Code Complete 2:
A Decade of Advances in Software Construction

www.construx.com

Construx®

Delivering Software Project Success™

Introduction

History of Code Complete

- ❖ Published in 1993
- ❖ Comprehensive survey of construction practices in 1993
- ❖ Revised in 2003-04
- ❖ Comprehensive survey of construction practices in 2004
- ❖ What have we learned in 10 years?

Underlying Distinction

- ❖ “Technology” knowledge (accidental)
  - ◆ Short lived
  - ◆ Readily acquired
- ❖ “Principles” knowledge (essential)
  - ◆ Longer lived
  - ◆ Not-so-readily acquired

Where Does Code Complete 2 (CC2) Fit into this?

- ❖ Attempt in 1993 was to capture lasting knowledge of software construction (principles knowledge)
- ❖ I've asserted for many years that 95% of the content of CC1 is still relevant
- ❖ Is this true?

Overview

- ❖ Worst Ideas of 1993 and 2004
- ❖ A Decade of Advances in Construction
- ❖ Construction Realities as of 2004

Worst Ideas of 1993 and 2004

Some of the Worst Ideas of 1993

- ❖ Hacking
- ❖ “Optimize as you go” programming
- ❖ “All design up front” programming
- ❖ “Design ahead” programming
- ❖ Flow charts
- ❖ Automatic programming
- ❖ Formal methods as a cure all
- ❖ Calling everything “object oriented”

Some of the Worst Ideas of 2004

- ❖ Hacking
- ❖ “Optimize as you go” programming
- ❖ No design up front
- ❖ Planning ahead to refactor later
- ❖ Automatic programming
- ❖ Extreme Programming as a cure all
- ❖ Calling everything “agile”

Worst Ideas, 1993 vs. 2004

1993	**2004**
❖ Hacking	❖ Hacking
❖ “Optimize as you go” programming	❖ “Optimize as you go” programming
❖ “All design up front” programming	❖ No design up front
❖ “Design ahead” programming	❖ Planning ahead to refactor later
❖ Flow charts	❖ -
❖ Automatic programming	❖ Automatic programming
❖ Formal methods as a cure all	❖ Extreme Programming as a cure all
❖ Calling everything “object oriented”	❖ Calling everything “agile”

A Decade of Advances in Software Construction

How Far Have We Come In 10 Years?

- ???

1. Daily Build and Smoke Test

- ❖ Institutionalizes incremental integration

2. Code Libraries

- ❖ Good programmers have always done this
- ❖ Now supported by languages (C++, Java, VB)

3. Visual Basic

- Visual programming innovation
- The first language to make widespread use of COTS components
- Only language to learn syntax lessons from Ada (case statements, control statements, etc.)

4. The Web, for Research

- FAQs
- Discussion groups
- Searchability in general

5. Open Source Software

- ❖ Great aid to programmers
- ❖ Beware of legal issues

6. The Web, for Code Libraries (including Open Source)

- ❖ Combination of searchability + Open Source + vast resources = genuine innovation

7. Test-First Development

- Shortens time to defect detection
- Increases personal discipline
- Nice complement to daily build & smoke test

8. Refactoring as a Discipline for Modifications

- Provides a discipline for making changes

9. Widespread Use of Incremental Development Practices

- Concepts were well known in 1993
- Practice is well known in 2004

10. Faster Computers

- Compare CC1 performance benchmarks to CC2 benchmarks
- Implications for optimization

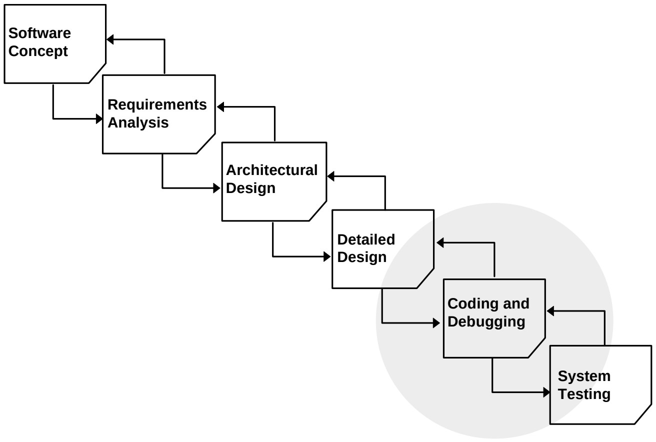
Construction Realities as of 2004

Overview of 2004's Construction Realities

- Construction as a Topic
- Individual variation
- Personal discipline
- Simplicity
- Defect cost increase
- Importance of design
- Technology waves
- Incremental development
- Toolbox metaphor

