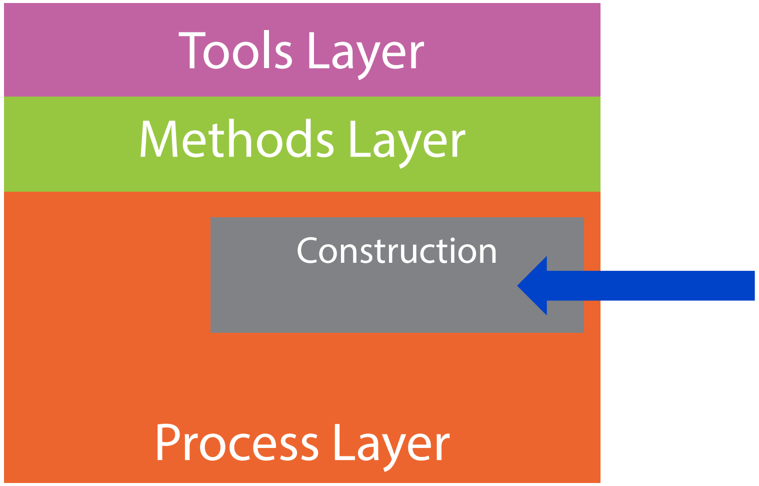
Construction

Mohamad Halabi
@mohamadhalabi

pluralsight
hardcore dev and IT training

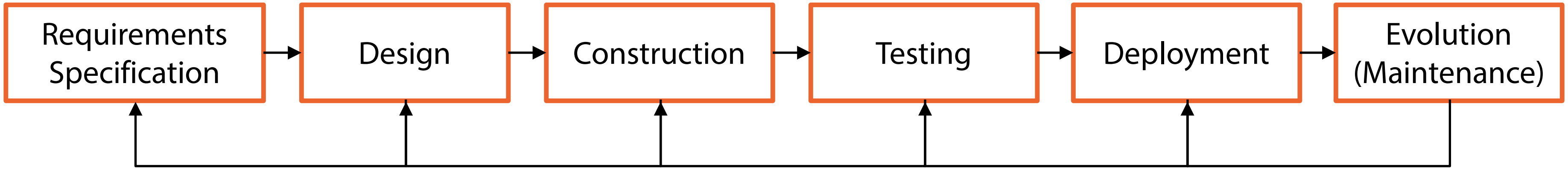
Introduction



From Design to Construction

- Construction: Transforming design into an executable program
- Relationship between Design and Construction activities varies based on the selected process model

