Software Craftsmanship for New Developers

Software Guild
June 27, 2019

Who is Chad Green?

Director of Software Development at ScholarRx

<u>Community Involvement</u>
Code PaLOUsa Conference Chair
Louisville .NET Meetup Organizer
Louisville Tech Leaders Meetup Organizer
Louisville Tech Ladies Co-Organizer

<u>Contact Information</u>
chadgreen@chadgreen.com
chadgreen.com
@ChadGreen
ChadwickEGreen

What is Software Craftsmanship

Software Craftsmanship for New Developers

What Software Craftsmanship is not

- Beautiful code
- Test-Driven Development
- Self-selected group of people
- Specific technologies or methodologies
- Certifications
- Religion

What is Software Craftsmanship

- Software developers have had hard time defining themselves:
    - Historically practitioners of well-defined statistical analysis and mathematical rigor of a scientific approach with computational theory
    - Changed to an engineering approach with connotations of precision, predictability, measurement, risk mitigation, and professionalism

Craft, Trade, Engineering, Science, or Art

- Craft/Trade – Profession that requires particular skills and knowledge of skilled work
- Engineering – Creative application of science, mathematical methods, and empirical evidence to the innovation, design, construction, operation, and maintenance of structures, machines, materials, devices, systems, processes, and organizations
- Science – Systematic enterprise that builds and organizes knowledge in the form of testable explanations and predictions about the universe
- Art – Diverse range of human activities in creating visual, auditory, or performing artifacts, expressing the author’s imaginative, conceptual idea, or technical skill, intended to be appreciated for their beauty or emotional power

What is Software Craftsmanship

Agile Manifesto question some these assumptions

Individuals and interactions over processes and tools

Software Craftsmanship is about **professionalism** in software development.

Software Craftsmanship History

- 1992 – Jack W. Reeves publishes “What Is Software Design?” essay
- 1997 – Andrew Hunt and David Thomas publish *The Pragmatic Programmer*
- 2001 – Pete McBreen publishes *Software Craftsmanship*
- 2002 – Software Apprenticeship Summit
- 2006 – 8th Light Founded
- 2008 – Bob Martin proposes fifth value for the Agile Manifesto: Craftsmanship over Crap
- 2008 – Bob Martin publishes *Clean Code: A Handbook of Agile Software Craftsmanship*
- 2008 – Software Craftsmanship Summit
- 2009 – Manifesto for Software Craftsmanship
- 2011 – Bob Martin publishes *The Clean Coder: A Code of Conduct for Professional Programmers*

Manifesto for Software Craftsmanship

Not only working software,
but also **well-crafted software**

Not only responding to change,
but also **steadily adding value**

Not only individuals and interactions,
but also **a community of professionals**

Not only customer collaboration,
but also **productive partnerships**

Manifesto for Software Craftsmanship

Not only working software,
but also **well-crafted software**

Not only responding to change,
but also **steadily adding value**

Not only individuals and interactions,
but also **a community of professionals**

Not only customer collaboration,
but also **productive partnerships**

Manifesto for Software Craftsmanship

Not only working software,
but also **well-crafted software**

Not only responding to change,
but also **steadily adding value**

Not only individuals and interactions,
but also **a community of professionals**

Not only customer collaboration,
but also **productive partnerships**

Try and leave this world a little better than you found it, and when you turn comes to die you can die happy in feeling that at any rate you have not wasted your time but have done your best.

Robert Stephenson Smyth Bader-Powell, founder of The Scout Association

Manifesto for Software Craftsmanship

Not only working software,
but also **well-crafted software**

Not only responding to change,
but also **steadily adding value**

Not only individuals and interactions,
but also **a community of professionals**

Not only customer collaboration,
but also **productive partnerships**

Manifesto for Software Craftsmanship

Not only working software,
but also **well-crafted software**

Not only responding to change,
but also **steadily adding value**

Not only individuals and interactions,
but also **a community of professionals**

Not only customer collaboration,
but also **productive partnerships**

Technical Debt

Software Craftsmanship for Non-Developers

What is Technical Debt

- Reflects the implied cost of additional rework caused by choosing an easy solution now instead of using a better approach that would take longer

