



www.clipcode.net

Clipcode Training

2019 Curriculum

***A series of intensive one-day
training courses for
senior knowledge professionals***

***"The mind is not a vessel to be filled,
but a fire to be kindled" - Plutarch***



Clipcode Limited, 6 Cabinteely Green, Cabinteely, Dublin 18, Ireland

Web: <http://www.clipcode.net> E-Mail: training@clipcode.com

Clipcode Training is pleased to announce its latest set of intensive one-day training courses aimed at senior knowledge professionals, in areas such as mathematics, technology and operations.

These courses are available for on-site presentation throughout Europe. Your team may request them "as is" or customized, combined and extended to suit your specific requirements (e.g. you may design your own course by selecting modules of interest from different courses, or requesting new modules).

Our courses are based on the most up to date versions of the mathematical theories, programming languages, operating systems, runtimes, development tools, database engines and international standards that they cover.

To arrange on-site training, please e-mail training@clipcode.com with a description of the type/location/duration of training you require and the number of attendees and their experience levels, together with your full contact details.

In addition to training, Clipcode also offers a wide range of professional services aimed at assisting software engineering teams use the above technologies. These services include Kickstarts, Workshops, People, Projects, Reviews, Helpdesk and Content. See www.clipcode.net for details.

Entire Document – (C) Copyright 2019 Clipcode Training – all rights reserved.

Mathematics Faculty <ul style="list-style-type: none">● Foundations School● Structure School● Space School● Change School● Applied School	Technology Faculty <ul style="list-style-type: none">● Code School● Compute School● Data School● Networking School● UX School	Operations Faculty <ul style="list-style-type: none">● Cloud School● Identity School● FM School● DevOps School● SysAdmin School
--	--	--

<Design Your Own Course>

<You decide what goes here>

This is our most popular course!

If you need on-site training for your team, and our existing courses match your specific needs – excellent! However, perhaps they partially but not completely match your requirements, or perhaps your team needs something more custom. In either case, then go ahead and design a training course to your own specifications. We would be willing to write and present it in your offices to your development team. To design your own course you identify the target audience & their existing skill sets, decide what new competencies they need and then select a coherent set of topics to be covered. You can treat each of our current courses as a menu of

modules & choose as needed from each. Also you can identify new topics. Using existing specialist material we have, and where necessary creating new material specifically for you and combining this with appropriate selections from our standard courses, we can come up with the exact course you need. We work on the basis that if one client requires a course for a specialist area now, it is quite likely that others will require it in future.

We consider the new material we create as part of our ongoing curriculum development initiative and there is no additional charge for you beyond our normal fees.

Contents of One-Day Training Course	
Target Audience <You decide what goes here>	<div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p style="color: blue; font-weight: bold;"><You decide what goes here></p> <p><You decide what goes here></p> </div> <div style="width: 45%;"> <p style="color: blue; font-weight: bold;"><You decide what goes here></p> <p><You decide what goes here></p> </div> </div>
Prerequisites <You decide what goes here>	<div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p style="color: blue; font-weight: bold;"><You decide what goes here></p> <p><You decide what goes here></p> </div> <div style="width: 45%;"> <p style="color: blue; font-weight: bold;"><You decide what goes here></p> <p><You decide what goes here></p> </div> </div>

Mathematics Faculty

Foundations School

Core

- Fundamentals of Mathematical Foundations
- Cubical Type Theory
- Category Theory
- Mathematical Logic And Structural Proof Theory
- Automata Theory
- Description Logic

Structure School

- Graph Theory
- Advanced Data Structures
- STL using C++

Space School

- <in planning, will cover geometry, topology, etc.>

Change School

- <in planning, will cover calculus>

Applied School

- <in planning, will cover cryptography, etc. >

Fundamentals Of Mathematical Foundations

Terminology, Tour, Induction, Set Theory, Lambda Calculus, Complexity, Monad, Number Theory

Like ships passing nearby on a foggy night, each oblivious to the presence of the other, up to recently most mathematicians and software engineers ignore the work of the other. This course is about removing the fog, and letting developers see what can be achieved in production environments using really good ideas from modern mathematics. More and more ideas from mathematics are beginning to seep into the world of programming. Many software innovations – from deep learning to 3D graphics to modern programming type systems - are based on modern mathematics. All developers have studied some mathematics at college,

so this course builds on that. It should be considered a refresher, with an emphasis on practical application, to bring everyone up to speed with the basics and be ready to explore more advanced topics.

We pay particular attention to how mathematical ideas are presented. For example, saying “a monad is just a monoid in the category of endofunctors” utterly confuses developers, whereas we prefer “monads are programmable semicolons” (just used to insert custom code between each statement) is much clearer, yet equally accurate.

Contents of One-Day Training Course	
<p>Target Audience This course is aimed at modern developers who need a better grasp of how areas of mathematics can be practically applied to programming.</p> <p>Prerequisites It is expected attendees will have completed some mathematics training as part of their college education.</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <p>Review Of Fundamentals The language of mathematics (it is really not all Greek!) A mathematical object has certain properties and can be used in operations Mathematical structures are mathematical objects themselves that contains some arrangement of mathematical objects (often a set or similar, + something extra)</p> <p>Terminology Mathematical object (much broader use of ‘object’ term compared to programming) Symbols and (mathematical) variables Mathematical statement Proposition, expression, formula “a <i>proposition</i> is a statement susceptible to proof, whereas a <i>theorem</i> is such a statement that has been proven.” (HoTT)</p> <p>Branches of Mathematicsand their uses in software development Quantity/Arithmetic; Change/Calculus; Structure/Algebra; Space/Geometry Need to describe mathematical universes ..</p> <p>Mathematical Whirlwind Tour Quick tour of all of mathematics So many aspects to it – where do we start? We like “The Map Of Mathematics” Which new/unfamiliar parts we should use</p> <p>Mathematical Induction Definition by induction, in steps Base step (e.g. 0) is the starting point Induction step: builds on base (n) Example of deductive reasoning</p> </div> <div style="width: 48%;"> <p>Set Theory In the past, set theory was considered the most suitable approach to the foundations of all of mathematics More recent approaches (e.g. type theory, category theory) are better However, set theory is still an important area and worth studying Deductive system based on first order logic Law of the excluded middle (LEM) Axiom of choice, ...</p> <p>Complexity Theory Can a computational problem be solved? If so, how long will it take? Worst case scenario Big O notation</p> <p>Monads Isolating change in a non-changing world (think of carefully managed assembly lines - some steps make changes, others do not)</p> <p>Lambda Calculus The three terms (and how they work) – variables, abstraction and application Untyped vs. simply-typed Extensions (exceptions, recursion, ..) Currying (higher order functions) How Lambda Calculus is used in modern programming languages/type systems</p> <p>Number Theory Kinds of numbers (naturals, integers, reals, complex numbers) Broadening the scope (e.g. algebra) Specialist topics – e.g. Dedekind cut</p> </div> </div>

Cubical Type Theory

Theory of Truth, Types, Equality, Identity, Universes, Univalence, HITs, Kan Filling, Cubical Type Theory

The related [trinity](#) of type theory, mathematical logic/proof theory and category theory forms a modern foundation for all of mathematics and computation.

A type is a precise mathematical specification of behavior. An element of a type satisfies its specification. Both a type and its elements are programs, subject to evaluation. Elements evaluate down to values and types evaluate down to canonical types [Martin-Löf] or type values [Harper] (same idea). Application developers experienced with mainstream programming languages will already understand elements (objects) are subject

to evaluation, but will find types being so to be new and significantly more expressive. Type theory supports the idea of indexed families of types-the index being a value (also known as dependent types). This will also be a new idea to most developers. By introducing these ideas and lots more, modern type theory goes far beyond what we see as type systems for regular programming languages. This course focuses on the most promising of the latest type theories, known as cubical type theory- a computation-friendly approach to new ideas such as univalence & higher inductive types. The redtt open source project is a good implementation.

Contents of One-Day Training Course	
<p>Target Audience This course is aimed at modern developers who are specifically interested in modern type theory and how it can be practically used in computation</p> <p>Prerequisites Good foundational mathematical education along with some programming experience, as we include exploring type theory from a computational viewpoint. Attendees can select which programming language they wish to use, as all concepts will be developed from first principles.</p>	<div style="text-align: center;">A New Foundation</div> <p>Type theory Category theory Mathematical logic/proof theory How they relate to each other (Shulman) Constructivism</p> <div style="text-align: center;">Type Theory Overview</div> <p>Think of modern type theory as a highly expressive programming language useful for mathematics (+everything based on it) A theory of truth – based on Martin-Löf's ground-breaking paper: “Constructive mathematics and computer programming”</p> <div style="text-align: center;">Basic Concepts</div> <p>Type Element Evaluation Judgment Witness Intuitionistic Type Theory</p> <div style="text-align: center;">Simple Types</div> <p>Defining a type Defining an element Look at bool and nat(natural number)</p> <div style="text-align: center;">Common Types</div> <p>Type Function Sum Product Top / Bottom</p> <div style="text-align: center;">Dependent Types</div> <p>The idea behind dependent types Creating an indexed families of types</p> <div style="text-align: center;">Additional Concepts</div> <p>The meaning explanation Functionality (in a type theory sense) The hypothetical Recursion</p> <div style="text-align: center;">Equality and Identity</div> <p>Introduction to equality in type theory Variations of equality type (exact, ..) Identifications</p> <div style="text-align: center;">Higher Inductive Types (HITs)</div> <p>HITs as generalization of inductive types Uses in various forms of construction Diagonals</p> <div style="text-align: center;">Univalence</div> <p>Initially considered in terms of Homotopy Type Theory (HoTT) That uses an axiom – not desirable from a computational viewpoint – why? So other approaches explored ...</p> <div style="text-align: center;">Cubical Type Theory - Intro</div> <p>This is the cutting edge (2019) type theory Simple intro Uses points, lines, planes, cube, n-cube The idea of paths Transporting</p> <div style="text-align: center;">Cubical Type Theory - Details</div> <p>Kan filling Higher dimensionality Cartesian variant</p> <div style="text-align: center;">Redtt</div> <p>Redtt is an open source implementation of cubical type theory – let’s explore how it works</p>

Category Theory

Category, Category Object, Maps And Composition, Functor, Natural Transformation, Adjoint

Category theory is a foundational area of mathematics that examines in a uniform manner mathematical structures and their mappings. In category theory, a category is a mathematical universe. A category is populated by category objects and there are mappings (also called morphisms) between these objects.

It is best to think of the goal of category theory is to describe an abstract multiverse, containing one or more universes, with mappings within each universe, and between them. From this surprisingly simple core of constructs, a rich description of much of modern mathematics can accurately be built.

A category is itself a mathematical object. There can be mappings between categories, known as functors, and even mappings between such functors, known as natural transformations. Categories are abstract representations of concepts from other areas of mathematics.

This course helps mathematicians and software developers gain an appreciation of what is category theory, both basic concepts and more advanced capabilities, and to see how it can be practically applied in real-world situations.

Contents of One-Day Training Course	
<p>Target Audience This course is aimed at mathematicians and software engineers interested in learning about category theory</p> <p>Prerequisites Good foundational mathematical education along with some programming experience, as we include exploring category theory from a computational viewpoint. Attendees can select which programming language they wish to use, as all concepts will be developed from first principles.</p>	<p style="text-align: center;">Category Theory Overview</p> <p>Why the interest? What is a category? Practical applications Examining structure-preserving mappings between objects</p> <p style="text-align: center;">Basic Ideas</p> <p>Introduction to the basic constructs A category object is a mathematical object (to begin with, think of it as a set of elements) - so is not the same as an object in typical OO programming Think of a mapping (morphism) between category objects as a relation Maps can be composed – so if we have map f and map h, we compose them as h o f (read as “h follows g”) Such compositions form paths</p> <p style="text-align: center;">Mathematical Definition</p> <p>A category is defined as a collection of category objects, with maps (with domain and codomain) between them Each object has an identity map Two important laws: * Identity Law governs identity map use * Associative Law: (h o g) o f = h o (g o f)</p> <p style="text-align: center;">Functors</p> <p>Functors are mappings between categories Structure-preserving Endofunctors are mappings from a category object to itself -”endo” (as in “endoscope”) means “looks inside oneself Left/right adjoint functors (as in adjoining)</p>
	<p style="text-align: center;">Exploring Sample Categories</p> <p>Set, Ring, Monoidal, Group, 2-category, custom categories</p> <p style="text-align: center;">Advanced Category Theory</p> <p>Natural transformations A deeper look at a variety of morphisms Limits and colimits Universal property Initial and terminal object Presheaf</p> <p style="text-align: center;">∞-groupoid</p> <p>“Sets in the next dimension are groupoids” Groupoid builds on group in group theory A groupoid is a category where each morphism is an isomorphism ∞-groupoid generalizes groupoids k-morphisms and equivalences</p> <p style="text-align: center;">Cartesian Closed Categories</p> <p>Corresponds to lambda calculus Mapping to and from the lambda calculus “A CCC is a category that has an exponent and a product, and is closed over both. The product is an abstract version of cartesian set product; the exponent is an abstraction of the idea of a function, with an “eval” arrow that does function evaluation.” [link]</p> <p style="text-align: center;">Triumvirate</p> <p>Role of category theory as one member of the triumvirate that includes type theory and mathematical logic “Roughly speaking, a category may be thought of as a type theory shorn of its syntax” [link]</p>

Mathematical Logic & Structural Proof Theory

Logic Fundamentals, First Order/Higher Order Logic, Natural Deduction, Props as Types, Sequent Calculus

Mathematical logic permeates all of mathematics and programming.

Building on simple propositional logic, a host of richer logics can be constructed to target different needs. For example, First Order Logic is the deductive system used by set theory and is also the basis for Description Logic. Higher Order Logics have richer predicates.

We need to look at what we can represent in logic and what valid claims we can make about such. We also are interested in building our own logics.

One area of logic that is of particular interest is proof theory. Many believe the future of programming will be significantly influenced by automated verification of correctness of code - but what does that mean and how can it be achieved? There are already [open source projects](#) pointing the way to much more extensive use of formal verification.

There are a number of sub-fields in proof theory but we focus in on structural proof theory, which has some very interesting features. We are also curious as to what impact treating a proof as a mathematical object has.

