleaf.com/2009/12/prod-test-dev-environments/