- Technical debt can be compared to monetary debt – If not repaid, it can accumulate interest, making it hard to implement changes later on

Example of Technical Debt

- Start writing an application and there is no need for user roles – everyone can do everything
- Requirement comes in for a permission for a specific requirement
- Some time later another things requires the differentiation of users, and then another and another
- The company has the opportunity to add five customers in a week – but really need another permission change in a couple of days

Common Causes of Technical Debt

- Insufficient up-front definition
- Business pressures
- Lack of process or understanding
- Tightly-coupled components
- Lack of a test suite
- Lack of documentation
- Lack of collaboration
- Parallel development
- Delayed refactoring
- Lack of alignment to standards
- Lack of knowledge
- Lack of ownership
- Poor technological leadership
- Last minute specification changes

SOLID Principles

Software Craftsmanship for Non-Developers

S.O.L.I.D.

- First five object-oriented design principles
  - **S** – Single-responsibility principle
  - **O** – Open-closed principle
  - **L** – Liskov substitution principle
  - **I** – Interface segregation principle
  - **D** – Dependency Inversion Principle

Single Responsibility Principle (SRP)

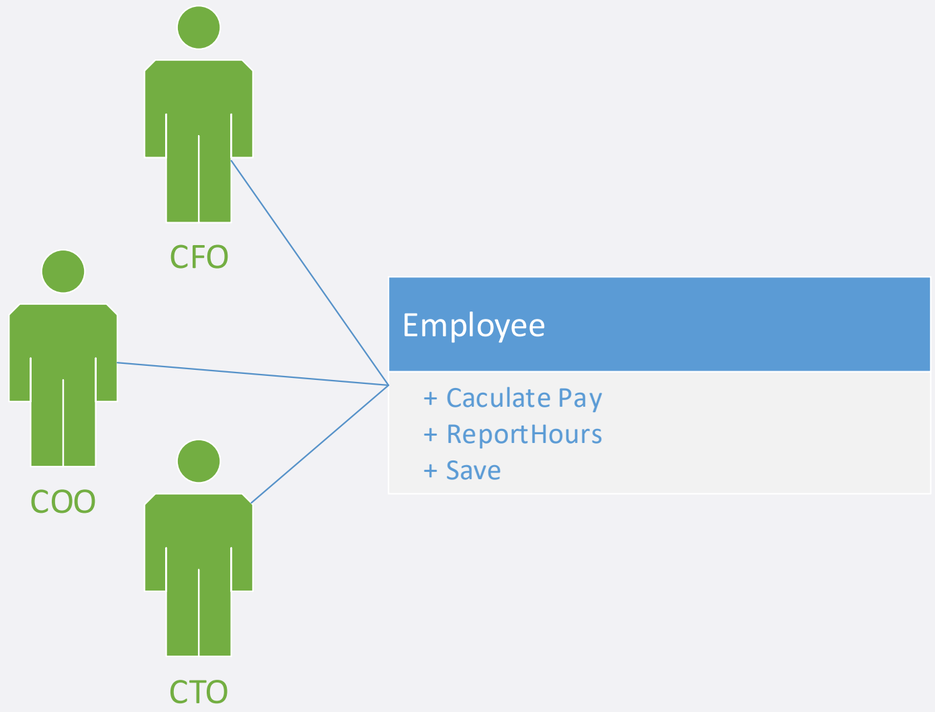
- ~~A module should have one, and only one, reason to change~~
- A module should be responsible to one, and only one, actor

Single Responsibility Principle (SRP)

- A module should be responsible to one, and only one, actor

Class violates the SRB because the three methods are responsible to different actors

- The `CalculatePay` method is specified by the accounting department, which reports to the CFO
- The `ReportHours` method is specified and used by the human resources department, which reports to the COO
- The `Save` method is specified b the database administrators, who report to the CTO