Contents of One-Day Training Course	
<p>Target Audience This course is aimed at mathematicians and modern developers who need a better grasp of how mathematical logic and proof theory can be used in practice</p> <p>Prerequisites Good foundational mathematical education along with some programming experience, as we include exploring logic from a computational viewpoint. Attendees can select which programming language they wish to use, as all concepts will be developed from first principles.</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <p style="text-align: center;">Logic Fundamentals</p> <p>Review of logic as branch of mathematics</p> <p style="text-align: center;">Propositional Logic</p> <p>Conjunction, disjunction, negation, conditional, truth tables</p> <p style="text-align: center;">Predicate (First Order) Logic</p> <p>\forall means “for all” \exists means “there exists” Quantifiers, more symbols, ..</p> <p style="text-align: center;">Higher Order Logic</p> <p>Predicates themselves having parameters What variables range over (sets of sets)</p> <p style="text-align: center;">Additional Logics</p> <p>Modal Logic - modal terms and what impact they might have Temporal Logic – time-based, more ..</p> <p style="text-align: center;">Model Theory</p> <p>Mathematical models An interpretation gives meaning to symbols in a formal language When is an interpretation a model Interpretation function / Domain</p> <p style="text-align: center;">Structural Rules</p> <p>Weakening Contraction Exchange Associativity</p> <p style="text-align: center;">Linear Logic</p> <p>Resources (use once) vs. truth (constant) Substructural logic Writing deadlock-free code Specifications for communication Session types</p> </div> <div style="width: 48%;"> <p style="text-align: center;">Formal Verification</p> <p>Mathematically proving that code works in all circumstances will always be more desirable than unit testing it for known scenarios</p> <p style="text-align: center;">Overview of Proof Theory</p> <p>What is proof theory? Many branches – structural proof theory, provability logic, proof mining, automated theorem proving, ... Structural proof theory includes natural deduction/sequent calculus/hilbert A proof as a mathematical object- can be manipulated and reasoned about like any other mathematical object</p> <p style="text-align: center;">Natural Deduction</p> <p>Judgment, evidence and witnesses Depending on what kind of logic that interests us, differing judgments needed Introduction rule Elimination rule</p> <p style="text-align: center;">Proposition As Types</p> <p>Relationship to Lambda Calculus Propositions as types Proofs as programs Normalization as evaluation of programs Nirvana: code is (provable) logic is code</p> <p style="text-align: center;">Sequent Calculus</p> <p>A sequent consists of propositions to the left (ANDed), a turnstile and propositions to the right (OR) Cut elimination Importance for linear logic</p> </div> </div>

Automata Theory

Finite State Machines, State, Inputs, Outputs, DFA, NFA, Turing Machines, Uses

An automaton (plural: automata) is a logical model of a machine that, based on input events, transitions from state to state. To describe an automaton we need to identify its states, the set of acceptable inputs and the expected outputs, and describe how transitions work. There are a number of optional additional features to constructing automata and these can add a range of extra capabilities.

Automata theory is used throughout mathematics and programming. It is sometimes so natural that users are often unaware of its presence. We see it as a substrate feature of lexical analysis / parsing in compilation, as an alternative to the tableau algorithm in description logic, as finite state machines in any server product, as the construct used by generated networking comms layer from linear logic/session types & neural networks in AI.

The abstract machine represented by an automaton can be subjected to mathematical reasoning and certain important characteristics can be reliably proven.

Having a clear understanding of automata theory helps everyone on a team have a richer appreciation of how automata can be beneficial to a product's architecture.

Contents of One-Day Training Course	
<p>Target Audience This course is aimed mathematicians and developers who wish to become familiar with the theoretical and practical usage of automata theory.</p> <p>Prerequisites Attendees need a good foundation in mathematics and programming.</p>	<p>Overview Of Automata Theory Practical uses of automata Overview of automata theory Deterministic vs. non-deterministic How different automata vary</p> <p>Types of Automata In increasing order of complexity: * Finite state machine * Pushdown automata * Linear bounded automata * Turing machine What more complex automata brings</p> <p>What is Needed to Build States Inputs – what drives transitions Outputs – result of transitions Transitions</p> <p>States And Transitions Identifying states Optionally - identifying initial / final states May be more than one Transition function</p> <p>Deterministic Automaton A given sequence of inputs will result in a given set of state transitions A set of states A set of inputs the next state function the final predicate</p> <p>Non-Deterministic Automaton Impact of non-determinism Transition relation Converting a NFA to a DFA</p> <p>Specialist Automata Topics Acceptance conditions Automata with an infinite number of states Cooperation between multiple automata Relationship to computational theory Asynchronicity</p> <p>Automata and Category Theory Representing automata as categories Morphisms for automata The Aut Category</p> <p>Use Of Automata In Compilers Modern lexical analysis and parsing are essentially the use of automata with some supporting code (to load source file, or to construct abstract syntax tree) Dotted items to walk through input stream Base items (shift/reduce) & non-base items</p> <p>A Stack For Automata Sometimes it can be interesting to allow states to have memory A stack with push and pop operations Use of such a stack for compilers Designing pushdown automata engines</p> <p>Complexity And Automata Automata with large numbers of states can have performance issues Specifically for these, need to consider variation of approaches (pruning, compression, combination, ..)</p> <p>Project Use of automata theory in a non-trivial project to shows its benefits in a practical setting</p>

Description Logic

Representation, Reasoning, Tableau Algorithm, Subsumption, DL Extensions, Querying, Ontologies

Description Logic is the mathematics of the knowledge graph.

Description Logic is the discipline of mathematics concerned with knowledge representation and reasoning. There are many ways to represent knowledge as data but where Description Logic excels is doing this in such a way that greatly facilitates reasoning.

DL builds on a number of areas of mathematics such as the decidable fragment of First Order Logic and model theory, and pays attention to complexity theory.

There are a variety of Description Logics (DLs), and we compare and contrast a number of these. More advanced DLs feature richer expressivity, but also come with increased complexity in reasoning (especially for large knowledge bases), so it is important to understand the tradeoff.

A good understanding of Description Logic is essential when working with graph stores, reasoners and semantic query languages. DL provides the mathematical formalism that underpins the semantic web, the W3C OWL Language and the reasoners that process it.

Contents of One-Day Training Course	
<p>Target Audience This course will be of keen interest to mathematicians and software developers who wish to understand Description Logic - the mathematical foundations behind the Semantic Web and W3C specifications such as OWL, RDF and SPARQL.</p> <p>Prerequisites Good foundational mathematical education along with some programming experience, as we include exploring Description Logic from a computational viewpoint.</p> <p>Attendees can select which programming language they wish to use in the labs, as all concepts will be developed from first principles.</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p>Description Logic Overview</p> <p>Relationship to first order logic</p> <p>Description based on concept, role and individual plus operators</p> <p>TBox and ABox</p> <p>Importance of inferencing</p> <p>Constructing a mathematical model</p> <p style="text-align: center;">DL Basics</p> <p>Defining a simple DL</p> <p>How to represent knowledge using it</p> <p>What services could we layer on top of it?</p> <p style="text-align: center;">Model Theory</p> <p>An interpretation is a mapping for concepts, roles etc. and is a model if certain conditions hold</p> <p>Relationship between DL & Model theory</p> <p style="text-align: center;">Knowledge Services</p> <p>Subsumption / Consistency</p> <p>Inferencing</p> <p>Satisfiability</p> <p>Querying</p> <p style="text-align: center;">Tableau Algorithm</p> <p>More expressive DL needs richer reasoner</p> <p>Outline of tableau algorithm</p> <p>Variants</p> <p>DeMorgan's theorem</p> <p style="text-align: center;">DL Extensions (extra letters)</p> <p>Nominals</p> <p>Cardinality restrictions</p> <p>Inverses</p> <p>Temporal</p> <p>Extra role constructors</p> <p>Concrete domains</p> </div> <div style="width: 45%;"> <p style="text-align: center;">Family of Logics</p> <p>SHOIN (OWL DL)</p> <p>SROIQ (full OWL2)</p> <p>SHIF (OWL-Lite)</p> <p>... lots more</p> <p style="text-align: center;">Resources</p> <p>Tooling</p> <p>Frameworks</p> <p>Servers</p> <p>Reasoners</p> <p style="text-align: center;">Advanced Topics</p> <p>Automata as a specialist alternative to the tableau algorithm</p> <p>Study of complexity on relation to DLs</p> <p>Scalability (very large knowledge bases)</p> <p>Fixpoints</p> <p style="text-align: center;">Use of DL in OWL</p> <p>The W3C semantic web defines the Web Ontology language (OWL) which is based on description logic</p> <p>The DL terms (individual, concept, role) map to OWL terms (individual, class, prop)</p> <p style="text-align: center;">Ontologies vs. SQL databases</p> <p>There are similarities & differences (TBox = schema, ABox = data)</p> <p>Open vs. closed world assumption</p> <p>Incomplete information</p> <p>Role that reasoning plays</p> <p>Unique name assumption</p> <p style="text-align: center;">Query Answering</p> <p>How it works (query processing)</p> <p>Covers given and inferred knowledge</p> <p>Re-writability</p> </div> </div>

Graph Theory

Representation, Storage Alternatives, Traversing, Searching, Operations, Graph Drawing, Project

A graph is one of the most versatile structures in mathematics with widespread research & practical uses. We see use of graph concepts in areas ranging from the knowledge graph, the social graph and in organization-specific graphs (e.g. Microsoft Graph). Devs are used to graph terms such as the “object graph” or the “call graph”. Graph databases are becoming very popular. Anywhere you see the work “network” in industry there is a mathematical graph lurking underneath. Because of its popularity, we even see use of the “graph” term where it shouldn’t be – e.g. GraphQL (which is neither graph-based, nor indeed, a query language).

This course covers all aspects of graph theory, from simple representation, to traversal and search, to transformations, to display. Starting with simple edges and vertices (with storage as either a matrix or a linked list [preferred for sparsely populated graphs]), we investigate how to build, transform and present graphs.

We explore how to efficiently handle large graphs with a keen interest in high performance. This course also covers graph drawing (a surprising complex topic in its own right). We conclude with a project to build a graph engine.

Contents of One-Day Training Course	
<p>Target Audience This course is aimed mathematicians and developers who wish to become familiar with the theoretical and practical usage of graphs in a variety of scenarios.</p> <p>Prerequisites Attendees need a good foundation in mathematics and programming, as this course will be covering graph-related ideas from both disciplines.</p> <p>Attendees need to be familiar with one programming language. Any will do, as in the hands-on labs they will be developing a graph library in that language from first principles.</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <p>Overview of Graph Theory Part of discrete mathematics No “top” or root Overview of graph concepts Vertices and edges</p> <p>Introduction to Graphs $G = (V, E)$ where V is a node set and E is an edge set Simple graph Multi-graph Common operations on graphs Basic terminology Directed (digraphs) vs. undirected graphs</p> <p>Traversing A Graph The visitor pattern Ensuring each node is visited Weighted graph (e.g. assign a “cost” to each edge, to influence selected path)</p> <p>Searching A Graph Breath-first search Depth-first search Dealing with cycles</p> <p>Graph Operations Graph composition: merge points to be based on identity of edges Subsumption Node selection Edge contraction</p> <p>Types of Graphs Connected graph Bipartite graph Complete graph Regular graph</p> </div> <div style="width: 48%;"> <p>Graph Properties Shortest path Minimum spanning trees Traveling salesman</p> <p>A Tree As A Kind Of Graph Every tree is a graph, but not reverse Binary search trees Subtrees</p> <p>Graph As Network Flow Networks appear throughout engineering, science, business and daily life – how to best represent as graphs Max flow / min cut theorem Interacting with networks as graphs</p> <p>Graph Data Structures Adjacency matrix vs. adjacency list How to optimize for large graphs Indexing Improving storage on hard disk</p> <p>Graph Databases Review of how graphs are handled by a graph database Querying possibilities Sample usage</p> <p>Graph Drawing How to automate placing graph constructs on a planar surface The crossing number How to improve automated layout</p> <p>Project Creating a custom graph engine to store and interact with large-scale graphs efficiently</p> </div> </div>

STL: The C++ Standard Template Library

Algorithms, Containers, Iterators, Functors, Adaptors, Allocators, Performance, Internals

STL is an amazing masterpiece of software engineering. In addition to learning about a rich object collection system, developers will profit from studying STL deeply as they will learn how to put together their own modern frameworks to comprehensively tackle their specific needs.

The goal of this course is to bring C++ developers up to speed with all aspects of STL programming. We start with a tour of what STL has to offer and how it builds on some of the latest ideas in modern C++. We explore all the technical constructs defined by STL.

Iterators are used to flexibly define sequences of elements and manage the navigation among elements, without exposing the internal arrangement of elements. Algorithms are the operations we wish to perform on sequence of elements (independently of how those elements are stored). Containers (both sequence and associative) are how storage of groups of elements is managed. Adaptors provide more specialist access to containers. Allocators are used for memory layout. We also examine the very interesting STL architecture and discover how some of its design ideas used internally may be applied in designing our own class libraries.

Contents of One-Day Training Course	
<p>Target Audience C++ developers who wish to learn about the power of STL and see how to use it aggressively in their own applications</p> <p>Prerequisites Good knowledge of the fundamentals of C++ programming, especially templates and memory management.</p>	<p>STL Overview Tour of STL capabilities Generic programming Visiting with iterators Generic containers Container-independent algorithms Application programming using STL Highly efficient and flexible solutions</p> <p>C++ Review Review of aspects of modern C++ (e.g. templates, memory) that STL leverages</p> <p>Iterators A specialist pointer to an element Category Element type & distance type Reverse/stream/insertion iterators</p> <p>Algorithms Operates on sequences Passing in sequences using begin-/end- Template-based functions to perform ops Review of available algorithms Functors as algorithm predicates</p> <p>Sequence Containers Containers are collections Ordered collection of elements deque, list, vector Random access vs. sequential access</p> <p>Associative Containers Elements and their associated keys map, set, bitset</p> <p>Adaptors Adapting a container to a specialist need e.g. look at stack/[priority]queue interface</p>
	<p>Allocators Use default allocators at first, expand later Various strategies for managing blocks of memory using <memory> header</p> <p>STL & Shared Libraries Issues when passing STL collections across DLL/.so shared library boundaries Need for using same binary layout</p> <p>Deploying STL Optimizing STL use in your own projects Container selection – different containers have differing performance capabilities and differing feature sets Algorithm selection – being aware of large range of algorithms available is important</p> <p>Custom Building custom: - containers - algorithms - iterators - adaptors - allocators</p> <p>Design Ideas Review of architecture of STL Incorporating ideas from STL into your own framework designs</p> <p>Internals STL is delivered as a set of header files Exploring how it is put together More specialist functionality</p> <p>Project Building a C++ project that uses STL extensively</p>

Technology Faculty / Code School

Modelset

- Domain Driven Design
- Semantic Models
- UML
- Design Patterns

Assembly

- x86-64 Assembly
- WebAssembly

IR

- .NET IL

Compiler

- Fundamentals Of Object Programming
- C
- C++ 17
- C#
- The Java 11 Language
- TypeScript 3.2
- Python 3.7
- The Go Language
- OCaml Functional Programming

Domain Driven Design

Ubiquitous Language, Entities, Aggregates, Events, Services, Repositories, Contexts, Strategic Design

Every enterprise application has a domain – the actual area where the application delivers business value to real users. The database, user interface framework, messaging infrastructure, etc. are just tangential to what the application is really about. In the past, engineering teams concentrated so much on SQL database tables and GUI form layout that the handling of the domain was swamped and lost in the mix.

central pillar of their project development strategy. Its substantial benefits become very clear on larger projects and on projects that evolve over many iterations. This course explores all the patterns that underlie domain driven design with the goal that at the end of it, attendees will be fluent in DDD and can move from being participants in, to contributors to future projects that incorporate an important domain model. What’s above (e.g. the UI) the domain model and what is below (e.g. database, messaging) may well change frequently over iterations, but the domain layer itself will have longevity, so it is extremely important to get it right.