"Construction" is a Legitimate Topic

Software "Construction" – Used to Look Like This


Software Concept

Requirements Analysis

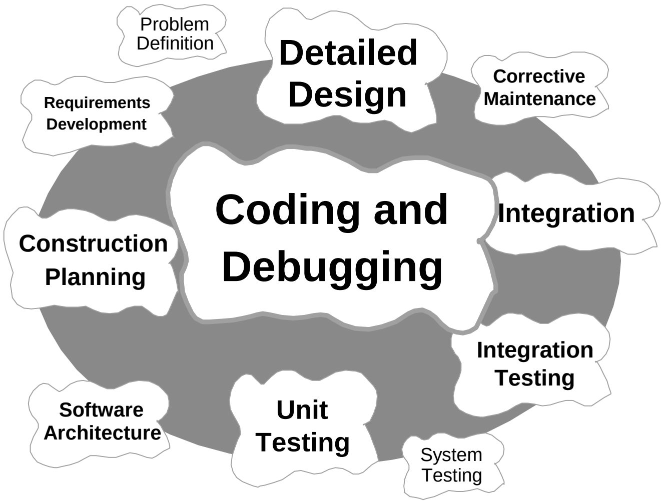
Architectural Design

Detailed Design

Coding and Debugging

System Testing

Software “Construction” – Now Looks Like This



Distinction Between Activities and Phases

- Activity != Phase (<> for VB programmers)
- Talking about “Construction” as an activity does not imply a distinct phase
- Differentiating between kinds of activities is extremely helpful

Individual Variation Is Significant

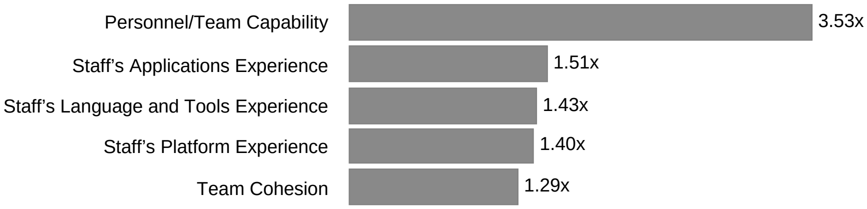
Where do Variations Exist?

Researchers have found 10:1 variations in:

- **Coding speed**
- **Debugging speed**
- **Defect-finding speed**
- **Percentage of defects found**
- **Bad-fix injection rate**
- **Design quality**
- **Amount of code generated from a design**
- **Teamwork**
- **Etc.**

Significance of Individual Variability

- ❖ **According to Cocomo II calibrations, worst personnel will require 1,380% as much effort to produce software as best personnel**



Source: *Software Cost Estimation with Cocomo II,* Barry W. Boehm, et al, Prentice Hall, 2000

Key Skills of an Expert Programmer

- ❖ **Designing**
- ❖ **Flushing out errors and ambiguities in requirements**
- ❖ **Coding (naming, formatting, commenting, compiling)**
- ❖ **Integration**
- ❖ **Debugging**
- ❖ **Unit testing**
- ❖ **Using tools for all of the above**

Personal Discipline Matters

Programmers Have Not Gotten Better at Predicting the Future

- ❖ Optimizing “As you go”
- ❖ “Design ahead”
- ❖ Guessing end-user needs
- ❖ Etc.…

Why Personal Discipline Matters

- Being realistic about predicting the future
- Areas where discipline matters
  - Refactoring
  - Prototyping
  - Optimization
  - Managing Complexity
- Endpoints
  - Watts' Humphrey's PSP
  - Kent Beck's Extreme Programming