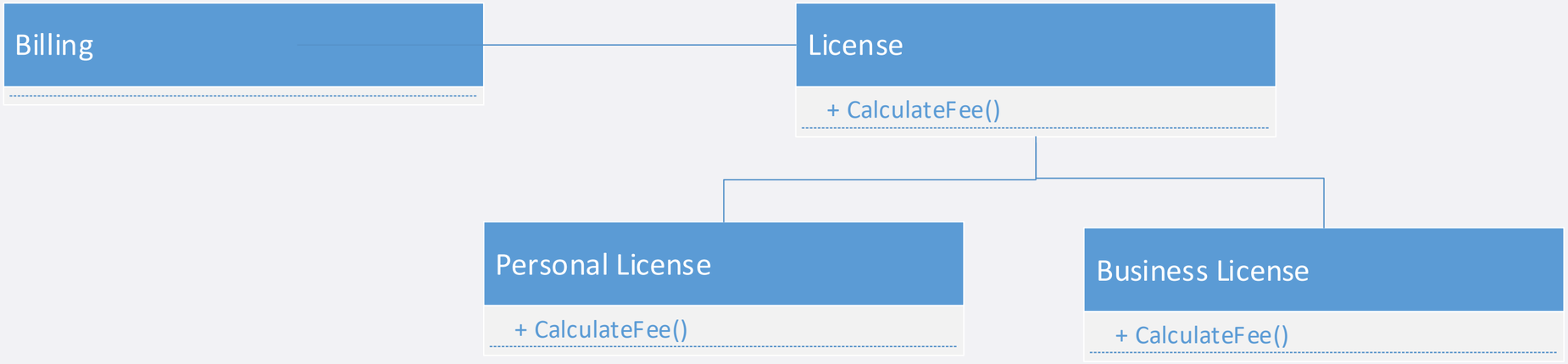


Open-Closed Principle (OCP)

- A software artifact should be open for extension but closed for modification

Liskov Substitution Principle (LSP)

- Let **q(x)** be a property provable about objects of **x** of type **T**. Then **q(y)** should be provable for objects y of type **y** where **S** is a subtype of **T**
- Every subclass/derived class should be substitutable for their base/parent class



Interface Segregation Principle (ISP)

- A client should never be forced to implement an interface that it does not use or clients should not be forced to depend on methods they do not use

Dependency Inversion Principle (DIP)

- Entities must depend on abstractions not on concretions. It states that the high level module must not depend on the low level module, but they should depend on abstractions.

Other Key Principles

Software Craftsmanship for Non-Developers

DRY – Don’t Repeat Yourself

- Every piece of knowledge must have a single, unambiguous, authoritative representation within a system

- Alternative is to have the same thing expressed in two or more place. If you change one, you have to remember to change the others.

- It isn’t a question of whether you will remember: it’s a question of when you will forget

If you write it once, think about encapsulating it. If you write it twice, you have to encapsulate it. If you write it three times, programming isn't for you.

Phil Japikse, Microsoft MVP, ASPInsider, MCSD, MCDBA, PSM II, PSD, CSM, Consultant, Coach, Author, Trainer

KISS – Keep it Simple Stupid

- The simplest explanation tends to be the right one

YAGNI – You Aren’t Going to Need It

- Always implement things when you actually need them, never when you just foresee that you need them

- Principle behind XP practice of “do the simplest thing that could possibly work”

Key Practices

Software Craftsmanship for Non-Developers

TDD – Test Driven Development

- Software development process that relies on the repetition of very short development cycle: requirements are turned into very specific test cases, then the software is improved to pass the new tests, only

- Three Laws of TDD
  1. You are not allowed to write any production code until you have first written a failing unit test.
  2. You are not allowed to write more of a unit test than is sufficient to fail – and not compiling is failing
  3. You are not allowed to write more production code that is sufficient to pass the currently failing unit test

Pair Programming

- Technique in which two programmers work together at one workstation
  - The driver writes code while the observer reviews each line of code as it is typed

Practicing – Coding Katas

- Practice, Practice, Practice
- Practice on *how* to solve the problem
- Katas – simple coding exercises
  - codingdojo.org/kata
  - codekata.com
  - codewars.com

Code Smells

Software Craftsmanship for Non-Developers

Code Smells - Comments

- **Inappropriate Information**

Code Smells - Comments

- Inappropriate Information
- **Obsolete Comment**

Code Smells - Comments

- Inappropriate Information
- Obsolete Comment
- **Redundant Comment**

```
i++ // increment i
```