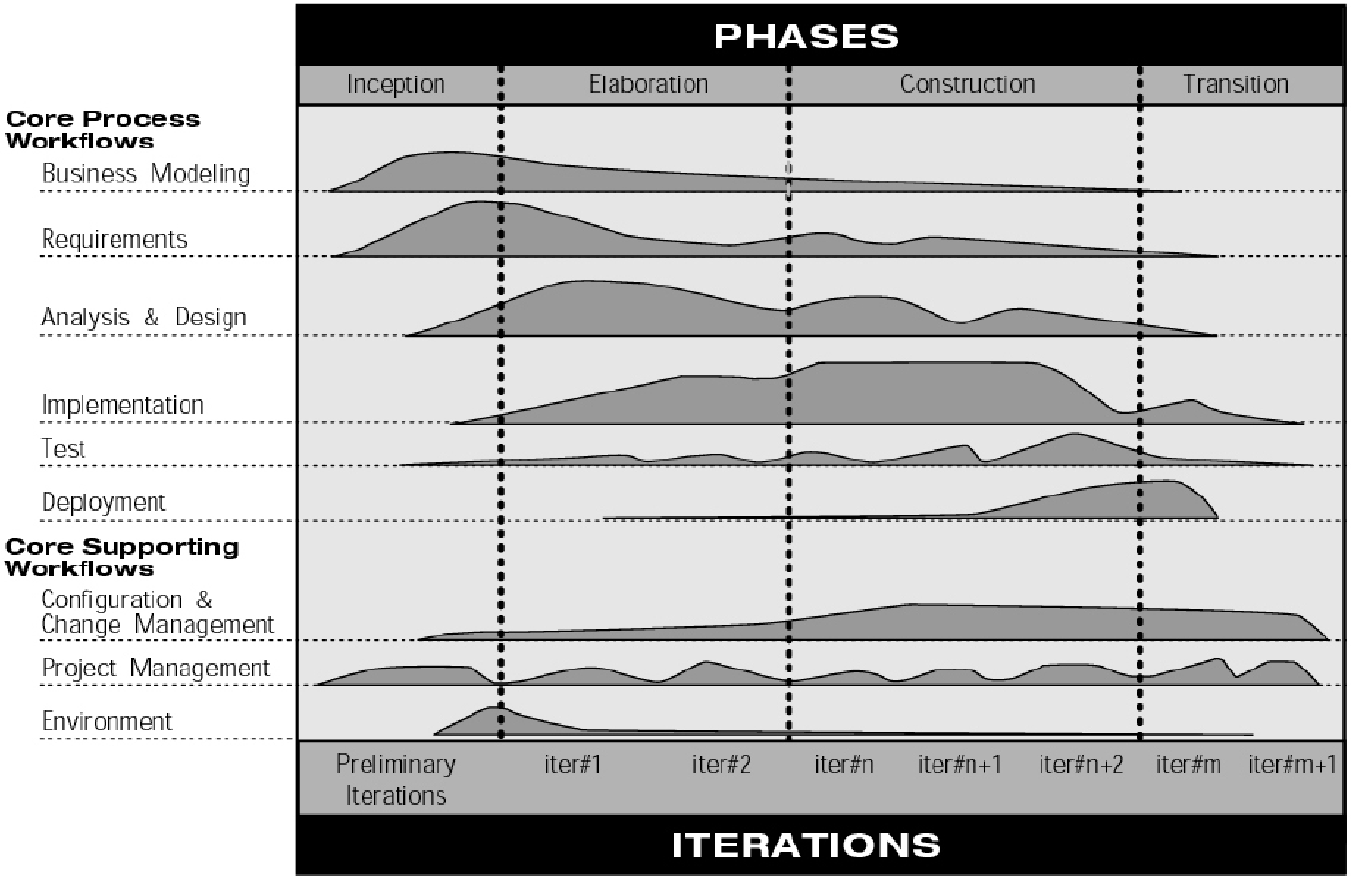
From Design to Construction: Waterfall



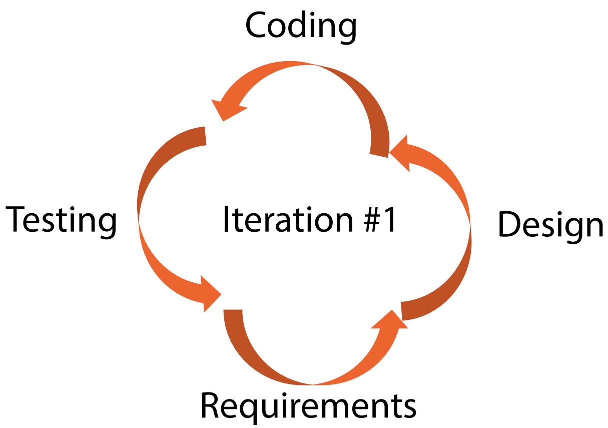
- **Construction refers only to coding**

From Design to Construction: Iterative/RUP

- Construction: Mix of requirements, design, coding, and testing activities



From Design to Construction: Iterative/Scrum



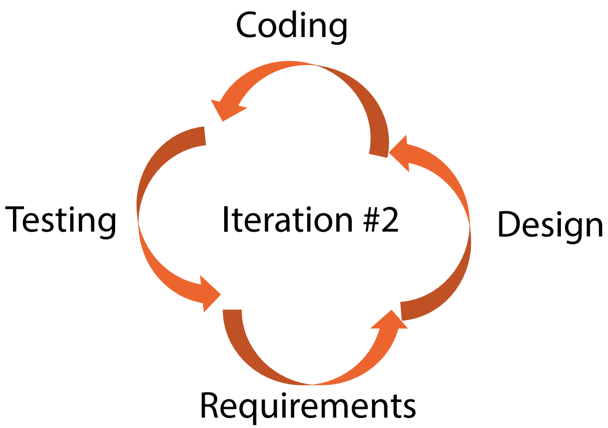
Coding

Testing

Iteration #1

Design

Requirements



Coding

Testing

Iteration #2

Design

Requirements

So What Do We Mean by Construction?