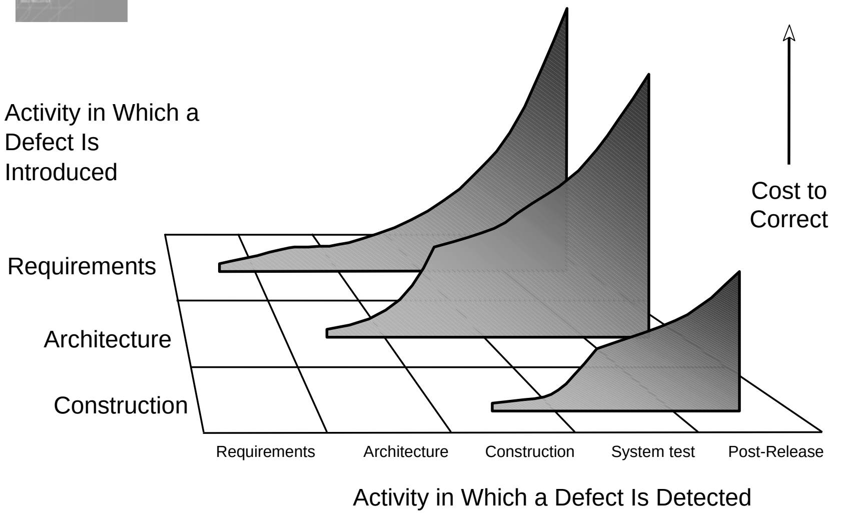
A Focus on Simplicity Works Better than a Focus on Complexity

Simplicity vs. Complexity

- ❖ Why do projects fail?
- ❖ Focus on write-time vs. read-time convenience
- ❖ YAGNI and “design ahead”

Defect-Cost Increase is Alive and Well

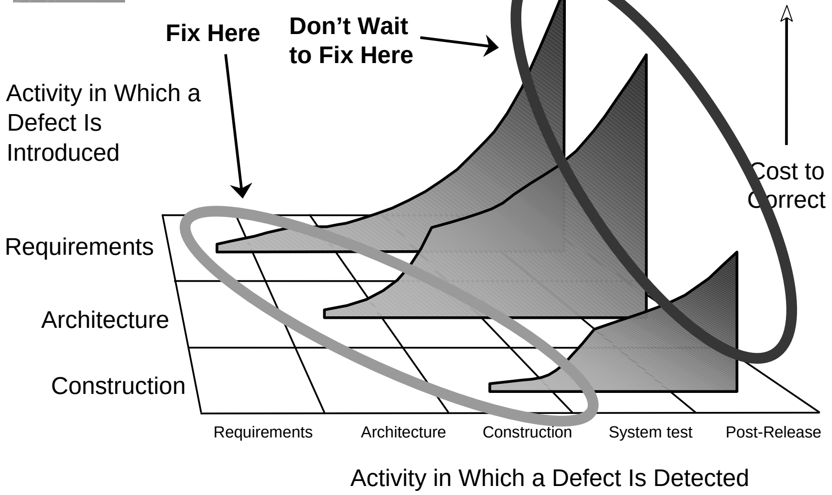
Defect Cost Increase



Decades of Research Support Defect-Cost Increase

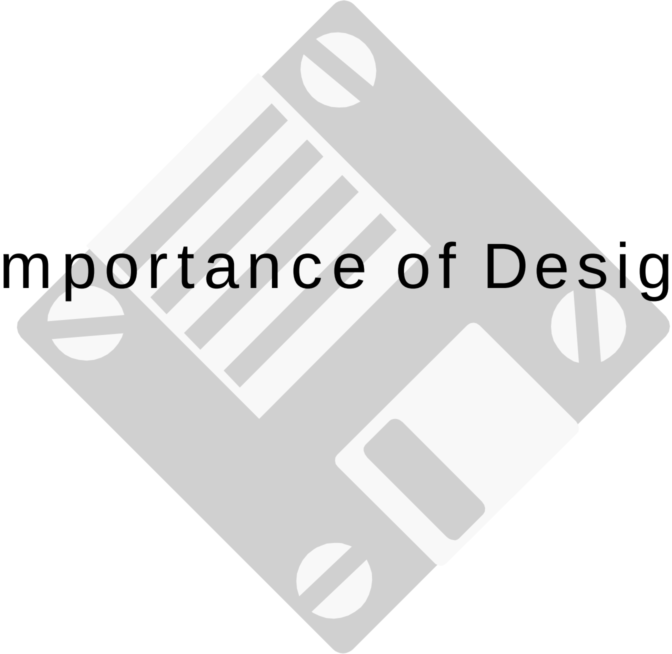
- ❖ Fagan, Michael E. 1976. “Design and Code Inspections to Reduce Errors in Program Development.” *IBM Systems Journal* 15, no. 3: 182–211.
- ❖ Humphrey, Watts S., Terry R. Snyder, and Ronald R. Willis. 1991. “Software Process Improvement at Hughes Aircraft.” *IEEE Software* 8, no. 4 (July): 11–23.
- ❖ Leffingwell, Dean, 1997. “Calculating the Return on Investment from More Effective Requirements Management,” *American Programmer*, 10(4):13-16.
- ❖ Willis, Ron R., et al, 1998. “Hughes Aircraft’s Widespread Deployment of a Continuously Improving Software Process,” Software Engineering Institute/Carnegie Mellon University, CMU/SEI-98-TR-006, May 1998.
- ❖ Grady, Robert B. 1999. “An Economic Release Decision Model: Insights into Software Project Management.” In *Proceedings of the Applications of Software Measurement Conference*, 227-239. Orange Park, FL: Software Quality Engineering.
- ❖ Shull, et al, 2002. “What We Have Learned About Fighting Defects,” *Proceedings, Metrics 2002*. IEEE; pp. 249-258.
- ❖ Boehm, Barry and Richard Turner, 2004. *Balancing Agility and Discipline: A Guide for the Perplexed*, Boston, Mass.: Addison Wesley, 2004.