Code Smells - Comments

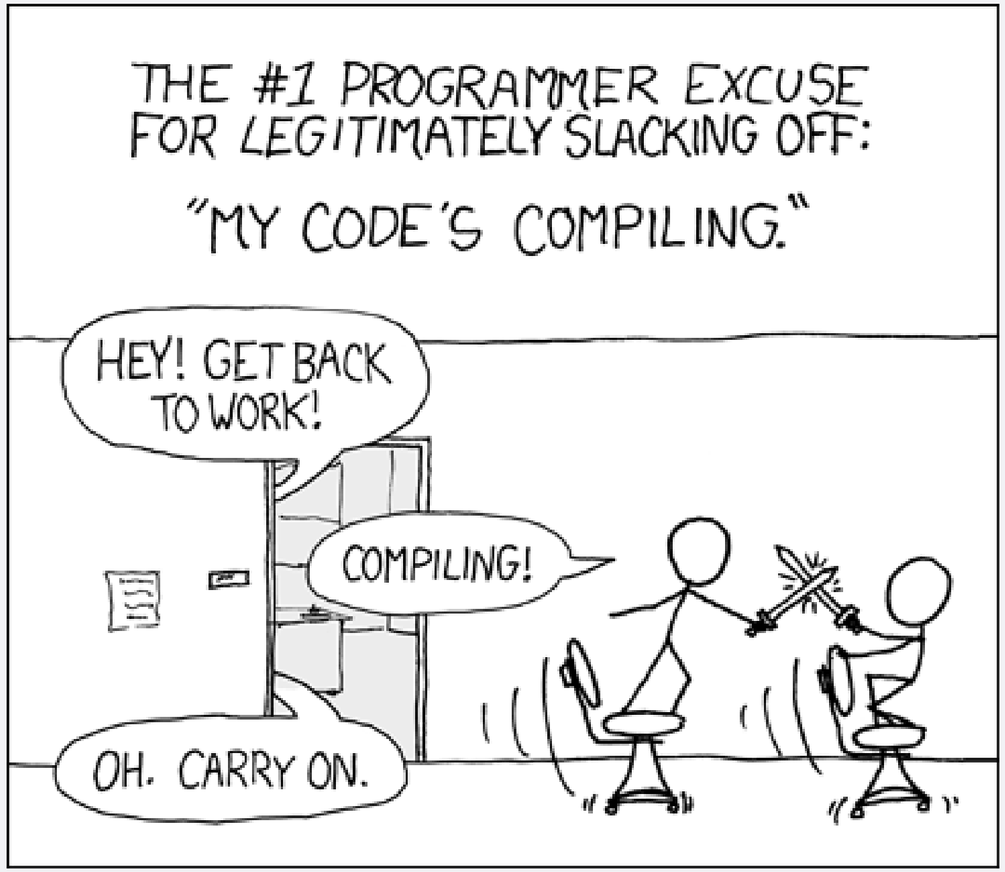
- Inappropriate Information
- Obsolete Comment
- Redundant Comment
- **Poorly Written Comment**

Code Smells - Comments

- Inappropriate Information
- Obsolete Comment
- Redundant Comment
- Poorly Written Comment
- **Commented-Out-Code**

Code Smells - Environment

- **Build Requires More Than One Step**



Code Smells - Environment

- Build Requires More Than One Step
- **Tests Require More Than One Step**

Code Smells - Functions

- **Dead Function**

Code Smells - General

- **Obvious Behavior is Unimplemented**

Code Smells - General

- Obvious Behavior is Unimplemented
- **Incorrect Behavior at the Boundaries**

Code Smells - General

- Obvious Behavior is Unimplemented
- Incorrect Behavior at the Boundaries
- **Overridden Safeties**

Code Smells - General

- Obvious Behavior is Unimplemented
- Incorrect Behavior at the Boundaries
- Overridden Safeties
- **Duplication (DRY – Don’t Repeat Yourself)**

Code Smells - General

- Obvious Behavior is Unimplemented
- Incorrect Behavior at the Boundaries
- Overridden Safeties
- Duplication
- **Dead Code**

Code Smells - General

- Obvious Behavior is Unimplemented
- Incorrect Behavior at the Boundaries
- Overridden Safeties
- Duplication
- Dead Code
- **Inconsistency**