In contrast, modern software engineering teams rightly place much greater focus on the domain layer of the project and incorporate domain driven design as a

Contents of One-Day Training Course	
<p>Target Audience Object-oriented developers, architects and product managers who wish to build robust domain models as the central core of their applications.</p> <p>Prerequisites Good all-round experience of software engineering and product development</p>	<p style="text-align: center;">Overview</p> <p>What is domain driven design? What is its role in the larger software development ecosystem</p> <p style="text-align: center;">Layering</p> <p>Interaction (or UI) layer Application (or command) layer Domain layer Infrastructure layer Think of the domain as the middle of a sandwich rather than a slice of a pyramid</p> <p style="text-align: center;">Ubiquitous Language</p> <p>“All singing from same hymn sheet” Identifying a common terminology and set of meanings that all stakeholders can use Language of the domain (so non-techies can easily understand it)</p> <p style="text-align: center;">Entities and Value Objects</p> <p>Important role of identity What do we need to identify (and how) How do we attach values to identities</p> <p style="text-align: center;">Aggregates</p> <p>Boundaries and associations between groupings of entities and value objects Controlled access</p> <p style="text-align: center;">Domain Events</p> <p>State changes What is happening inside the domain model and exposing this to outside</p> <p style="text-align: center;">Factories & Services</p> <p>Constructing and supplying entities Segregating specific responsibilities Integration with dependency injection</p> <p style="text-align: center;">Repositories</p> <p>Connecting to a database (e.g. ORM) with an aggregate access service Creating and calling queries Role of testing</p> <p style="text-align: center;">Bounded Context</p> <p>What is inside and outside the scope of a domain model Importance of boundaries & multiple models (good fences make good neighbors)</p> <p style="text-align: center;">Supple Design</p> <p>Intention revealing interfaces Side-effect free functions Assertions Conceptual contours On-going model evolution</p> <p style="text-align: center;">Strategic Design</p> <p>Core domain Segregated core Abstract core Dependencies – managing relationships between large project subsystems</p> <p style="text-align: center;">Large-Scale Projects</p> <p>Review of layering Published language and internals Extensibility Flexible architecture for longer lifecycles Handling large systems</p> <p style="text-align: center;">Project</p> <p>The combined use of many domain driven design ideas inside a larger project – including creating the domain model and its use from other layers</p>

Semantic Models

Domain / UI / Entity Data / Security / REST / Learning / Extensibility / Admin / Deployment Model

[[Detailed Intro](#)] Imagine a dev team has mastered core technologies and building on that solid foundation, now needs to design a next generation solution – how do they go about it? If you team is in this situation, this advanced course is what you need. Looking beyond the technologies, this course examines how to develop cutting-edge solutions that are of production quality, commercially competitive and feature all the “-abilities” your customers demand (scaleability, testability, manageability, usability, reliability, ...). The central idea is that software development revolves around a series of semantic models (domain, entity data, user interaction,

security, admin, deployment, etc.) and these are grounded in a “single truth” of the source code which ensure all models work together.

For a new solution, we need to examine what is required to design and build each of these models so that the integrated end result delivers upon the expectations. Modern developer technologies allow us to be highly creative and more productive in how we go about designing web solutions. However, using them all together is somewhat of a challenge and this course carefully explores how to proceed.

Contents of One-Day Training Course	
<p>Target Audience Advanced software architects and senior software engineers who wish to design cutting-edge solutions using the latest design ideas.</p> <p>Prerequisites It is essential that attendees have a good all-round experience of the technologies they wish to use.</p>	<p>Model Driven Development Q: Assuming we know the very latest technologies and APIs, how do we go about designing next generation solutions? A: Using a series of semantic models</p> <p>Domain Model Domain Driven Design Handling complexity Central role of the Domain Model Evans' patterns for Domain Driven Design – Ubiquitous Language, Entities, ValueObjs, Intention-Revealing Interfaces, Services, Repositories, Layered Arch, ...</p> <p>Entity Data Model How we bind objects and relational data Creating data models & abstractions Using entities in other parts of app</p> <p>User Interaction Model Thinking past the widgets and visual layouts, we concentrate on what the user is really trying to achieve Task-focused design Making available functionality as needed Fluid interaction; paradigm selection</p> <p>Security Model Deciding what needs to be protected and how; verifying security of solution How to explain your solution's security model to others (e.g. security auditors)</p> <p>REST Model Representational state transfer can be used to selectively expose the Domain Model and Entity Data Model to remote clients</p> <p>Deployment Model Get the bits deployed is important (on average, 50% of support calls relate to this) Treating deployment as first class feature</p> <p>Administration Model Different levels of users Administrative features</p> <p>Support Model What happens when a user has a problem How can software help with support Instrumentation for support</p> <p>Education Model Often an afterthought, the education model is how users/admins learn to use the app It is crucial to on-ramping of new users and the amount of (costly) support they need</p> <p>Test Model Unit tests, load tests, security tests, ... Dependency injections and mocking Managing the testing process</p> <p>Intelligence Model Extracting actionable results from various information stores and making it promptly/easily available</p> <p>Extensibility Model Exposing an API to enable third parties extend your solution – approaches</p> <p>Reference Architecture Exploring a reference architecture built on ideas covered in this course.</p> <p>Project Attendees will work in teams to develop slice of a suitable project (they choose)</p>

Unified Modeling Language

UML Views of a Project, Diagramming Notation, Use Cases, Classes, Relationships, Behaviors, States, OCL

UML is used by object-oriented designers to unambiguously specify an OO design, to discuss it with other designers and to communicate it to all stakeholders - developers, end-users and management.

We can think of UML as a graphical design language for object oriented software. It has become the graphical lingua franca of object design, supported by most design/development tools and understood by most software professionals. To discuss modern software among a team of stakeholders, we need to describe objects – their internal static information, how they

relate to each other, how they behave dynamically (both intra- and inter-object) and how they are delivered to end-users. Software systems are becoming increasingly complex and at the same time there is persistent pressure to decrease development costs and time scales, and improve quality.

UML is the key to succinctly describing correct software architecture, and this in turn is the cornerstone of successful projects. A very important point is such design needs to be sufficient without being excessive.

Contents of One-Day Training Course	
<p>Target Audience Experienced object-oriented architects and developers who need a detailed understanding of all aspects of UML.</p> <p>Prerequisites A good understanding of object-oriented principles and previous OO design & programming experience.</p>	<p>UML Introduction “The UML is a language for visualizing, specifying, constructing and documenting the artifacts of a software-intensive system” (Booch et al)</p> <p>UML Views There are five ways of looking at systems: User, Design, Process, Implementation and Deployment views UML diagramming notation Class name, operation and attribute, interface, component, package, note, state, state transition, event, action, etc.</p> <p>UML Diagram Types Use cases, class, object (instance), sequence, collaboration, statechart, activity, component, deployment</p> <p>Classes and Relationships The structural (static) aspects of a project Completely defining operations/attributes Generalization</p> <p>Advanced Classes and Relationships The four ways classes relate to others Aggregation / composition</p> <p>Interfaces/Components/Deployment Functionality as components Collaborations</p> <p>Dynamic Behavior The dynamic aspects of a project Interaction among a set of object instances</p> <p>Finite state machine within an object (statechart diagrams) General system activity (activity diagrams)</p> <p>UML For Database Design The Persistent tagged value Storing objects in a relational database Generating database schemas from UML</p> <p>Object Constraint Language OCL enables tighter specification by adding detailed constraints to elements within object models</p> <p>Round-Trip Engineering Converting between UML and languages such as C++, Java, C#</p> <p>UML and ... Multithreading Networking User Interface Frameworks</p> <p>UML and Agile Development Requirements, Analysis, Design, Implementation, Test, Delivery – all progressing in parallel</p> <p>Design: sufficient, not excessive How much/little design does an agile actually project need? How to go about creating it? UML as a form of sketching for devs</p> <p>Complete Example Walk through of creating a set of UML diagrams for the lifecycle design of a modern multithreaded HTTP web server</p>

Design Patterns

Pattern Template, GoF Patterns, Applying Patterns, Advanced Patterns, Anti-Patterns, Pattern Extraction

[Detailed examples] Design patterns capture successful design experience for later reuse by the original/other designers. They capture solutions that have evolved over time, in a concise and easily applied fashion. Typically they are not the “first attempt” at solving a problem – but rather the result of an iterative design process by experienced designers (who have benefited from hard-learned lessons of previous projects). Teams are under tremendous pressure to produce higher quality software at lower cost. One option is to ship the work to cheap offshore development partners. A better option is to use smaller, but much higher skilled teams - who will

compete by working more effectively to build the software. Such advanced teams will be trained in design patterns and hence can aggressively leverage them to retain a productivity and quality advantage over lesser skilled competition. As software projects increase in scale, cost and complexity, and involve more inter- and intra-company relationships, there is a need to adopt techniques such as design patterns, to ensure use of best practices in design issues. The goals of this course are to look at a range of design patterns, to examine how to apply them in your own projects and explore how to create your own pattern catalog.

Contents of One-Day Training Course	
<p>Target Audience This course targets senior software engineers and architects who need to leverage design patterns correctly in their own projects.</p> <p>Prerequisites Attendees require good OO knowledge and plenty of development experience.</p>	<p>Overview of Design Patterns Simple and elegant solutions Applying the concept of generics from programming to software architecture Catalog of design patterns</p> <p>Defining a Design Pattern Documentation template Important GoF fields (intent, motivation, applicability, structure, participants, collaborations, consequences) Additional fields by others</p> <p>GoF Creational Patterns Abstract Factory, Builder, Factory Method, Prototype, Singleton</p> <p>GoF Structural Patterns Adapter, Bridge, Composite, Decorator, Façade, Flyweight, Proxy</p> <p>GoF Behavioral Patterns Chain Of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor</p> <p>Applying Patterns Applying patterns in your projects Implementing patterns in code Evolution of project over time</p> <p>Design Patterns and ... eXtreme Programming Aspect Oriented Programming (AOP) Agile Development Documentation</p> <p>Server Platform Patterns Patterns to satisfy competing demands</p> <p>Pooling, tuning, managing, sharing, distributing, extending, scheduling</p> <p>Concurrency/Network Patterns Wrapper, Component configurator, Interceptor, Extension interface, Reactor, Proactor, Async completion token, Acceptor-connector, *-locking, Active object, Monitor object, Half-sync/halfasync, leader-follower</p> <p>Enterprise Integration Patterns Gregorgrams Message exchange, channels, headers</p> <p>Security Patterns How to correctly design security features into your software systems Secure channel, Session, Role, Checkpoint, Single access point, Full/limited views</p> <p>Anti-Patterns Anti-patterns “let you zero in on the development detonators, architectural tripwires and personality booby-traps that can spell doom for your project” (Brown)</p> <p>Pattern Extraction Finding patterns in your own projects Effort involved in extraction Optimizing and generalizing patterns</p> <p>Custom Pattern Catalog How development teams can build up a catalog of patterns for their own needs</p> <p>Project Using Patterns An advanced project whose architecture uses an assortment of design patterns</p>

x86-64 Assembly

Architecture, Opcodes, Primitives, Functions, Flow Control, OS Calls, C interop, Virt, SIMD, Project

“What Andy giveth, Bill taketh away”. As the processor folks manage to speed up the computer, the software folks invariably manage to slow it down with layer upon layer of software inserted between the application and the CPU. It is a major problem that many of today’s application developers have little or no appreciation of what is happening “under the hood” as their code mysteriously interacts through all these opaque layers with the CPU. The solution is for developers to learn assembly for the CPU their applications use. This intensive course provides a whirlwind tour of major features of 64-bit x86 assembly.

There are three real benefits to learning x86-64 assembly. Firstly, you can program the performance-critical sections of your app directly in it, and exploit rich features such as SIMD. Secondly, you can optimize code written in high-level languages (HLL). With more knowledge, you can make more informed decisions about structuring your HLL code. Thirdly, you can debug complex problems at the instruction set level. The x86 opcodes are what gets passed to the CPU to execute. In tough debugging scenarios the more you understand what is transpiring, the quicker you catch bugs.

