Gradually Improving Software Design

In-house training with examples in C#

Background

Successful software projects take months to hit the market and then years to mature. As this process progresses, design principles get sacrificed one after the other, best practices get neglected, sources of goodwill dry up. Big ideas pop up, such as to rewrite entire software, but that rarely ends in success.

Gradually Improving Software Design is the series of lectures that are following one specific line of thought. Through real world examples, attendees will get acquainted with design and coding practices that will help them improve their code "here and now."

These lectures have been carefully crafted by a ***distinguished Pluralsight author*** and a programmer with more than 20 years of professional career developing business applications. All techniques have been tried and tested in professional software development during the span of many years.

Main Objective

Main takeaway from the Gradually Improving Software Design series of lectures is the set of techniques that can be applied to improve quality of code, both at the design and implementation level.

We are aware that any delays and slowdowns in development process incur costs and risks that will likely be unacceptable for the stakeholders. Therefore, we are paying close attention to ensure that any technique proposed will not incur delays in development. In most of the cases, we can proudly say that these techniques will in fact incur measurable speedup in development.

The following tables are showing content of all available lectures. You are free to mix and match lectures as per your team's needs.

Training Format

Trainings are organized either as interactive workshops, or as traditional lectures. Both approaches have their pros and cons, and it is up to the team to decide which form of learning suits them better. Most of the materials we offer support both presentation forms. Some materials are only available as workshops, while some others fit better in form of lectures. Please refer to the list of topics below for full information.

Examples are implemented as ASP.NET Core applications.

Certificate

We firmly believe that there is no correlation between possessing a certificate and being able to produce working code. Where such correlation appears, we find it accidental. Hence, this learning programme produces no certificate of completion.

Our major aim is to empower programmers to design correct and maintainable code.

Contact

For any additional information or if you wish to arrange presentations, please feel free to contact the author directly:

Zoran Horvat
CEO & Principal Consultant, Coding Helmet Consultancy sprl
Rue Abbe Cuypers 3
1040 Etterbeek, Belgium

http://codinghelmet.com
zh@codinghelmet.com
Twitter: @zoranh75
+381(63)8927415

Training – Principles of Object-Oriented Design

Lecture 1 – 8h	***Elements of Object-Oriented Design***		
Level	***Intermediate***		
Part 1		***Part 2***	
Introducing Emergent Design	0:40	Avoiding null references	1:40
Branching and Looping	1:00	Option/Maybe functional type	1:00
Map-reduce Principle	2:20	Value Objects	1:20
Total duration	***4:00***	***Total duration***	***4:00***
Lecture 2 – 8h	***Emergent Object-oriented Design***		
Level	***Intermediate/Advanced***		
Part 1 – Basic Principles		***Part 2 – Implementing Emergent Design***	
Iterative Top-down Design	1:30	Traditional Emergent Design (ED)	2:00
Unifying the Design	1:00	Introducing Clean ED	0:30
Developing the Deep Model	1:30	Advanced Principles of ED	1:30
Total duration	***4:00***	***Total duration***	***4:00***
Lecture 3 – 8h	***Coding Standards***		
Level	***Intermediate***		
Part 1		***Part 2***	
Naming Concepts	0:50	Functional Thinking in C#	1:00
Single Responsibility of Classes	1:00	Error Handling w/o Exceptions	1:20
Single Responsibility of Methods	0:40	Pattern Matching in C#	1:00
Interface Segregation	1:30	Tuples and Fast Prototyping	0:40
Total duration	***4:00***	***Total duration***	***4:00***

Training – Design Patterns

Lecture 4 – 8h	***Design Patterns***		
Level	***Intermediate***		
Part 1		***Part 2***	
Abstract Factory	1:00	Chain of Responsibility	0:40
Factory Method	0:40	Message Queue and messaging	1:40
Builder	1:00	Command	1:40
Composite	1:20		
Total duration	***4:00***	***Total duration***	***4:00***

Training – Unit Testing

Lecture 5 – 8h	***Unit Testing Techniques***		
Level	***Intermediate***		
Part 1		***Part 2***	
White-box testing	1:00	Testing generic types	1:00
Black-box testing	1:00	Enforcing Liskov principle	1:00
Testing abstract data types	1:20	Testing value-typed objects	0:40
Writing tests against interfaces	0:40	Testing complex domain logic	1:20
Total duration	***4:00***	***Total duration***	***4:00***
Lecture 6 – 8h	***Test-Driven Development***		
Level	***Intermediate***		
Part 1		***Part 2***	
Developing public interface	1:00	Designing complex classes	1:00
Simplifying implementation	0:40	Refactoring and tests	1:00
Inventing class dependencies	1:00	Integration tests in TDD	0:40
Designing abstract types	1:20	Designing non-testable algorithms	0:40
		Testing nonfunctional requirements	0:40
Total duration	***4:00***	***Total duration***	***4:00***

Workshop – Principles of Object-Oriented Design

Workshop 1 – 16h	***Elements of Object-Oriented Design***		
Level	***Intermediate***		
Part 1		***Part 2***	
Emergent Design	1:20	Avoiding null references	3:20
Branching and Looping	2:00	Option/Maybe functional type	2:00
Map-reduce Principle	4:40	Value Objects	2:40
Total duration	***8:00***	***Total duration***	***8:00***
Workshop 2 – 16h	***Emergent Object-oriented Design***		
Level	***Intermediate/Advanced***		
Part 1 – Basic Patterns		***Part 2 – Structuring Behavior***	
Iterative Top-down Design	3:00	Traditional Emergent Design (ED)	4:00
Unifying the Design	2:00	Introducing Clean ED	1:00
Developing the Deep Model	3:00	Advanced Principles of ED	3:00
Total duration	***8:00***	***Total duration***	***8:00***
Workshop 3 – 8h	***Refactoring to Patterns***		
Level	***Intermediate/Advanced***		
Part 1 – Basic Patterns		***Part 2 – Structuring Behavior***	
Refactoring to Strategy	1:30	Refactoring to Builder	1:00
Refactoring to Composite	1:30	Refactoring to Basic Rules	2:00
Refactoring to Visitor	1:00	Composing Rules	1:00
Total duration	***4:00***	***Total duration***	***4:00***