Code Smells - General

- Obvious Behavior is Unimplemented
- Incorrect Behavior at the Boundaries
- Overridden Safeties
- Duplication
- Dead Code
- Inconsistency
- **Clutter**

Code Smells - General

- Obvious Behavior is Unimplemented
- Incorrect Behavior at the Boundaries
- Overridden Safeties
- Duplication
- Base Classes Depending on their Derivatives
- Dead Code
- Inconsistency
- Clutter
- **Misplaced Responsibility**

Code Smells - General

- Obvious Behavior is Unimplemented
- Incorrect Behavior at the Boundaries
- Overridden Safeties
- Duplication
- Base Classes Depending on their Derivatives
- Dead Code
- Inconsistency
- Clutter
- Misplaced Responsibility
- **Function Names Should Say What They Do**

```
DateTime newDate = date.add(5)
DateTime newDate = date.AddDays(5)
```

Code Smells - General

- Obvious Behavior is Unimplemented
- Incorrect Behavior at the Boundaries
- Overridden Safeties
- Duplication
- Base Classes Depending on their Derivatives
- Dead Code
- Inconsistency
- Clutter
- Misplaced Responsibility
- Function Names Should Say What They Do
- **Follow Standard Conventions**

Code Smells - General

- **Replace Magic Numbers with Named Constants**

3.141592653589793

3.141592753589793

Code Smells - General

- Replace Magic Numbers with Named Constants
- **Functions Should Do One Thing**

Code Smells - Names

- **Chose Descriptive Names**

Code Smells – Names: Choose Descriptive Names

```
CREATE PROCEDURE dbo.Act_Mobile_Cond_Workout_Activities_Log_View
  @ID INT = 0,
  @Workout_ID INT = 0
AS
BEGIN

  SELECT al.ID,
         al.cond_workout_ID,
         al.activity,
         al.mins,
         al.calburn,
         ae.Category,
         ai.ID AS intensity_id,
         ai.Intensity
    FROM HIE_Mobile_Cond_Workout_Activities_Log al
   INNER JOIN HIE_Cond_Activities a ON a.ID = al.activity
    LEFT JOIN HIE_Cond_Intensity ai ON ai.ActivityID = a.ID AND ai.ID = al.intensity
   WHERE al.active = 1
     AND (al.cond_workout_ID = @Workout_ID OR al.ID = @ID)
   ORDER BY al.created_date

END
```

Code Smells - Names

- Chose Descriptive Names
- **Avoid Encodings**

intRepeat

repeatCount

Code Smells - Names

- Chose Descriptive Names
- Avoid Encodings
- **Names Should Describe Side-Effects**

```
public void GetFoo() { }

public void CreateAndGetFoo() { }
```

Code Smells - Tests

- **Insufficient Tests**

Code Smells - Tests

- Insufficient Tests
- **Use a Test Coverage Tool**

Code Smells - Tests

- Insufficient Tests
- Use a Test Coverage Tool
- **Don’t Skip Trivial Tests**

Code Smells - Tests

- Insufficient Tests
- Use a Test Coverage Tool
- Don’t Skip Trivial Tests
- **Test Boundary Conditions**

Code Smells - Tests

- Insufficient Tests
- Use a Test Coverage Tool
- Don’t Skip Trivial Tests
- Test Boundary Conditions
- **Exhaustively Test Near Bugs**