Contents of One-Day Training Course	
<p>Target Audience This course target experienced software developers who wish to learn about x86-64 assembly, so that they can code in it directly and so that they understand how their high-level languages actually execute, to help with debugging and optimization.</p> <p>Prerequisites Good experience of software programming. No previous assembly programming experience required.</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <p>Assembly Features Common aspects of assembly Assembly as a language Opcodes CPU Instruction Set Architecture (ISA)</p> <p>Microprocessor Architecture Machine code, microcode & micro-ops The CPU’s role in a computer system The bus, devices, I/O, addressing</p> <p>The x86 Instruction Set What makes up the instruction set? Registers, cache, SIMD, float-point, etc. Traps and interrupts Application structure</p> <p>Tools Assembler, disassembler, profiler, debugger, dumper, other</p> <p>Integers And Floats Representing scalar and floating numbers Basic mathematical calculations</p> <p>Strings String operations Storing, loading and manipulating</p> <p>Conditionals Flow control opcodes Labels, jumps, branches</p> <p>Functions Calling functions Creating functions Parameters</p> <p>Executable Formats ELF (Linux) and PE/COFF (Windows) How assembly is placed inside executable</p> </div> <div style="width: 48%;"> <p>The OS and System Calls How system calls are exposed to assembly Calling the OS API Error handling</p> <p>C and Assembly Passing data from one to the other How C code is accessible from assembly How assembly is accessible from C Inline assembly</p> <p>Optimization Understanding the cost of each instruction Reducing the number of instructions Using more appropriate instructions</p> <p>Stream Computing Vector programming Single instruction, multiple data (SIMD) Opcodes for SIMD Highly efficient for certain workloads</p> <p>Virtualization Hypervisor Doing virtualization in software Modern on-chip virtualization Working with virtualization in assembly</p> <p>64-bit Lengthening the registers Address space changes</p> <p>Other Microprocessor Families Contrast x86 with competitors (e.g. PowerPC, ARM) Interesting futuristic architectures</p> <p>Project Developing a high-performance custom solution in x86-64 assembly</p> </div> </div>

WebAssembly

Embedders, Modules, WABT, Primitives, Functions, Linear Memory, Tables, Control Flow, JS API

All the major browser vendors - Google, Mozilla, Microsoft and Apple - have cooperated to define an agreed standard called WebAssembly that specifies how an executable looks like for embedders to execute.

WebAssembly allows you to run code written directly in assembly or in a high-level language (e.g. C/C++) compiled into assembly in a browser without plug-ins.

WebAssembly is a virtual Instruction Set Architecture that runs in an embedder. Currently the most popular embedder is the modern standard web browser and on the server the latest Node.js 11 supports it. Other more

specialist embedders include the OCaml based [spec interpreter](#) and the C++ based [WABT interpreter](#).

WebAssembly modules are binary, low-level (e.g. support 64-bit integers, unlike JavaScript), and are very fast both to load and run. Support for calling WebIDL-defined APIs (e.g. the DOM) will be added in future.

Web developers with requirements for very high performing applications will benefit from exploring WebAssembly as it the basis for considerable industry innovation and has wide cross-browser support.

Contents of One-Day Training Course	
<p>Target Audience Developers wishing to get the very best performance out of their web browser and web server code</p> <p>Prerequisites Knowledge of C/C++ /JavaScript/TypeScript with some background experience of working with assembly</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <p>WebAssembly Overview Part of the modern web platform Designed for the web (security, etc.) Tour of WebAssembly concepts</p> <p>WebAssembly Modules Binary files with .wasm suffix Can be written by hand (assembly programming) or compiled from C/C++ Overview of assembly syntax S-Expressions</p> <p>Embedder Role of embedders Embedding in web browsers Embedding on server (Node.js 11/v8 7.0) Spec and WABT interpreters Custom embedders in your applications</p> <p>WABT – WebAssembly Binary Toolkit Low-level CLI tools to work with wasm modules Assembler, disassembler, interpreter, linker, extractor etc.</p> <p>Primitive Types Just four!! - i32 / i64 / f32 / f64 Working with primitives Primitive operations</p> <p>Functions Defining and calling functions Function parameters and return value The call opcode The start function Importing/exporting functions</p> </div> <div style="width: 48%;"> <p>Linear Memory Handling strings in linear memory Memory imports and exports Data section</p> <p>Structured Control Flow Hierarchical targets Don't jump to specific address, rather move up levels in hierarchy block/loop/if/else/br_table opcode</p> <p>Tables Indirect function calls and security Tables and the call_indirect opcode Table imports and exports</p> <p>JavaScript API Think of a WebAssembly module as a low-level representation that is passed to web browser for internal/local compilation Calling from JS to wasm / wasm to JS</p> <p>Module Binary Format Binary format with well-defined layout Sections are either named or indexed Extensible – can add custom sections LEB128, opcodes, ..</p> <p>WebAssembly Threads Chrome 70 introduces wasm threads Shared state via shared array buffer</p> <p>Future Enhancements Future enhancements coming in areas such as SIMD, exceptions, garbage collection and synchronization</p> <p>Project Using all the major constructs of WebAssembly in a large project</p> </div> </div>

.NET Intermediate Language (IL)

Stack Engine, IL Fundamentals, Metadata, IL Syntax, Opcode Injection, Building Compilers, Reflection

Intermediate Language (IL) is .NET's low-level platform-independent representation of an executable. Many .NET developers are content to write high-level code in an IDE and then compile/run it, oblivious to IL. More advanced developers and those with specialist needs are more ambitious – they wish to program directly in IL, to browse and edit the IL generated for them by high-level language compilers, to auto-generate source code from other logical representations, to create compilers, and to really know “under the hood” how code runs when using high level languages (to help optimize performance, aid more precise debugging, etc.)

This course covers all aspects of IL, including the opcodes, metadata, assembly syntax, compilation/de-compilation tools, binary file format and .NET's reflection (which provides classes to browse existing assemblies and to emit assemblies directly). We also examine usage scenarios, such as building your own compiler and code generation tools.

Attending this course will allow you get a jumpstart on understanding all aspects of .NET IL, to produce a variety of code manipulation functionality and gain a much better appreciation of how .NET code executes.

Contents of One-Day Training Course	
<p>Target Audience This course will interest advanced .NET developers who wish to code directly in IL, or who need a richer understanding of how their higher-level code executes, or who need to create code generators and specialist developer tools.</p> <p>Prerequisites In-depth knowledge of C# and all round experience using the .NET CLR</p> <p>Experience of language design, compiler creation and low-level code manipulation useful</p>	<p style="text-align: center;">Review of CLR Issues</p> <p>Assemblies & modules, how code executes, security issues, type loader CLR architecture from IL viewpoint Stack-based execution engine</p> <p style="text-align: center;">IL Fundamentals</p> <p>Overall IL Model Verbose/compact IL JIT compiler Hello world in IL</p> <p style="text-align: center;">IL Tools</p> <p>Ilasm.exe, Ildasm.exe Ngen.exe, PEVerify.exe</p> <p style="text-align: center;">Introduction to Structure of IL</p> <p>PE/COFF headers and sections Metadata tables Manifest Managed code representations</p> <p style="text-align: center;">Metadata Fundamentals</p> <p>Set of tables with very detailed data about contained code; Table types and uses</p> <p style="text-align: center;">Advanced Metadata</p> <p>Important tables (ModuleDef, TypeDef, MethodDef, FieldDef, AssemblyRef, ModuleRef, ClassLayout, NestedClass</p> <p style="text-align: center;">Types, Fields and Methods</p> <p>The IL instruction set Use of IL language constructs How code from high-level .NET languages appears in IL</p> <p style="text-align: center;">Advanced Types</p> <p>Signatures, visibility, inheritance, ctors Primitive/native/managed types</p>
	<p style="text-align: center;">Other IL Features</p> <p>Unmanaged code Exception handling/events/delegates</p> <p style="text-align: center;">Programming with IL</p> <p>Writing more complex programs in IL Coding issues to be aware of Object interactions in IL</p> <p style="text-align: center;">Profiler API</p> <p>The unmanaged Profiler API allows you to add custom code that will be called when the CLR is about to JIT IL code You can change the IL on-the-fly</p> <p style="text-align: center;">Code Interactions</p> <p>Coverage of why and how one might wish to programmatically interact with code Overview of required code services</p> <p style="text-align: center;">Reflection</p> <p>System.Reflection namespace Dynamically loading & invoking types Browsing contents of assemblies</p> <p style="text-align: center;">Emitting</p> <p>System.Reflection.Emit.*-Builder classes Emitting persistent & transient assemblies</p> <p style="text-align: center;">.NET Native and IL</p> <p>.NET Native – concepts and toolchain Converting from IL to native code</p> <p style="text-align: center;">Building Custom IL Tools</p> <p>Coverage of why and how to programmatically interact with IL code Overview of required code services</p> <p style="text-align: center;">Project</p> <p>How to integrate IL modules in your own custom project</p>

Fundamentals Of Object Programming

Modern OOD, Design By Contract, Reusability, AOP, MDA, Design Patterns, Software Failure, Generics

Object oriented techniques has now been applied to a large number of projects with a varying degree of success. With a better understanding of where the real benefits lie, object technology is being continuously refined and improved accordingly. This training course brings software developers - who already have a working knowledge of the basics of OO - up to speed with modern object-oriented design and the evolving OO programming techniques and enables them to discover how to use objects successfully. It answers the question: "What is happening in the OO world, beyond the fundamentals such as inheritance, encapsulation

and polymorphism?" The goal of any OO project is to produce the best software for as few resources (time /money/devs) as possible. The software should evolve well in the future and adapt to changing needs. Parts of the software should be reusable on other projects. It should behave robustly when it encounters errors.

You will benefit from attending this course by gaining a clear understanding of the very latest object technology concepts, understanding its vocabulary and identifying how it fits into the broader picture of s/w engineering when describing how to approach software tasks.

Contents of One-Day Training Course	
<p>Target Audience Software engineers wishing to learn about the latest advances in object-oriented concepts</p> <p>Prerequisites Some previous OO software development and general application design experience.</p>	<p style="text-align: center;">Object Modeling</p> <p>Why do we need to model? What benefits does it produce? Object technology involves a series of simple concepts – why is it so difficult to really get right?</p> <p style="text-align: center;">Design By Contract</p> <p>Design By Contract concepts Preconditions/postconditions/invariants</p> <p style="text-align: center;">Object Discovery</p> <p>How do I find the appropriate objects Object discovery techniques Is this one object, two objects, or none?</p> <p style="text-align: center;">Modern OOD</p> <p>Getting the class and instance structure right. The importance of typing and ADT. Meyer’s Software Construction Principles: Linguistic Modular Units Self-documentation Uniform Access Open-Closed Single Choice</p> <p style="text-align: center;">Reusability</p> <p>How to reuse objects Design for reuse Managing reuse</p> <p style="text-align: center;">Aspect Oriented Programming</p> <p>What are aspects Reuse through the callstack Setting up the call environment with AOP</p> <p style="text-align: center;">MDA</p> <p>Concept of Model Driven Architecture How to apply MDA</p>
	<p style="text-align: center;">Design Patterns</p> <p>The essence of a good project is not in its codebase but in its design A good design is far more reusable than a good piece of source code Design patterns are a technique of capturing design experience for later reuse by the original and other designers</p> <p style="text-align: center;">Software Failure</p> <p>Identifying failure points Catching, reporting and reacting to failure Importance of handling failure within OO</p> <p style="text-align: center;">Generics</p> <p>Type-independent programming Generics in various languages</p> <p style="text-align: center;">Agile Modeling</p> <p>Problems with “heavy-weight” processes AM is a family of light-weight processes, suitable for demands of fluid projects Modeling the AM way</p> <p style="text-align: center;">eXtreme Programming (XP)</p> <p>XP Principles: pair programming, integrate & test every day, continuous feedback, get to running code ASAP, evolutionary design, managing communication</p> <p style="text-align: center;">Latest OO Ideas</p> <p>Orderly vs. Experimental Engineering AntiPatterns Internet time Designing for continuous change</p> <p style="text-align: center;">Case Study</p> <p>Detailed case study showing how to apply advanced OO concepts to a project</p>

C 18

Data Types, Functions, Conditionals, Pointers, Functions, I/O, Structures, Pre-processor, DLLs

C is available on virtually every form of computing hardware and for every OS. C libraries are callable from almost every high-level programming language (this is needed because the API to most OSes is in C). Applications seeking close-to-the-metal performance need to be written in C. Most kernel and device driver development is in C. The syntax of custom languages for GPUs (shader languages) and FPGAs are C-like.

Hence every developer should know C and know it well. What you learn on this intensive training course will be applicable on all these target platforms.

In this course all the main C programming concepts are covered. This includes flow control constructs, pointers, functions, the pre-processor and typedefs along with the importance of data-types, type safety and custom structures. Compiling, linking and debugging multi-file applications are covered with demo code that evolves from simple ten-line utilities to large-scale projects involving multiple developers (e.g. libraries & header files). As well as covering the syntax of the C language, we also cover how to use the important C standard library (e.g. I/O) and explain how to use C effectively in app development.

Contents of One-Day Training Course	
<p>Target Audience This training course is aimed at software engineers who need to quickly get up to speed developing in C.</p> <p>Prerequisites Attendees must have a programming background but no experience of C required.</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <p style="text-align: center;">Overview of C</p> <p>C evolution – K&R, ANSI, C99, C11, C18 Whirlwind tour of features of C Hardware issues (e.g. volatile)</p> <p style="text-align: center;">A Simple Program</p> <p>Freeform language Data types and variables Constants, functions, comments, layout</p> <p style="text-align: center;">Variables and Data Types</p> <p>Simple data types, naming and size Strings - char and wchar Security and buffer overflow issues Enumerators User defined structures and unions Static and global variables Visibility of global variables</p> <p style="text-align: center;">Intro to C Functions</p> <p>Splitting features into functions Small and self-contained chunks of code A multi-function C program</p> <p style="text-align: center;">Operators</p> <p>Arithmetic & relational operators '=' and '==' Increment and decrement operators Bitwise operators</p> <p style="text-align: center;">Flow of control</p> <p>if / if else / for loop switch/case while break and continue</p> <p style="text-align: center;">Non-Local Jumps</p> <p>Practical uses of non-local jumps setjmp and longjmp</p> </div> <div style="width: 48%;"> <p style="text-align: center;">Functions in Detail</p> <p>Function prototypes Variable arguments lists Functions in separate files (.c and .h files)</p> <p style="text-align: center;">Pointers and Arrays</p> <p>Handling arrays and array arithmetic Pointers to standard data & custom structs Function pointers (defining/setting/calling) void vs. void *, use of restricted modifier</p> <p style="text-align: center;">Pre-processor</p> <p>#include, #define, #ifdef and #endif Concatenation/stringizing/varadic macros</p> <p style="text-align: center;">Dynamic Memory</p> <p>malloc, free, realloc; costs of memory ops Examples of dynamic memory usage</p> <p style="text-align: center;">C Standard Library</p> <p>Standard I/O, file I/O Assert, math, conversions, time</p> <p style="text-align: center;">Compiling, Linking, Debugging</p> <p>Role of modules and compilation units Constructing apps and libraries from code in multiple files Building, linking and debugging</p> <p style="text-align: center;">C11 [major]/C18 [minor]</p> <p>Tour of new features of latest standard Anonymous structs New C threads and stdatomic header files</p> <p style="text-align: center;">Project</p> <p>Object-based C - let's build OO in C with inheritance (tree of anonymous structs), encapsulation (redefining structs), exceptions (using setjmp/longjmp), namespaces, RTTI, generics, etc.</p> </div> </div>

C++ 17

Classes, Inheritance, Visibility, Templates, RTTI, Exceptions, Namespaces, Class Hierarchy Design

This course provides developers with an intensive introduction to programming in C++, together with an overview of its powerful standard library. C++ is a general purpose programming language, highly suited for objected oriented and generic application, component and systems development. It is used for advanced, complex, full functionality projects. It is fast – and often selected where high performance coupled with a rich object-oriented language is needed.