Living With Defect-Cost Increase



Fix Here

Don't Wait to Fix Here



Importance of Design

Design Advice—What has Changed in 10 Years?

- In 1993, design pundits wanted to dot every *i* and cross every *t* before writing any code
- In 2004, design pundits say BDUF! YAGNI!
- There are lots of valid points on the "no design" "all design" continuum
- The only 2 points guaranteed to be wrong are the two being advocated!

General Point: Extremes are Usually Not Productive

- All design up front vs. no design up front
- Entirely sequential vs. entirely improvised
- All discipline vs. all art
- Document everything vs. document nothing
- Formal methods vs. hacking

Your Location on the Technology Wave Affects Your Construction Practices

Effect of Technology Waves on Construction

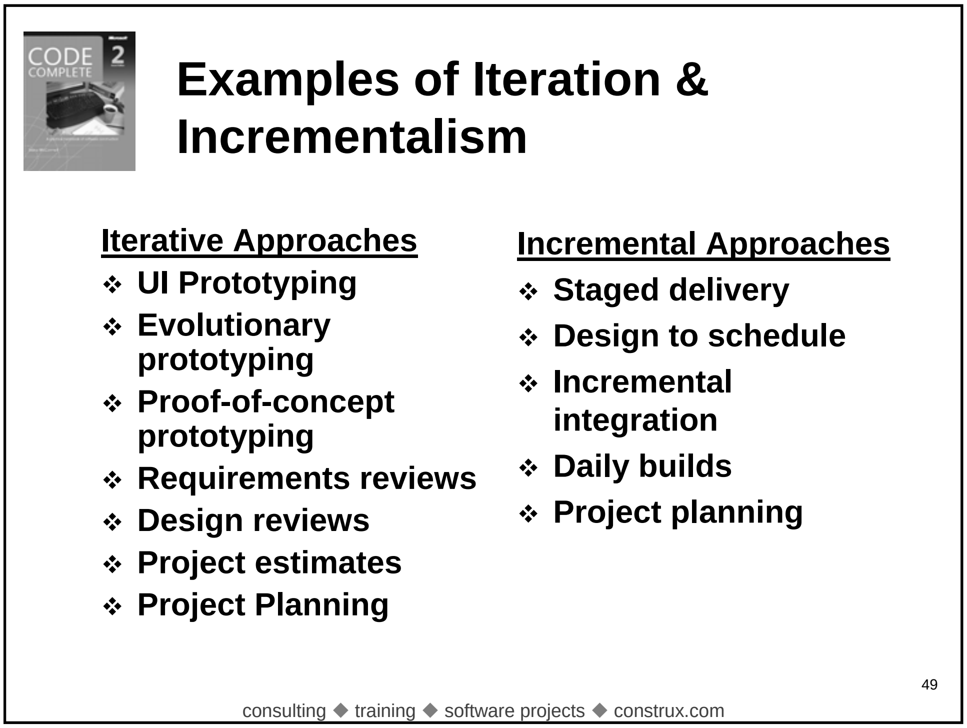
- ❖ Definition of “technology wave”
- ❖ Early wave characteristics
- ❖ Late wave characteristics
- ❖ Construction is affected by technology—more than I thought (duh!) but still at the “principles” level

Incremental Approaches Work Best

CC1’s View of Incrementalism

“Evolution during development is an issue that hasn’t received much attention in its own right. With the rise of code-centered approaches such as prototyping and evolutionary delivery, it’s likely to receive an increasing amount of attention.”

“The word “incremental” has never achieved the designer status of “structured” or “object-oriented,” so no one has ever written a book on “incremental software engineering.” That’s too bad because the collection of techniques in such a book would be exceptionally potent.”