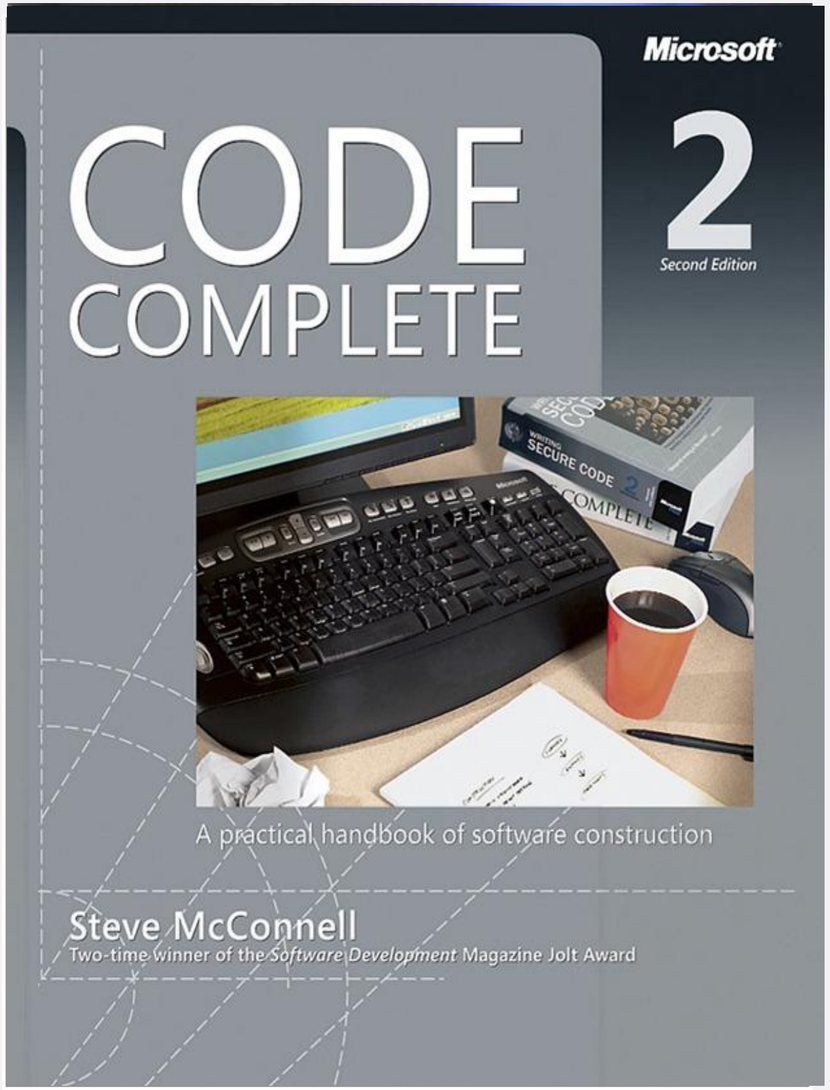
Code Smells - Tests

- Insufficient Tests
- Use a Test Coverage Tool
- Don’t Skip Trivial Tests
- Test Boundary Conditions
- Exhaustively Test Near Bugs
- **Tests Should Be Fast**

Getting More

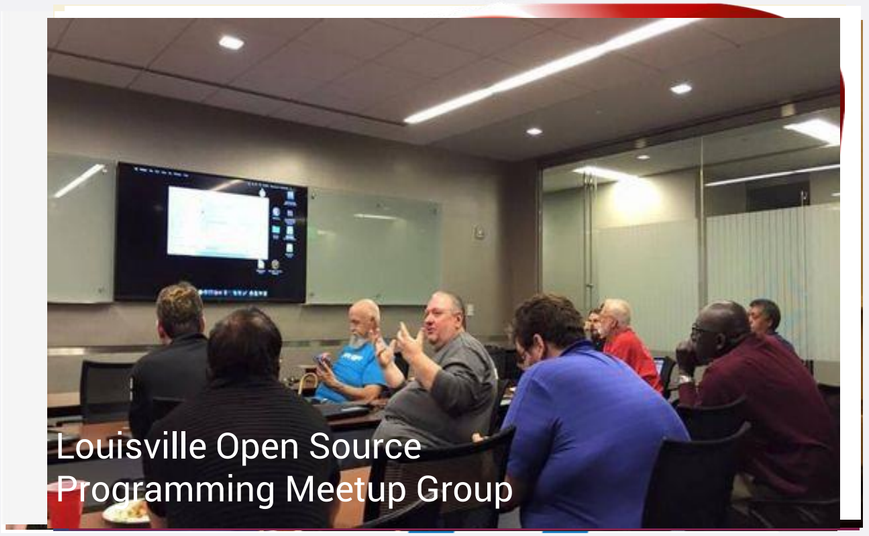
Software Craftsmanship for Non-Developers

Getting More – Books



Getting More – Podcasts

Getting More – Meetups



Getting More – Conferences

Software Craftsmanship is about **professionalism** in software development.

Questions

chadgreen@chadgreen.com
chadgreen.com
@ChadGreen
ChadwickEGreen

Ask Me Anything

thank you.