Above and beyond the new syntax aspects of C++, this course also examines how to profit from a range of new

object programming techniques that are evolving, such as genericity and refactoring. Also explored is the optimum architecture for modern C++ class hierarchies, the latest C++ standards (C++11, C++14, C++17) and interaction with C code.

A core competency of all software engineers is to be having completely mastery of the programming language they use. If you are currently a programmer moving to a new C++ project, then you will greatly benefit from attending this course.

Contents of One-Day Training Course	
<p>Target Audience This training course is aimed at people who need to quickly get up to speed developing in C++.</p> <p>Prerequisites Attendees should be experienced software developers with a good knowledge of any high-level programming language.</p>	<p>Tour of C++ Features Primitives Flow control Enumerations Preprocessor Compilation</p> <p>Classes Class members and this ptr Visibility – public, protected, private Constructors and destructors static</p> <p>Inheritance Superclass and sub-classes Constructors and the inheritance hierarchy Impact of the virtual keyword</p> <p>Multiple Inheritance Benefits and problems Which function is executed The diamond and vtbl issues</p> <p>Templates Function templates Class templates Function specialization How to write code once and then use it with a variety of data types Debugging template code Designing with genericity in mind</p> <p>Lambda Expressions What problem are these trying to solve Unnamed functions Expression syntax Writing code that accepts lambda expressions</p>
	<p>Exceptions Benefits, costs and recommendations associated with C++ exception handling Families of exceptions Throw-try-catch syntax Declaring exceptions in header files</p> <p>Namespaces Avoiding naming conflicts The std and other namespaces Creating & using your own namespace</p> <p>RTTI Run Time Type Information Exploring objects + their types at runtime</p> <p>Refactoring Software grows and changes Sub-optimal solutions evolve Refactoring is a set of coding best practices which aim to fix & improve</p> <p>C Code Interaction How C++ code can call C code How C code can call C++ code extern C</p> <p>Design of C++ Class Libraries Classes are seldom needed on their own How groups of classes can cooperate Designing hierarchies Optimally placing data and functionality</p> <p>Advanced Language Features Temporaries/ Header file changes New keywords: mutable, typename, etc. Smart pointers: rationale & how to use Potential problems to avoid C++20: What's coming soon</p>

The Java 11 Language

OO, Classes, Inheritance, Interfaces, Generics, Annotations, Exceptions, Events, Modules, JNI

With a much improved standards update cadence (the [Java 11 LTS Language Specification](#) arrived in September, 2018, with [Java 12](#) expected in March, 2019), there is re-invigorated interest in advancing the Java language and ecosystem, by Oracle and external teams using and contributing to Java’s evolution.

The Java 11 Language is a powerful object-oriented language with modern features such as the Java Platform Module System, lambdas, generics and annotations and a comprehensive set of OO features such as classes, inheritance, interfaces and exceptions.

Some of the largest enterprise projects in the world are written in Java. These are multi-year projects that are vital corporate assets and need to continuously evolve well into the future. Java is an important language for many university courses. Many high throughput cloud platforms are written in Java. The main language for Android is Java. Plenty of cutting-edge open source projects are written in Java. In addition to existing projects, Java is also being regularly selected as the main language for all kinds of new projects. Hence now is an ideal time to learn Java well – starting with the language itself.

Contents of One-Day Training Course	
<p>Target Audience Software developers wishing to become Java developers, starting with learning the language itself.</p> <p>This is an ideal first course in Java.</p> <p>Prerequisites Programming experience with an OO language such as C++, TypeScript or C#, along with good exposure to object-oriented design.</p> <p>No previous experience of Java is needed, as this course covers the language from the fundamentals up.</p>	<p>Java - Language & Ecosystem Review of the ecosystem surrounding the Java language and how it is evolving Features - language, VM, bytecode, runtime services, framework, tooling What’s new in Java 11</p> <p>A Java Project Walkthrough Simple hello world project Main and command line args Primitive data types Code layout & basic syntax Classpath / JAR files</p> <p>Classes And Constructors Simple class definitions Static vs. instance members Access modifiers Variety of constructor layouts Nested classes</p> <p>Inheritance Inheritance trees Extending classes Overriding / hiding</p> <p>Generics Generic functions Different invocations Generic types (type variables) Shadowing Constraints</p> <p>Annotations We often wish to attach additional data to a class -without impacting the inheritance hierarchy Java annotations allow use of metadata</p> <p>Interfaces & Mixins Defining behaviors via interfaces Implementing multiple interfaces Uses for interfaces Default methods How to provision mixins using interfaces</p> <p>Exceptions Basic exception handling Creating an exception Throwing and re-throwing Catching an exception</p> <p>Event Handling java.lang.EventObject Observer and Observable Creating event listeners</p> <p>Lambdas Lambda expressions Quick and easy way to define a method Arguments and body Designing code using lambdas</p> <p>Java Platform Module System A named set of packages Module keyword Organizing modules</p> <p>JNI - Calling C Code JNI – Java Native Interface Calling out to C code from Java Passing parameters and accepting returning results between different languages</p> <p>Java Project Developing a Java project using a selection of language features showing how they can sensibly be used together</p>

C#

C# Fundamentals, .NET Fx Intro, Types, Classes, Inheritance, Attributes, Delegates, Generics

C# is the premier development language for the .NET platform. C# was designed from scratch with .NET in mind. Most of the internals of .NET Core and Visual Studio are written in it. It has been selected by the majority of application teams creating commercial .NET projects. C# builds on the rich common heritage of languages such as C++ and Java - but avoids their pitfalls and adds certain interesting new concepts, such as LINQ. There are some aspects of C# that developers will already know, there are some they have experienced similar but slightly different syntax in other languages, and there are some that are highly

innovative. C# can be used to develop stand-alone apps, local and distributed components, web services and mobile code. It produces code that can target desktop PCs, mobile devices, servers and IoT devices. It is capable of supporting user interface, Internet, database and security projects. Hence it is an excellent all-round development language for all .NET applications. This intensive course aims to take experienced software engineers rapidly through all the major aspects of C# - using plenty of demo source code and hands-on labs to show it in action. This is an ideal first course for those moving to the C# language and the .NET Framework.

Contents of One-Day Training Course	
<p>Target Audience Experienced software engineers wishing to rapidly get up to speed with C#.</p> <p>Prerequisites Programming experience with an OO language such as C++, TypeScript or Java, along with good exposure to object-oriented design.</p> <p>No previous experience of C# or .NET is needed.</p>	<p style="text-align: center;">C# and the .NET Core</p> <p>What is the .NET Core? The Base Class Library The CLR How C# is used with .NET Delivery of C# functionality in assemblies</p> <p style="text-align: center;">A C# Project Walk through</p> <p>Solutions, projects and files Parts of a C# project Structure of code Setting up a solution with a C# app and class library</p> <p style="text-align: center;">Base Types</p> <p>Built-in data types .NET value types and reference types How C# and .NET data types compare Building code in C# that is callable from other languages</p> <p style="text-align: center;">Language Fundamentals</p> <p>Main starting point Flow control, operators Variables, methods Enumerators, bit flags, arrays, indexers Namespaces</p> <p style="text-align: center;">Class Fundamentals</p> <p>Members, constructors, visibility, ref and out, constant fields, structs Fields & properties, methods, nested types</p> <p style="text-align: center;">Inheritance</p> <p>Single inheritance only (for classes) Virtual functions Override and new keywords Designing libraries using inheritance</p> <p style="text-align: center;">Delegates And Events</p> <p>Equivalent of function pointers Defining and exposing delegates Registering an interest in a delegate Async info with events Design pattern for event handling</p> <p style="text-align: center;">Interfaces</p> <p>When to use interfaces Multiple inheritance & hierarchies Abstract classes vs. interfaces</p> <p style="text-align: center;">Exception Handling</p> <p>Try .. catch ... finally Detecting and responding to exceptions Strategies for exception handling</p> <p style="text-align: center;">Generics & Constraints</p> <p>Generics (for methods and classes) Constraints Partial types Anonymous methods Type inferencing</p> <p style="text-align: center;">Expression Bodied Members</p> <p>Succinct member definitions Methods, constructors, properties, indexers</p> <p style="text-align: center;">Specialist Features</p> <p>Null conditional operator Auto-property initializer nameof</p> <p style="text-align: center;">Calling C Code</p> <p>Calling out to C code from C# Passing parameters / accepting return val</p> <p style="text-align: center;">C# Project</p> <p>Developing a C# project using a selection of language features</p>

TypeScript 3.2

Object Foundations, Classes, Mixins, Generics, Specialist Types, Iterators, Ambients, Lib.d.ts

For modern larger-scale applications that target the JavaScript VM, either in browsers (e.g. Angular 7.2) or on the server and command-line tools (Node 11), or mobile apps (Ionic 4) or desktop apps (Electron 4), many senior developers have a desire for a more robust and comprehensive programming language compared to JavaScript; and TypeScript is the answer.

generics, decorators, interfaces, mixins, additional tools, ambient type declarations and lots more. This is convincing more and more project teams to adopt it as their core programming language. We see it use internally with Angular, Zone.js, RxJS, NgRx and many commercial applications (including very large ones).

TypeScript is a JavaScript-like language that transpiles to JavaScript so can run anywhere JavaScript runs. In addition to everything the JavaScript language offers, TypeScript also offers a much richer type system,

The aim of this course is to quickly bring you up to speed with programming in TypeScript. We explore the language syntax, its access to libraries, how to build applications and see why it is more and more being selected instead of JavaScript by senior web developers.

Contents of One-Day Training Course	
<p>Target Audience Developers wishing to create modern apps using TypeScript</p> <p>Prerequisites Software developers with an object-oriented background and some browser programming experience.</p> <p>Notes This course covers the very latest release, TypeScript 3.2.</p>	<p>TypeScript Introduction Relationship to ECMAScript standards Language tour What we should be familiar with and what may be new to us (any, never, tuple) tsconfig.json and transpiling</p> <p>Object Foundations Type system hierarchy Type inferencing Visibility and immutability</p> <p>Object Types Duck typing Type system ObjectType definition Properties and accessors Call signatures Tuples</p> <p>Classes, Interfaces & Mixins Defining a class Constructors Inheritance Specifying an interface What happens to interfaces after transpilation (they disappear!) Partial or full implementation of interface Additional construct which can be very useful in certain circumstances</p> <p>Generics and Constraints Type-independent code Separating algorithm from types Constraining permissible type parameters Relationship to transpiled code</p> <p>Namespace & Modules Modules as a unit of delivery and unit of code management Importing and exporting Sub-dividing module types in namespaces Use in conjunction with module naming</p> <p>Iterators & Generators Symbol.iterator and for..of Generator function</p> <p>Specialist Types Intersection type Union type Nullable Alias</p> <p>Reflection/Decorators/Metadata Attaching metadata to a class Using decorators The reflect-metadata package</p> <p>Ambient Declarations Interacting with non-TypeScript libraries and use of @types with npm Writing and publishing .d.ts files Ambient syntax</p> <p>lib.d.ts Standard Library A modular collection of ambient declarations for various targets</p> <p>New for TypeScript 3.x CallableFunction / NewableFunction project refs, unknown, enhanced tuple Properties on function declarations</p> <p>Project Using TypeScript in a project to build a modern flexible framework</p>

The Go Language

Goals, Workspaces, Go Type System, CGo, Packaging, Concurrency, Remote Repositories

The Go Language (GoLang) is a next-generation system programming language that is rapidly growing in popularity. Many wildly successful commercial and open-source projects such as Docker and Kubernetes are written in Go. It is appreciated for a range of innovations, such as built-in concurrency including its channel architecture, simplified tooling, sensible conventions, and lots more. This course is aimed at experienced developers and brings them quickly up to speed programming Go and being able to understand and enhance existing Go source trees and being able to build their own.

We cover creating Go commands and libraries, package management, interaction with repositories, structural typing. We look at Go's rich range of system packages. We see how lack of classes and inheritance is not a problem. We explore the use of Go in a range of popular projects and see what real-world benefits it brings to system-level programming.

Go is the language of choice for modern low-level work and it is increasingly being selected by cutting-edge system developers for their most challenging projects. This course helps each attendee become one.

Contents of One-Day Training Course	
<p>Target Audience Developers wishing to create modern system-level applications using the Go language.</p> <p>Prerequisites Software developers with an object-oriented and system programming (e.g. concurrency) background.</p> <p>Knowledge of C or other system-level language is a plus.</p> <p>Notes All samples and labs in this course use Go v1.11 - the very latest production release.</p> <p>Our instructor will use the GoLand IDE; attendees may use any suitable editor they wish to select</p>	<p style="text-align: center;">Go Introduction</p> <p>What problem set is Go trying to solve? A sensible evolution of C Excellent for system programming Setting up Go on your dev machine</p> <p style="text-align: center;">Tour Of Language Features</p> <p>Goroutines Packaging Building Functions as first class citizens Interfaces and structural typing Lack of classes; no inheritance</p> <p style="text-align: center;">Workspace Management</p> <p>The Go workspace is how code is managed Naming and file placement conventions Handling packages Building & using commands and libraries Role of \$GOPATH go build vs. go install Creating / importing packages / exports</p> <p style="text-align: center;">Go Type System</p> <p>bools, strings, ints, runes (codepoints), floats, complex Zero (default) values Conversions Type inferencing Variables with var, :=, constants</p> <p style="text-align: center;">Go Functions</p> <p>Defining function signatures Returning more than one result Named returns Advanced function usage</p> <p style="text-align: center;">Constructs</p> <p>if and if-else statements for loop / defer statement switch (how break is different) ("C's while is spelled for in Go" !!)</p> <p style="text-align: center;">Grouping of Data Items</p> <p>Arrays and maps Slices Structs and pointers</p> <p style="text-align: center;">Types and Their Interfaces</p> <p>Methods – functions with a receiver arg Interfaces define sets of methods</p> <p style="text-align: center;">Concurrency</p> <p>Concurrency primitives built into Go itself goroutines and channels Synchronization</p> <p style="text-align: center;">Testing</p> <p>Go has a built in testing system Test preparation The testing package Executing tests with go test</p> <p style="text-align: center;">CGo – Calling C Code</p> <p>Most OS APIs are written in C Need to call them and other C libraries How the pseudo-package C works How Go and C code can interact</p> <p style="text-align: center;">Remote Packages</p> <p>How to access remote repositories Use of go get Incorporating remote repositories into app</p> <p style="text-align: center;">Project</p> <p>Review of Go usage in a larger project.</p>

Python 3.7.2

Language Features, OO Constructs, Framework, Pip Installers, IPython, Jupyter Notebooks, Testing

Python is the world’s most versatile language. Unlike other languages that tend to be really good at one area but not so good at other areas, Python is really good at many areas. We see Python being successfully used for AI and machine learning, general application development, rich shell scripting, configuration, build systems, interactive notebooks, and lots more. Some of the largest technology companies in the world (e.g. Google) heavily use Python for their engineering and production systems. Some of the latest hot technologies (e.g. Keras and TensorFlow 2) use it extensively; so now is time for your team to use Python.