- **Construction: Specifically coding and its related principles**
- **Recall module 2: Discussion of Software Engineering activities is independent of any specific process model**
  - Principles of activities are the same
  - Process models bring their own flow and emphasis of activities

“*Any fool can write code that a computer can understand. Good programmers write code that humans can understand*”

— **Martin Fowler**

Minimizing Complexity

- **Coding is not about being ‘clever’**
- **Do not aim for ‘fancy’ code**
- **Always go for simple and readable code**
  - Easy to maintain, review, evolve, test, and document
- **Some guidelines to writing simple code:**
  - Following standards
  - Following design best practices

Standards

- **Follow coding standards**
  - Ex: Naming, layout, and commenting standards
- **Standards can be:**
  - Platform-specific (ex: c# coding standards)
  - External organizations (ex: modeling using UML)
  - Built internally
- **Standards must be applied consistently**
- **Adopt external standards whenever possible**
- **Ex: C# coding conventions: http://msdn.microsoft.com/en-us/library/ff926074.aspx**

Design Best Practices

- **Following coding best practices is essential**
  - General (ex: modular design and decomposition)
  - Programming methodology-specific (ex: SOLID of OO programming)
  - Platform/language-specific idioms and practices (ex: .NET design guidelines for exception handling, memory management, etc…)
- **Language idioms capture solutions to language-specific problems**
  - Equivalent to architectural styles (at architecture level) and design patterns (at detailed design level) at the coding level
- **Resources:**
  - http://www.pluralsight.com/courses/encapsulation-solid
  - http://msdn.microsoft.com/en-us/library/ms229042%28v=vs.110%29.aspx

Extensible Code

- **Change is inevitable**
- **Writing extensible code is a key aspect**
  - Code adapts to changes with minimum impact on overall structure and behavior
- **Techniques**
  - High cohesion
  - Low coupling
  - Abstraction and encapsulation
  - Configuration files and dynamic rules
  - Decouple code for environment and infrastructure
  - Language/platform specific techniques – such as C# Attributes and Reflection

Reuse

- **Enables productivity, cost saving, and increased quality**
  - □ Writing reusable code
  - □ Reuse other sources code
- **Writing reusable code**
  - □ Writing components that are independent from a specific system or subsystem
  - □ Ex: Logging and exception handling components
- **Reuse other sources code**
  - □ Utilizing community or 3rd party components
  - □ Ex: Telerik

Resources

- **Cohesion and coupling: http://msdn.microsoft.com/en-us/magazine/cc947917.aspx**
- **“*Code Complete: A Practical Handbook of Software Construction*” by Steve McConnell**
- **Pluralsight’s “*Clean Code: Writing Code for Humans*” by Cory House**

Testing

- **Testing is a Software Engineering activity**
- **However, do not leave testing entirely till after construction**
  - Huge cost in finding and fixing problems
- **In Construction a subset of test types must be implemented**
  - Reduces gap between faults creation and detection
- **Two testing types must be part of Construction:**
  - Unit Testing
  - (Limited scope) Integration Testing

Unit Testing

- **Considered more of a coding practice than a testing practice**
- **Unit Testing:**
  1. Write a function or other block of code
  2. Create a unit test that verifies:
     - Behavior of code in response to standard, boundary, and incorrect input data
     - Explicit and implicit code assumptions
- **Do not wait until you have an executable code**
  - Testing occurs on code-block level

Unit Testing

- **Q: How to test un-executable code?**
- **A: Via a “stub” and a “driver”**
  - Driver simulates a calling unit
  - Stub simulates the called unit
- **Testing frameworks provide stubs and drivers – such as Visual Studio**
- **Test Driven Development (TDD) development practice:**
  - Create unit test before writing the code to be tested
  - Unit tests also serve as design documentation and functional specification
- **Resources:**
  - http://msdn.microsoft.com/en-us/library/hh694602.aspx
  - Search “unit testing” in Pluralsight library

Integration Testing

- **Bringing together individual components and testing system or sub-system functionality in terms of these interacting components**

- **In Construction, scope of Integration Testing is limited**

- **Example:**
  - □ Component A decrypts a message
  - □ Component B transforms the output of component A
  - □ Testing the integrating between A and B is the only way to guarantee that both components perform their intended functional scope

