.NET Architecture and Design Principles

Duration: 5 Days **Course Code: GK3344**

Overview:

Applications that span more than one machine require a deliberate and radically different design approach. .NET Architecture and Design Principles presents key concepts in distributed systems. Learn to build systems that are scalable, reliable and secure. Discussions range from object-oriented programming to enterprise patterns, networking to Web Services, caching to distributed databases, and client/database applications to very large-scale web sites. You'll get answers to these questions: How do I build scalable and reliable systems?How can I use patterns to design extensible, reusable services?What's the best way to communicate between distributed layers?

Target Audience:

Those who want to design and build large scale systems.

Objectives:

- Think in terms of layers and tiers
- Use patterns in your code and across the enterprise
- Write secure code
- Use concurrency to build highly available systems
- Make distributed calls using Windows Communication Framework and queues
- Utilize asynchronous communication with message queues
- Horizontally scale every tier of your system
- Deploy software across distributed systems

Prerequisites:

Content:

Day 1

Architecture

As .NET architects, we transform high-level business requirements into a functional

system within budget, resource, and schedule constraints. We look at how non-trivial

systems inevitably become distributed systems and how distributed systems introduce a

slew of additional architectural problems: communication, reliability, flexibility, reusability

and scalability. We seek to understand and recognize these issues and to anticipate the

problems they may cause.

Design Patterns I ; II

We want to create each individual component of a distributed system to be adaptable

and maintainable. Design patterns are recipes for organizing code to achieve these

goals. Using C# more advanced language features, we can codify many design patterns

- in a simpler, more flexible way, in contrast to other popular OO languages.

Day 2

Serialization

Distributed systems work by having the disparate parts somehow communicate by

passing around data and objects. Here we take the preliminary step of understanding the

various serialization libraries in .NET. We then learn to serialize objects efficiently so

programming (i.e., COM) did for application development. We can quickly build

large distributed systems from reusable building blocks. To do so requires careful

planning and coordination to prevent the services from becoming too tightly coupled,

- which reduces our opportunities to reuse them.

Day 3

Web Services

When building systems, many architects rely on web services to implement SOA

principles. Because web services are based on open standards, companies can expose

systems, either internally or externally, and not have to worry so much about the

communication layer. We delve into the WS-* standards, as well as the REST style of

software architecture popular on many large Internet services.

Concurrency

With more CPU cores and cheaper machines available, the incentive to build multithreaded

applications is stronger than ever. However, it hasn't gotten easier to write

correct concurrent applications. We survey the technologies available for .NET:

threading, the thread pool library, and theTasks library. We conclude with a variety of

design patterns that can reduce the complexity of concurrency and, we hope, a

some extreme cases, techniques to perform transactions manually.

Security

Security is critically important for any distributed system and getting it wrong is not an

acceptable outcome. We begin with an overview of distributed security, including

encryption and hashing. Next we survey the technologies available, from code access

security to web services.

Hosting and Deployment

Once a component of a distributed system is built, we must push it out into a production

environment or to the customer. We look at the options available for hosting an

application either with Windows Services or ASP.NET. Next we look at a variety of ways

to deliver code to customers, including Windows Installers and ClickOnce deployment.

Finally, we look at tools available to help deploy software out to the web and application

- tier, and further into the cloud.

Day 5

Performance and Reliability

As important as application performance is, we must weigh it against overall system

performance. We explore techniques to measure and improve the overall performance of

more work gets done for every distributed invocation. We close by surveying other

serialization technologies used in very large systems.

WCF

Windows Communication Foundation is a convenient library for building enterprise-scale

services. It allows developers to use Service Oriented Architecture techniques on top of

any communication technology, from HTTP to shared memory. It integrates smoothly

with many of Microsoft's products, allowing companies to leverage their existing

infrastructure. We will discuss the overall architecture of WCF and introduce contract-first

design.

Service-Oriented Architecture

Service-Oriented Architecture (SOA) aims to do for distributed systems what componentbased

corresponding number of bugs.

Messaging

The core of large, reliable distributed systems is often asynchronous communication,

usually using message queues. If we want our distributed systems to effectively use

message queues, the overall architecture must be capable of handling disjoint messages

that may arrive out of order or not at all. The extra effort put into designing an

- asynchronous distributed system can pay off by increasing reliability and scalability.

Day 4

Transactions

Transactions in databases ensure that several actions either succeed or fail together.

Transactions become more complicated in an environment that includes different kinds of

services: message queues, web services, databases, file systems, etc. Since

transactions can be quite expensive, we explore techniques to reduce overhead and, for

a distributed system. Next, we examine ways to enforce the business requirement that a

system must not fail, despite the fact that every component within the system certainly

will fail at some point. In some cases, we must prepare a system to deliver partial service

when some components have failed.

Scalability

A scalable architecture can handle more load (hits, users, data) by simply adding more

resources (CPU, disks, databases). We discuss a variety of scalable architectures for

each tier of a distributed system. Our goal is to build systems that grow horizontally, i.e.,

add more cheap machines to the system, rather than vertically, which means buying

larger machines which are exponentially more expensive. We look at real-world

architectures that implement these techniques.

Further Information:

For More information, or to book your course, please call us on 0800/84.009

info@globalknowledge.be

www.globalknowledge.be