This fast-paced course covers all important aspects of Python programming., It is aimed at multi-disciplinary software engineers already experienced with object oriented programming using other languages. They will find much of their hard-earned knowledge easily transfers to Python programming – albeit delivered via a significantly simpler and more compact syntax. Invariably a given algorithm written in a different language when re-written in Python will result in smaller amounts of code, which is an excellent result (after all, the best developer writes the least amount of code).

Contents of One-Day Training Course	
	Python Tour
	What Python offers Feature tour What make Python different from competing languages Emphasis on clean syntax
Target Audience Developers wishing to create modern apps using the very latest version of Python.	Language Constructs
	Common data types Control flow Loops Functions RegEx
Prerequisites Software developers with practical programming experience of an object-oriented languages such as C++, C#, or Java. No prior Python experience needed.	OO Programming in Python
	Classes: layout, methods and attributes The <code>__init__()</code> method Inheritance Typing
	Runtime Features
	Memory management Generators Modules Multithreading & locks (threading.py)
	Error Handling
	Raising and catching exceptions (try, raise, except, ..) Designing with error handling in mind
	Framework
	The Python Standard Library offers: <ul style="list-style-type: none"> * Collections * File I/O * Data access * Network programming * User interface
	Pip
	Standard installer Pip usage Python Packaging Index Virtual environments Python modules
	(Interactive) IPython
	Interactive shell that supports a wide range of Python features, from visualization to threading to data access Also useful for and other languages
	Jupyter Notebook
	Jupyter Notebook mixes code, execution results, visualizations and markdown content in a single deliverable
	Embedding Python
	Many apps could benefit from a built-in macro language and Python is optimum We explore how to easily embed Python runtime in your custom application
	Testing
	Exploring Python’s testing infrastructure Unit testing – what’s similar & different Mocking Debugging
	Interacting with C
	Most OS APIs are written in C How Python and C code can interact – threading, memory, lifecycles, exceptions Handling common data types & constructs
	Project
	Using Python in a larger project to highlight its real-world capabilities

OCaml Functional Programming

FP Concepts, OCaml Language, OCaml Tools, OCaml Library, Dune, Opam, Projects Using OCaml

“ OCaml is an industrial strength programming language supporting functional, imperative and object-oriented styles ” [<https://OCaml.org/>]. OCaml is best known as a functional programming language and that is what we focus on in this intensive course. OCaml competes with Haskell to be the leading functional language. For a number of reasons, we prefer OCaml. It has a number of advanced features, a richer type system and a more extensive system library. It is also used on many cutting edge projects that interest us. Examples of practical uses of OCaml include: the experimental [redtt](#) (based on cubical type theory) and the well established

[Coq](#) proof assistants, samples in the important [TAPL](#) book, the WebAssembly [spec interpreter](#), the [mirageOS](#) unikernel and [Jane Street](#).

Functional programming is different from regular object-oriented programming. So we start by looking at FP for non-FP programmers. Then we explore all aspects of programming with OCaml – the language, tooling and system library. We also explore add-on libraries. Our goal is to ensure all attendees are up to speed with OCaml programming and immediately after this course can be productive as OCaml developers.

Contents of One-Day Training Course	
	<div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p>FP For non FP Programmers</p> <p>What is functional programming? Mathematics is important for modern code What is a variable (immutability) Pattern matching Handling change Pure functions Building more reliable code Many FP ideas are seeping into other types of programming – why and how?</p> <p style="text-align: center;">OCaml Tour</p> <p>Language basics (control flow, etc.) Structure source trees and individual units main function-conventional/not necessary Creating functions Events and callbacks Type inferencing</p> <p style="text-align: center;">OCaml Language</p> <p>Modules (submodules) .mli interface definition files More detailed look at functions Visibility Objects and object types Classes / polymorphism / class types Type mismatch and other errors</p> <p style="text-align: center;">OCaml Runtime</p> <p>OCaml runtime architecture How code executes (e.g. as native code) Dynamic linking -what’s involved in dynamically loading a library The GC Foreign function interface - interacting with C code (and other languages)</p> </div> <div style="width: 45%;"> <p style="text-align: center;">OCaml Library</p> <p>Structure of standard library Tour of main functionality areas Common collections Text handling Threads library (modules: thread, mutex, condition, event, ..) Async and deferred computation</p> <p style="text-align: center;">Tooling</p> <p>Debugging Testing Lint Pre-processor Compilation tools – (parsing, etc.) Abstract syntax tree</p> <p style="text-align: center;">Dune</p> <p>Dune is OCaml’s composable build system Defining steps needed for build Configuration S-Expression syntax Compilation flags</p> <p style="text-align: center;">Opam</p> <p>Opam is the OCaml package manager Package repository Extensive collection of pre-built packages Managing locally installed packages Packaging definition file – creating and populating for a custom project</p> <p style="text-align: center;">Projects</p> <p>Exploring usage of OCaml in a variety of open source projects that have shipped, to see how everything fits together in a larger production setting</p> </div> </div>
<p>Target Audience Experienced software developers who wish to start functional programming using the OCaml language.</p> <p>Prerequisites Attendees need prior programming experience in one of the mainstream object-oriented programming languages.</p> <p>Important: This course assumes attendees have no prior functional programming experience, so in addition to teaching OCaml, we also cover general aspects of functional programming.</p>	

Technology Faculty / Compute School

OS

- Windows System Programming Using C/C++
- Windows Multithreading Using C/C++
- Designing Server Platforms for Windows Server 2019 Using C/C++
- POSIX And Linux System Programming Using C/C++
- pthreads – POSIX and Linux Multithreading using C/C++
- Designing Server Platforms for POSIX And Linux Using C/C++

Container

- Containers and Microservices

Runtime

- Node.js 11.7 Runtime Programming Using TypeScript
- Browser Runtime Programming Using TypeScript
- Async, Parallel And Reactive (RxJS) Programming Using TypeScript
- Java 11 Runtime Programming
- .NET Core 2.2 CLR Programming
- .NET LINQ, Expression Trees And Rx

Windows System Programming Using C/C++

Using low-level Windows OS API for maximum performance, security, extensibility, flexibility

This course examines how to use the Windows C API to design and develop advanced systems-level software. The Windows C API has matured and gained a rock-solid quality reputation with modern features such as threads, symmetric multi-processing, system-wide object model, powerful networking, asynchronous I/O and Unicode. The Windows C API is the common programmatic interface shared by all implementations of the Windows OS family. There are some differences in how it behaves on each OS, but it's possible to create a single EXE/DLL to run on all OSes. In this course we focus on how it works on Windows [10 | Server 2019].

Important features in the areas of the registry, file systems, security and auditing are discussed. We cover the many techniques available for inter-process communication (e.g. pipes, mailslots, RPC and WinSock). Applications may be made run-time extensible by the configurable loading of DLLs.

Developers seeking extra performance and more flexible low-level control over OS system calls will benefit from writing their system-level application code in C/C++ and this course tells them what they need to know to quickly become productive.

Contents of One-Day Training Course	
<p>Target Audience System architects and experienced developers who need to create advanced systems using the Windows C API.</p> <p>Prerequisites Attendees must have some previous experience of system-level programming</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p>The Windows Platform</p> <p>Windows Architecture Windows SDK Issues of interest to app system developers</p> <p>Overview of Windows C API</p> <p>Applicability of Windows C API Important OS functionality Hardware issues Win32/Win64 on various OSes How to determine the underlying OS “Write to the API, not the OS”</p> <p>General Architecture</p> <p>Major OS components and subsystems Layout of OS and apps on file system Common Windows C datatypes & headers An application developer’s view of the OS</p> <p>Memory Management</p> <p>Flat memory structure Various types of alloc APIs Sharing memory / private heaps</p> <p>File System</p> <p>File APIs Directory handling Memory mapped files</p> <p>Security & Auditing</p> <p>How to programmatically interact with:</p> <ul style="list-style-type: none"> Security descriptors Security attributes SIDs and tokens ACLs and ACEs Privileges WinStations Desktops </div> <div style="width: 45%;"> <p>Registry</p> <p>Complete coverage of the Registry APIs Handling configuration data</p> <p>SEH</p> <p>Structured exception handling Exception handlers Termination handlers</p> <p>DLLs</p> <p>Comparison of DLLs & EXEs DLL functions / variables / memory Explicit loading of DLLs using: LoadLibrary FreeLibrary GetProcAddress</p> <p>C Run-Time Library (CRT)</p> <p>Using the CRT with Win32/Win64 How the CRT is layered above the OS API</p> <p>Inter-Process Communication</p> <p>Anonymous vs. Named pipes IPC using pipes & mailslots Distributing functionality using RPC</p> <p>Windows Networking APIs</p> <p>How to programmatically talk to the net Available APIs and when to use which Managing network connections with WNet</p> <p>WinSock & WinInet</p> <p>A protocol-independent API Relationship to Berkeley sockets WSA functions Coding client and server applications Blocking and non-blocking comms Advanced WinSock Socket options /out-of-band data</p> </div> </div>

Windows Multithreading Using C/C++ Threading Concepts, Kernel Obj, Processes/Threads, Synchronization, Async I/O, Debugging, DLLs

[Sample: [lab exercises](#)] This course explores how to use the Windows C API to build sophisticated multithreaded architectures. When designed correctly, multithreading can substantially increase application performance and responsiveness to distributed clients and end-users. The Windows kernel object is the opaque foundation for multithreading – based on this are the process, thread, job & various synchronization objects - mutex, event, semaphore and waitable timer – each of which targets different needs. Thread activity, lifetimes and inter-thread communication must be co-ordinated. Threads impact how to develop DLLs, memory and debug.

Various higher-level design patterns may be used to route workitems in multithreaded servers. The optimal server architecture is one active application thread per processor logical core. Tools may be developed to determine which thread is blocked waiting on which resource, and the state/owner of each resource. A server must efficiently multiplex many I/O requests over a few threads - which is the goal of I/O completion ports.

This course supplies attendees with an understanding of the concepts underlying threading, together with experience of multithreaded development.

Contents of One-Day Training Course	
<p>Target Audience System architects and experienced developers who need to gain an in-depth understanding of Windows multithreading.</p> <p>Prerequisites Attendees must have good experience of system-level programming, on either Windows or Linux.</p>	<p style="text-align: center;">Thread definition</p> <p>Scheduling vs. synchronization Parallelism and concurrency Compute-bound and I/O bound apps Race conditions, deadlock, starvation, priority inversion</p> <p style="text-align: center;">Kernel Objects</p> <p>Windows kernel objects Usage counting Kernel object handles Sharing handles among processes</p> <p style="text-align: center;">Processes</p> <p>CreateProcess API & child processes Retrieving the exit code of a process Job objects</p> <p style="text-align: center;">Threads</p> <p>CreateThread API & Threadproc The C runtime library Thread priority & processor affinity Thread management & lifetime</p> <p style="text-align: center;">Synchronization</p> <p>Critical Sections, Mutexes, Events, Semaphores, Waitable Timers, WaitForSingle/MultipleObject The “Protect data, not code” principle</p> <p style="text-align: center;">Memory and Threads</p> <p>Dynamic & static Thread Local Storage Heap storage vs. stack storage</p> <p style="text-align: center;">Asynchronous I/O</p> <p>Overlapped, APC & Scatter/Gather I/O Completion Ports, Asynchronous I/O, Overlapped, APC & Scatter/Gather I/O Completion Ports</p>
	<p style="text-align: center;">Thread Pools</p> <p>OS-managed pools of threads for processing timers, work-items and I/O</p> <p style="text-align: center;">DLLs and Threads</p> <p>How threads interact with DLLs Serialized DllMain, shared sections Robust DLL design for threads</p> <p style="text-align: center;">Debugging with Threads</p> <p>Querying information about running processes/threads and their attributes The serialized OutputDebugString API</p> <p style="text-align: center;">Resource Management</p> <p>Creating a custom resource browser, to display which thread is waiting on which synchronization resource Threads with C++ Threads & exceptions; threads & classes Accessing resources using smart pointers</p> <p style="text-align: center;">Design Issues</p> <p>Single Writer/Multiple Readers, Monitor, Once-Off Initialization, Dining Philosopher Calling legacy code from multiple threads Converting legacy code to multithreading</p> <p style="text-align: center;">Multithreaded Architectures</p> <p>Pipeline, Producer-Consumer, Work-Crew and Master-Slave Models Create threads on demand vs. elastic pool</p> <p style="text-align: center;">Multithreaded Project</p> <p>Development of a complete multithreaded embedded HTTP web server that uses I/O completion ports to efficiently manage large numbers of requests</p>

Designing Server Platforms for Windows Server 2019 Using C/C++

Installer, Service Process, Pipelines, Networking, Config, PerfMon, WMI, EIF, Patterns, HA/HT, Project

Server platforms consist of a mixture of multiple processes and threads, working in a co-ordinated manner, to provide some service to numerous clients on remote machines. These platforms must be flexible, extensible, configurable, scaleable and controllable. A pipeline architecture allows extensible processing of messages. Many techniques are available for flexible inter-process communication. Service processes are the best way to deliver long-lived non-GUI functionality.

Building server platforms for Windows Server 2019 is the logical choice for future-oriented projects.

If your team consists of senior developers experienced with Windows and C/C++ and your team is assigned the task of developing a high-quality server platform on Windows Server 2019, then this is the ideal course to get all team members up to speed on what is needed.