- **Integrating entire subsystems in a simulated real-time environment, happens in the Testing phase**

Continuous Integration (CI)

- **A development practice where team members integrate their work regularly**
- **Flow:**
  1. Team members collaborate via a version control (ex: TFS)
  2. Code changes are checked-in multiple times per day
  3. Depending on CI configuration:
     1. CI can run every time code is checked
     2. Multiple times per day
     3. Once per day – for example at midnight
     4. Mix; for example at every check-in plus once at midnight
  4. Once CI runs, automated tests run (ex: unit and integration tests)
  5. Problems are caught and automatically assigned for resolution

Continuous Integration (CI)

- **Q: How to automate the CI process?**
- **A: Via a Build Server**
  - Automatic integration of code
  - Running the automated tests
  - Automatic assignment of problems to team members
- **TFS contains a Build Server – called Team Foundation Build**
- **Martin Fowler on CI: http://martinfowler.com/articles/continuousIntegration.html**
- **TFS automated builds: http://msdn.microsoft.com/en-us/library/hh395023%28v=vs.110%29.aspx**

Continuous Integration (CI)

- **Plan CI such that it does not negatively affect the performance of the version control system**

- **Especially a concern when integration will run over huge number of source code files**
  - □ Follow best practice of Build Server deployment and configuration
  - □ Chose best times for build schedules

- **TFS Build Server configuration: http://msdn.microsoft.com/en-us/library/ms181712.aspx**

Automating Process Models

- **Software Engineering teams** collaborate **on different types of activities (requirements gathering, design, development, testing, etc…)**
- **This collaboration must be controlled and monitored**
- **Therefore automation becomes essential**
  - When team size grows
  - When team members work on different projects
  - When project managers need to control and monitor multiple projects

Automating Process Models

- **TFS forces the selection of a process template on project creation**
- **Process templates:**
  - Specify process models
  - Define work item types
  - Define reports for planning and tracking
- **Work item types refer to Software Engineering activities**
  - Ex: Requirement, Task, Bug, Change Request, etc…
- **TFS provides Agile and CMMI templates**
  - New process templates can be created – for example to support RUP

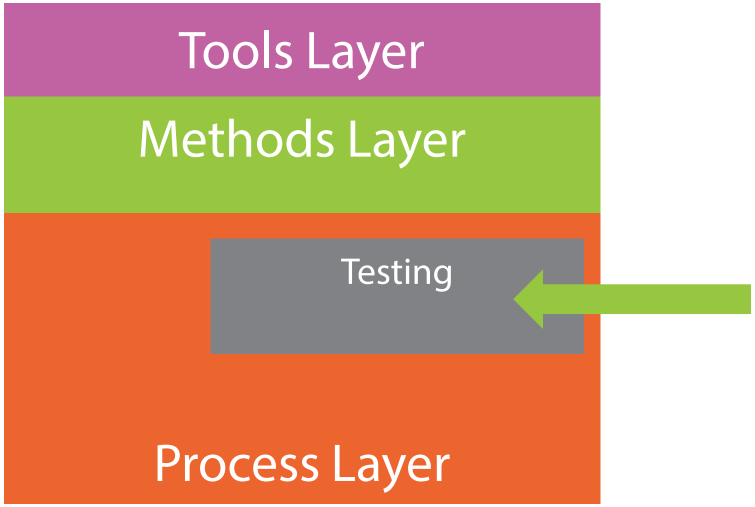
Automating Process Models

- http://msdn.microsoft.com/en-us/library/ms400752.aspx
- http://msdn.microsoft.com/en-us/library/ms243782.aspx
- Search “TFS” in Pluralsight library

Summary

- **Construction is about generating working software**
- **Activities vary based on the process model**
  - Waterfall: coding only
  - RUP: requirements, design, testing, coding. However, emphasis is on coding
  - Scrum: no clear boundaries. Short sprints contain all activities
- **However, regardless of the process model, principles are the same**
  - Coding principles facilitates simple, best practice, reusable, and extensible code
  - Unit testing and limited scope of integration testing
  - Continuous integration
  - Automating process models

What’s Next?


Tools Layer
Methods Layer
Testing
Process Layer