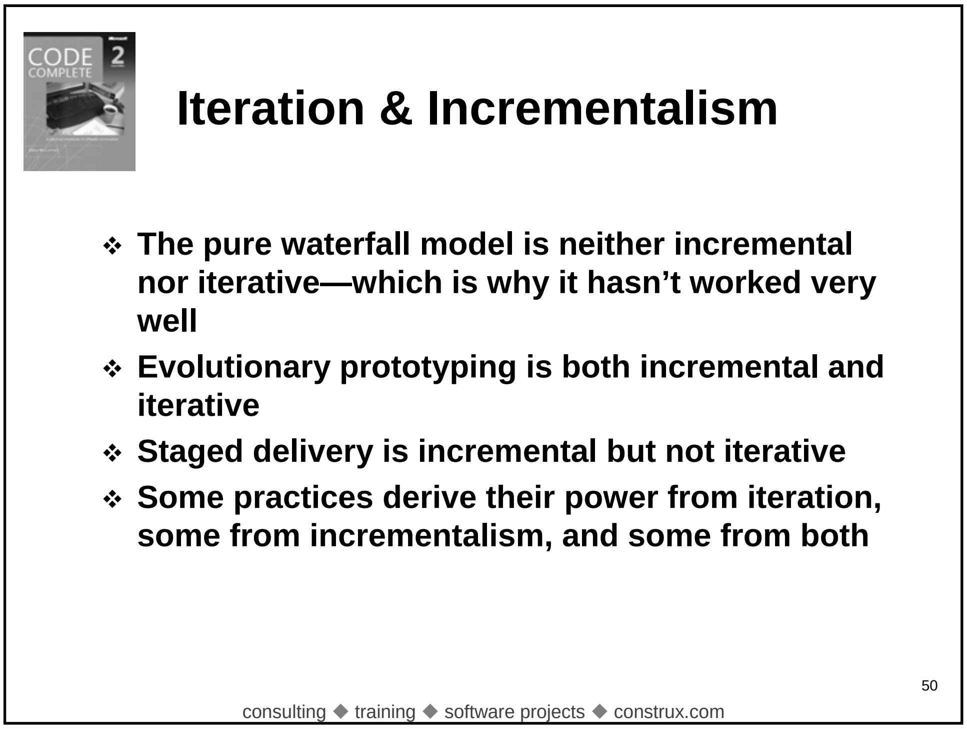
Examples of Iteration & Incrementalism

Iterative Approaches

- UI Prototyping
- Evolutionary prototyping
- Proof-of-concept prototyping
- Requirements reviews
- Design reviews
- Project estimates
- Project Planning

Incremental Approaches

- Staged delivery
- Design to schedule
- Incremental integration
- Daily builds
- Project planning



Iteration & Incrementalism

- The pure waterfall model is neither incremental nor iterative—which is why it hasn’t worked very well
- Evolutionary prototyping is both incremental and iterative
- Staged delivery is incremental but not iterative
- Some practices derive their power from iteration, some from incrementalism, and some from both

The Toolbox Metaphor Continues to be Illuminating

Toolbox Metaphor

- Toolbox explains there's no one right tool for every job
- What's in the Software Engineering Toolbox?
  - Best practices
  - Lifecycle models
  - Individuals' skills
  - Reusable materials (templates, checklists, standards, guides, patterns, samples, references)

Summary

Software’s Essential Tensions

- Software’s essential tensions have remained unchanged for years:
  - Rigid plans vs. Improvisation
  - Planning vs. Fortune Telling
  - Creativity vs. Structure
  - Discipline vs. Flexibility
  - Optimizing vs. Satisficing
  - Quantitative vs. Qualitative
  - Writeability vs. Readability
  - Process vs. Product
  - Theory vs. Practice
- Balance wavers, but basic tensions are constants
- In the end, these tensions are what keeps software development interesting!

Construx

Delivering Software Project Success

- ❖ **Software Projects**
- ❖ **Coaching & Consulting**
- ❖ **Training**

- ❖ ***info@construx.com***
- ❖ ***www.construx.com***

Major Updates to CC2

- ❖ **All programming examples (~500) updated to modern languages (Java, VB, C++)**
- ❖ **New chapters on Design, Classes, Defensive Programming, Collaborative Construction, Refactoring**
- ❖ **OO integrated throughout**
- ❖ **Web integrated throughout**
- ❖ **Numerous complementary resources on companion website cc2e.com**
- ❖ **Further Reading updated throughout**