It covers design concepts, important Windows C APIs, plenty of code samples and a chance to have architectural questions answered. It explores extra features (such as ETW, clean installer, PerfMon) that will distinguish your team's platform from the competition.

Contents of One-Day Training Course	
<p>Target Audience System architects and senior software engineers who need to create advanced server platforms for Windows Server 2019 using C/C++.</p> <p>Prerequisites General Windows system programming and especially multithreading experience.</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <p>Designing Server Platforms Multiple Processes/Threads Central Service (Manager) / Worker Processes / GUI+CLI Admin Processes Variety of threading architectures</p> <p>Service Process SCM-service code interaction Install+config of service “Log on as a service” security right Controlling worker processes</p> <p>Platform Installation How best to install server apps Installer formats and server extensions</p> <p>Platform Configuration Rich config choices Web garden/web-farm layout Config changes without restarting</p> <p>Pipelines Processing paths for messages Sequential and non-sequential steps Structure of pipeline (handlers&modules) Pipeline context</p> <p>Dynamically loading DLLs Dynamically loading DLLs Updating a server's DLLs without having to restart it</p> <p>Networking High performance sockets design Eliminating buffer copying Async I/O & Scatter/Gather I/O</p> <p>Use of Http Server API Same http.sys kernel service as used by IIS</p> </div> <div style="width: 48%;"> <p>Performance Monitor Detecting bottlenecks Tuning performance Developing for Performance Counter</p> <p>WMI Windows Management Instrumentation Management classes Exposing your server through WMI Management by GUI and CLI</p> <p>Event Tracing For Windows (ETW) Tracing architecture Event tracing APIs</p> <p>Designing Platform Security Leveraging Windows' security features Defense in depth platform security Secure communication with remote clients</p> <p>Design Patterns for Server Platforms Patterns to satisfy competing demands Pooling, tuning, managing Sharing, distributing Extending, scheduling</p> <p>High Availability / High Throughput Hardware for high-availability/throughput ccNUMA, Interconnects SAN, DAS, NAS Clustering concepts Design for high availability/throughput</p> <p>Project Overview of development of a sample server platform for Windows Server 2019</p> </div> </div>

POSIX And Linux System Programming Using C/C++ Using low-level C API for maximum performance, security, extensibility, flexibility, portability

This course examines how to use POSIX standard APIs and Linux libc-specific APIs to create system-level software. POSIX defines specifications (documents) and operating systems (such as Linux with its libc library) or layered libraries (such as [musl](#)) implement these specifications (code). POSIX is supported on a variety of modern operating systems and so system-level application code should strive to use it as much as possible. Implementations can (and almost always do) add extra functions to the API list defined by POSIX and these extra functions provide very useful additional capabilities, though they limit portability.

Developers seeking extra performance and flexible low-level control over OS system calls will benefit from writing their system-level application code in C/C++ (rather than a language that comes with a heavy runtime, such as Java or C#). This course tells them what they need to know to become quickly productive.

We explore the entire path from application code to libc calls, through system calls right to the kernel where the actual functionality is delivered. We look closely at the architecture of how the OS programming interface is exposed to applications.

Contents of One-Day Training Course	
<p>Target Audience System architects and experienced developers who need to create advanced system software using POSIX and Linux libc APIs.</p> <p>Prerequisites Attendees must have some previous experience of system-level programming.</p> <p>Please note: This course does not cover pthreads (multithreading with POSIX) – we offer a separate course for pthreads development.</p>	<p style="text-align: center;">Big Picture</p> <p>How everything fits together - POSIX, the standard C library, libc, Linux System Calls, the Linux Kernel Alternative implementations: musl, newlib Portability & POSIX on non-linux OSes Role of the Linux Standard Base (LSB) What's needed for development</p> <p style="text-align: center;">Overview of POSIX Specs</p> <p>POSIX.1-2017 : latest spec with sections on base definitions, system interfaces (APIs), shell & utilities, rationale “Write to the API, not the OS” Relation between POSIX and C (“POSIX is a superset of the standard C library” and libc API is a superset of POSIX) What is specifically outside POSIX scope</p> <p style="text-align: center;">Linux General Architecture</p> <p>Structure of Linux kernel and userspace Role APIs and system calls play Major OS components and subsystems Layout of OS and apps on file system</p> <p style="text-align: center;">Linux libc</p> <p>Common datatypes & headers An application developer’s view of the OS Let’s trace a system call from app to API to System Call Interface to kernel Trap 0x80 for x86</p> <p style="text-align: center;">Major POSIX Functional Areas</p> <p>STREAMS, I/O, signals Regular expressions File descriptors & advanced file handling Process creation (fork/exec) & status</p> <p style="text-align: center;">Inter-Process Communication</p> <p>IPC overview Message passing semaphores Shared memory</p> <p style="text-align: center;">What libc Adds To POSIX</p> <p>What’s in libc above and beyond POSIX e.g. cgroups (underpins containers) Multimedia Newer libc system APIs not in POSIX Security concepts</p> <p style="text-align: center;">Linux Binaries</p> <p>File structure Programmatic interaction Creating binaries Dynamic loading of .so</p> <p style="text-align: center;">Scheduling</p> <p>Overview of Linux scheduling How apps use APIs to influence scheduling Priority</p> <p style="text-align: center;">Sockets</p> <p>How to programmatically talk to the net Available APIs and when to use which A protocol-independent API Coding client and server applications Blocking and non-blocking comms Socket options / out-of-band data Addressing, queuing signaling, errors, Advanced sockets</p> <p style="text-align: center;">Large Codebases</p> <p>Large codebases need more than POSIX Retain portability via e.g. Apache Portable Runtime or ACE</p>

pthreads: POSIX And Linux Multithreading using C/C++

Threading Concepts, Creating/Managing Threads, Synchronization, pthreads & .so, NPTL Internals

This course explores how to use the pthreads C API to build sophisticated multithreaded architectures for POSIX-compatible OSes such as Linux. When designed correctly, multithreading can substantially increase application performance and responsiveness to distributed clients and end-users. POSIX defines a multithreading specification commonly known as pthreads. This is a C API that strictly specifies the expected behavior of threading and synchronization primitives. Code written to work against pthreads can run on any OS that implements this spec. Linux is one such OS, and the focus for this course, but it is noted

that pthreads is also implemented on a wide variety of other popular and specialist OSes.

This course supplies attendees with an understanding of the concepts underlying threading, together with hands-on experience of multithreaded development on Linux. Topics covered include a comprehensive tour of thread creation and lifetime management, the various synchronization approaches, how threads interact with share libraries, memory access and debugging, intra-thread comms and various higher-level design patterns to work with large multithreaded servers.

Contents of One-Day Training Course	
<p style="text-align: center;">Target Audience</p> <p>System architects and experienced developers who need to gain an in-depth understanding of POSIX and Linux multithreading.</p> <p style="text-align: center;">Prerequisites</p> <p>Attendees must have good experience of system-level programming on Linux.</p>	<p style="text-align: center;">Thread definition</p> <p>Scheduling vs. synchronization Parallelism and concurrency Compute-bound and I/O bound apps Race conditions, deadlock, starvation, priority inversion</p> <p style="text-align: center;">Threads</p> <p>Tour of pthreads.h Overview of main APIs and C structures Creating a thread with pthread_create() The threadproc Thread priority Thread management & lifetime pthread_exit() and joining a thread to catch exit and to access exit code pthread_[detach attr_setdetachstate]() Thread cancelation C11/18 threads vs. pthreads (quite similar)</p> <p style="text-align: center;">Synchronization</p> <p>Conditionals, mutexes, rwlock, spin, barrier – compare & contrast The “protect data, not code” principle pthread_cond_[init destroy attr_init]() pthread_mutex_[lock trylock unlock] Blocking vs. non-blocking pthread_[rwlock spin barrier]_init()</p> <p style="text-align: center;">Thread Pools</p> <p>Managing pools of threads for processing timers, work-items and I/O</p> <p style="text-align: center;">Memory and Threads</p> <p>Thread local storage : pthread_key_create Heap storage vs. stack storage pthread_attr_[set get]stack[size addr]()</p> <p style="text-align: center;">Shared Libraries and Threads</p> <p>How threads interact with shared libraries Serialized methods Robust .so design for threads</p> <p style="text-align: center;">Debugging with Threads</p> <p>Querying information about running processes/threads and their attributes Serialized calls</p> <p style="text-align: center;">Resource Management</p> <p>Creating a custom resource browser, to display which thread is waiting on which synchronization resource Threads with C++ Threads & exceptions; threads & classes Accessing resources using smart pointers</p> <p style="text-align: center;">Design Issues</p> <p>Single Writer/Multiple Readers, Monitor, Once-Off Initialization, Dining Philosopher Calling legacy code from multiple threads Converting legacy code to multithreading</p> <p style="text-align: center;">Multithreaded Architectures</p> <p>Pipeline, Producer-Consumer, Work-Crew and Master-Slave Models Create threads on demand vs. elastic pool</p> <p style="text-align: center;">NPTL Internals</p> <p>Native POSIX Thread Library (NPTL) implements pthreads on Linux Threads and Linux scheduling</p> <p style="text-align: center;">Multithreaded Project</p> <p>Development of a complete multithreaded embedded HTTP web server that uses efficiently manage large numbers of requests</p>

Designing Server Platforms for POSIX And Linux Using C/C++

Installer, Daemons/Worker Process, Pipelines, Networking, Config, Performance, Syslog, Systemd, Patterns, HA/HT, Project

Server platforms consist of a mixture of multiple processes and threads, working in a co-ordinated manner, to provide some service to numerous clients on remote machines. These platforms must be flexible, extensible, configurable, scalable and controllable. A pipeline architecture allows extensible processing of messages. Many techniques are available for flexible inter-process communication. Service processes are the best way to deliver long-lived non-GUI functionality.

Building server platforms for Linux is the logical choice for future-oriented projects.

If your team consists of senior developers experienced with Linux and C/C++ and you are tasked with developing a high-quality server platform on Linux, then this is the ideal course to get all team members up to speed on what is needed. We recommend developing as much as possible using POSIX APIs (for portability), with careful use of additional APIs where it makes sense. This course covers design concepts, important POSIX APIs, plenty of code samples and a chance to have architectural questions answered. It explores extra features (such as Systemd and syslog) that will distinguish your team's platform from the competition.

Contents of One-Day Training Course	
<p>Target Audience System architects and senior software engineers who need to create advanced server platforms for Linux using C/C++.</p> <p>Prerequisites General Linux system programming and especially multithreading experience.</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <p>Designing Server Platforms Multiple Processes/Threads Central Service (Manager) / Worker Processes / GUI+CLI Admin Processes Variety of threading architectures</p> <p>Daemon Process Overview of systemd Install+config of daemon Security right Controlling worker processes</p> <p>Platform Installation How best to install server apps Installer formats and server extensions</p> <p>Platform Configuration Rich config choices (role of /etc) Web garden/web-farm layout Config changes without restarting</p> <p>Pipelines Processing paths for messages Sequential and non-sequential steps Structure of pipeline (handlers&modules) Pipeline context</p> <p>Dynamically loading .so Dynamically loading shared libraries Updating a server's shared libraries without having to restart it</p> <p>Networking High performance sockets design Eliminating buffer copying Async I/O</p> <p>Use of Apache Server API Building custom Apache HTTP Sever module to link it to your daemon process</p> </div> <div style="width: 48%;"> <p>Performance Detecting bottlenecks Tuning performance Developing with performance in mind</p> <p>Monitoring Building server platform with sysadmins in mind – what capabilities do they need? How to adding monitoring to your platform</p> <p>Syslog Logging architecture Event tracing APIs</p> <p>Designing Platform Security Leveraging Linux security features Defense in depth platform security Secure communication with remote clients</p> <p>Design Patterns for Server Platforms Patterns to satisfy competing demands Pooling, tuning, managing Sharing, distributing Extending, scheduling</p> <p>High Availability / High Throughput Hardware for high-availability/throughput ccNUMA, Interconnects SAN, DAS, NAS Clustering concepts Design for high availability/throughput</p> <p>Project Overview of development of part of a sample large server platform for Linux Focus as much as possible on POSIX APIs Portability and use of OS-specific APIs</p> </div> </div>

Containers And Microservices

Concepts, OCI, runC, CNCF, Containerd, Docker, Microservices, App Architecture, Networking, Project

Microservices have revolutionized server-side application development. Most modern engineering teams have evolved to running as much code as possible in containers and benefit from the range of enhancements they offer.

As an application developer, you can think of a container as an efficient sandbox within which your process runs (multiple processes can run in the one container, but usually it is one process per container). Containers offer a sandbox based on unique namespaces and c(ontrol) groups (e.g. resource limits/accounting).

The world of containers is undergoing rapid transformation (Docker and its components - e.g. runc - are important but they certainly are not the whole story). Developers really need to understand how all the moving parts fit together in the modern container world. They also need to be aware of how similar and contrasting containers are with traditional “process in OS” approach. This course focuses on individual containers and how attendees can build microservices (slice of their application) to run inside-a recommended follow-on course on *Kubernetes* explores how to orchestrate clusters of containers in innovative ways.

Contents of One-Day Training Course	
<p>Target Audience Server application developers wishing to compose applications from many microservices running in containers.</p> <p>Prerequisites Developers experienced in server-side software. Knowledge of the Go language is useful.</p> <p>Note Please note this course does not cover Kubernetes (apart from a brief introduction). We have a separate course dedicated to <i>Kubernetes</i>, which we recommend attendees takes after this course.</p>	<p>Container Landscape What containers offers over VM approach Intro to microservice development Big picture review of the sometimes confusing container landscape: its standard bodies, specs, commercial businesses, cloud offerings, tooling</p> <p>OCI - Open Container Initiative https://www.opencontainers.org and https://github.com/opencontainers “An open governance structure for .. creating open industry standards around container formats and runtime” Three specs: * Image is for file system layout * Runtime is how to run a container * Distribution (new) is how images travel Open source projects – runc (includes libcontainer), image-tools, runtime-tools (runc is the runtime used by Docker and by most Kubernetes installations)</p> <p>CNCF - Cloud-Native Computing Foundation https://www.cncf.io “builds sustainable ecosystems .. around a constellation of high-quality projects that orchestrate containers as part of a microservices architecture.” Review of CNCF projects, including ... * Containerd - https://containerd.io/</p> <p>Docker The Docker toolsuite is the market-leading container platform – let’s explore what</p> <p>services it offers and how to use them Available for Linux, macOS and Windows</p> <p>Kata Containers OCI-compliant open-source project to run containers via a light-weight hypervisor (combine containers and VM approaches)</p> <p>Building Microservices Sub-dividing a large app into microservices As an application developer, what steps you need to take to prepare your code to run inside a container</p> <p>Application Architecture What steps are needed to build a microservice and how to optimize? Handling data, IPC, config, lifecycle, etc.</p> <p>Microservices And Security Under what security context does your microservice code run? Importance of isolation</p> <p>Microservices And Networking * CNI (container networking) - https://github.com/containernetworking * Envoy (distributed proxy) - https://www.envoyproxy.io</p> <p>Tour of Source Trees Understanding internals is good: exploring runc CLI and libcontainer (written in Go), containerd (written in Go), Docker (written in Go), Kata (yup, also written in Go)</p> <p>Project How best to partition a large server app to run as microservices in many containers</p>

Node.js 11.7 Runtime Programming Using TypeScript Feature Tour, Event Loop, Non-Blocking IO, VM, Crypto, Utilities, Streaming, N-API, Project

Node.js is a cross-platform easy-to-use runtime based on Google’s high performance V8 JavaScript execution engine (as found in Chrome). Node also comes with a well crafted and substantial framework which covers many application areas.

Node has a number of desirable core characteristics – simplicity (it is very easy to get started using Node); modularity (everything is based on modules, which can grow over time); extensible (both by JavaScript / TypeScript code and by C/C++ code [using N-API]) and asynchronous / event-driven (non-blocking IO).

This course cover the latest edition – Node.js 11.7 – from the ground up for developers with little or no previous Node experience but a strong desire to rapidly become proficient in Node. We compare Node to other popular runtimes that they might know and see there are many similarities (language VM, package management, framework layout, tools, etc.) but also some differences.

Note this course does not cover the Node.js HTTP[S/2] modules – we have a separate detailed course covering these along with Express and PUG, which is an ideal follow-on course to this one.

Contents of One-Day Training Course		
<p>Target Audience Experienced web developers who wish to get up to speed developing for the Node.js runtime using TypeScript.</p> <p>Prerequisites No previous experience of Node.js required. General experience with web programming is required. All demos and lab exercises will be in TypeScript, so attendees need to know TypeScript.</p>	<p>Feature Tour of Node.js 11.7 Node.js 11.7 is the latest edition of this very popular server-side runtime for web, CLI and other workloads – let’s see what Node has to offer</p> <p>Node And TypeScript Most Node apps up to now have used JavaScript – we will use TypeScript Benefits of TypeScript for server coding Using Node from TypeScript (.d.ts files)</p> <p>Launching Node Command line options for Node itself Environment settings Keeping a node app running</p> <p>Event Loop Review of how the event loop works The important role events play in Node The Events module EventEmitter Preference for asynchronous Role of listeners</p> <p>Non-Blocking IO Accessing the file system Accessing networks</p> <p>Network Programming DNS UDP/Datagram TCP using the Net module IPC servers using the Net module</p> <p>V8 Google’s V8 engine provides the core execution environment for node Using the V8 module</p>	<p>OS Module Portable access to capabilities/services of the operating system on which Node runs</p> <p>VM The VM module can be used to compile and execute code in a language VM</p> <p>Utilities Utilities module TTY Console Timers StringDecoder</p> <p>Crypto Cipher Decipher Handling certificates Hashing</p> <p>Streams Stream types – readable, writable, duplex and transform The readline module process.stdout</p> <p>N-API Building portable C/C++ modules that work with Node using the new N-API</p> <p>Advanced Node Process and child processes The cluster module Shared server ports and child processes</p> <p>CLI Project Building a command-line app showing how to use many parts of the Node framework together in a realistic app</p>

Browser Runtime Programming Using TypeScript File API, IndexedDB, GeoLocation, Beacon, HTTP/x, Events, CORS, Fetch, Formats and Device Info

The programming environment inside the modern web browser has significantly matured and now offers a rich and diverse range of capabilities, some at the UI level and some at the underlying runtime level. Many of the newer features are currently not being fully exploited by web developers, who focus exclusively on the UI. This course aims to change that by exploring in depth the non-UI aspects of browser programming, using the TypeScript language for all demos and lab exercises.

many natural advantages over mobile apps (here's four: the power of the URI, works everywhere, cloud-friendly, immediate app updates). In areas such as sandboxed file access, networking, data formats and device info, a modern browser offers the web application developer a comprehensive selection of functionality that, when used correctly, can easily compete with what is available for native mobile apps.

The runtime programming APIs in a modern browser now rival what modern OSes offer. A web app has

IMPORTANT: This course does not cover parallel (web worker) & asynchronous programming – we offer a separate full course that covers these topics in detail.

Contents of One-Day Training Course	
<p>Target Audience Web developers wishing to fully leverage the runtime (non-UI) capabilities of modern web browsers.</p> <p>Prerequisites Good experience of web development, including HTML 5.3.</p> <p>Knowledge of the TypeScript programming language.</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <p style="text-align: center;">Modern Web Platform</p> <p>Tour of the modern web platform (what we already know / what is less familiar) Major features and which browsers implement them (caniuse.com and iwanttouse.com websites) Dynamically detecting in code available browser capabilities and optimizing app</p> <p style="text-align: center;">File API</p> <p>File Sandboxing Accessing and updating files File metadata and raw blob content File[Reader Writer system] API</p> <p style="text-align: center;">Indexed DB</p> <p>A B-tree like persistence mechanism with powerful indexing that can serve as basis for in-browser client database Comparison with other storage options</p> <p style="text-align: center;">Text Handling And JSON</p> <p>Handling various text encodings TextEncoder and TextDecoder JSON parsing and generation Use of JSON in the browser</p> <p style="text-align: center;">GeoLocation</p> <p>“This specification defines an API that provides scripted access to geographical location information associated with the hosting device.”</p> <p style="text-align: center;">Beacon</p> <p>Sending data asynchronously to server (e.g after page close) navigator.sendBeacon Setting priority</p> </div> <div style="width: 48%;"> <p style="text-align: center;">HTTP/2 & HTTP/3</p> <p>Latest generation of HTTP protocol Binary, multiplexed, full duplex, priority Server push New app architectural possibilities</p> <p style="text-align: center;">Server Sent Events</p> <p>How it works EventSource API Event streams Event handlers</p> <p style="text-align: center;">CORS</p> <p>Cross-Origin Resource Sharing Role of origin in HTTP protocol cross-origin access CORS preflight request and CORS request</p> <p style="text-align: center;">Fetch</p> <p>A significantly improved replacement for XMLHttpRequest “The Fetch standard defines requests, responses, and the process that binds them: fetching.” (WHATWG)</p> <p style="text-align: center;">Data Formats</p> <p>Brotli Compressed Data Format Data URIs using base64 encoding data-* attributes</p> <p style="text-align: center;">Device Info</p> <p>Accessing device details Vibration API Battery status</p> <p style="text-align: center;">Project</p> <p>Bringing together the ideas covered in this course to design and build a specialist runtime engine to execute in a browser</p> </div> </div>

Asynchronous, Parallel & Reactive (RxJS) Programming Using TypeScript

Managing time, multiple workers and data streams

Coordinating multiple activities is one of the most difficult areas of advanced application development. In this course we explore all the different options open to developers and see how they can be integrated into modern TypeScript applications.

All modern hardware (even low-end mobile devices) support multiple CPU cores and allow parallel code execution. All have timers that allow asynchronous workloads to be queued for execution in the future. It is up to app developers to exploit the varying capabilities of hardware available to them to deliver optimum apps.

This course is ideal for TypeScript developers who wish to more tightly manage how their code and data are processed - where, when and in what order.

All samples and labs in this course use TypeScript as we think it is best for larger applications – much of what is covered is useful to JavaScript developers too.

We explore code both running on the server (e.g. as part of a Node server application) and in the browser (as standalone TypeScript code, or as part of a larger Angular application).

Contents of One-Day Training Course	
<p>Target Audience TypeScript developers who wish to have more control within their apps over time, multiple workers and observable data streams</p> <p>Prerequisites Understanding of asynchronous and multi-threaded programming from other environments, together with knowledge of TypeScript.</p>	<p>Options for Async & Parallel Review of all the possibilities available to TypeScript developers to manage time, distribute code across contexts to be executed and handle data streaming</p> <p>JavaScript VM Event Loop Need to fully understand the event loop and event ordering to optimize our code Adding events and consuming events</p> <p>Promises & Timers How a promise works: in real life / code Programming with a promise Error handling Using VM timers What <code>setTimer(0, <func>)</code> means Promises/A+ (https://promisesaplus.com/)</p> <p>TypeScript async/await Making asynchronous source more readable while maintaining capabilities Use of <code>async</code> and <code>await</code> keywords Designing an asynchronous framework</p> <p>Web Workers A web worker is a thread Threads cooperate via message passing Creating and managing web workers Types of web workers Worker lifetime control [Dedicated Shared]WorkerGlobalScope HTML5.3 WindowOrWorkerGlobalScope</p> <p>Messaging Between Workers Message ports / Message parameters PostMessage / onMessage Organizing message flows</p> <p>Shared Array Buffer & Atomics Shared memory within a browser Atomics provide synchronization to protect shared array buffers</p> <p>Zone.js Architecture of Zone.js Multiple zones can live within the same web worker or main browser context Creating and using zones Zone.js is heavily used in Angular - how?</p> <p>RxJS Overview Observable / Observer / Subject Hot & cold next/error/complete Testing RxJS code using jasmine-marbles</p> <p>RxJS Streams An observable as a dual of an enumerable Subscriptions App architecture and stream processing</p> <p>RxJS Operators Processing individual elements via a large collections of operators What's new with pipeable operators Custom operators</p> <p>Advanced RxJS Notifications Schedulers (incl. new TestScheduler) Connectables</p> <p>Project Having explored the fundamental constructs for managing time, code execution and data streams, we conclude with a project that demonstrates all these ideas together.</p>

Java 11 Runtime Programming

IO, Processes, Multithreading & Synchronization, Serialization, Security, Networking, Web Access

After developers learn the Java 11 Language they next must learn about the Java 11 runtime environment and the APIs it provides. Java and its add-on packages offer a vast range of APIs and often it can be daunting for developers new to Java to figure out what goes where. Initially, to simply get work done for their specific assignments can be a challenge. This course aims to overcome this and takes developers already proficient in the Java language on a walkthrough of common scenarios – we look at relevant APIs and the runtime ideas underlying them and help attendees write code efficiently and become productive as Java devs.

Beneath Java on every implementation is an OS, whose capabilities are exposed to Java applications via an API. The Java runtime itself, known as the JVM, adds additional capabilities. Base class libraries and layered libraries offer even more functionality. Taken together, a rich multi-layer of readily available functionality is provided for application developers to exploit in their own applications.

The aim of this rapid-paced course is to cover as much as possible of the fundamental APIs that devs need and provide them a good grounding in practical API usage.

Contents of One-Day Training Course	
<p>Target Audience Developers wishing to create libraries and applications using Java's runtime capabilities.</p> <p>Notes this course does not cover user interface features of Java.</p> <p>Prerequisites Attendees must already have attended our <i>Java 11 Language</i> course or have similar Java language programming experience.</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <p style="text-align: center;">Overview of Java Runtime</p> <p>Documentation and tooling How everything works together Overview of module layout Interaction with the VM Tour of all major runtime features</p> <p style="text-align: center;">java.io</p> <p>File handling File and string readers and writers Buffering java.util.zip</p> <p style="text-align: center;">Serializable</p> <p>Serializing and deserializing an object NotSerializableException Object stream APIs</p> <p style="text-align: center;">Managing Processes</p> <p>Process class represents a process Creating a process with <code>ProcessBuilder</code> Redirect via <code>ProcessBuilder.Redirect</code></p> <p style="text-align: center;">Multithreading</p> <p>Java's threading architecture The Runnable interface Creating threads with <code>java.lang.Thread</code> <code>ThreadLocal</code> – each thread has its own copy of the variable <code>ThreadGroup</code></p> <p style="text-align: center;">Synchronization</p> <p>A synchronized block <code>java.util.concurrent</code> for concurrency <code>java.util.concurrent.atomic.*</code> supports lock-free thread-safe programming on single variables <code>java.util.concurrent.locks=locks</code></p> </div> <div style="width: 48%;"> <p style="text-align: center;">Utilities</p> <p>Text handling & regex Internationalization Time / mathematics / etc.</p> <p style="text-align: center;">Java And Security</p> <p>A comprehensive security framework authentication, authorization, auditing Java and PKI Use of cryptographic algorithms</p> <p style="text-align: center;">Advanced Security Features</p> <p><code>SecurityManager</code> <code>Keystore</code> Code security – code signing, bytecode verification, avoiding common threats</p> <p style="text-align: center;">Java Networking</p> <p>Socket programming with Java specifying network addresses Socket options Creating UDP and TCP connections</p> <p style="text-align: center;">Web Access</p> <p>HTTP 1.1, HTTP/2 (JEP110) and HTTP/3 URI SSL/TLS</p> <p style="text-align: center;">Reflection</p> <p><code>java.lang.Object.getClass()</code> How to use <code>java.lang.Class</code> Reflection namespace – <code>java.lang.reflect</code> Constructor, Field, Method, Parameter</p> <p style="text-align: center;">Additional Libraries</p> <p><code>java.instrument</code> and logging Transactions Java Management Extensions (JMX) Java Naming & Directory Interface (JNDI)</p> </div> </div>

.NET Core 2.2 CLR Programming

Architecture, Assemblies, Type/instance, Attributes, Reflection, Security, Nuget, Unmanaged

The .NET Core 2.2 Common Language Runtime (CLR) is the foundation for all aspects of the modern .NET Core initiative. It offers leading edge services for applications on Windows, macOS and Linux. The .NET Core CLR provides a “managed” runtime environment, in the sense that an in-process runtime engine, separate from your code, to assist with its execution. Its modern feature-set includes a system-wide object model, full control of memory management, granular security, innovative metadata, sophisticated type loading, and virtualized contracts between types (so that their physical layouts may be defined / optimized at runtime).

Regardless of which .NET Core application type you are building – rich client UI, web UI, web service, class library, Windows Service, console - you must understand how it works with the CLR and how it can programmatically interact with the capabilities of the CLR. The .NET Core Framework works with the CLR to provide a rich *.NET Standard*-based class library.

.NET Core is becoming the platform of choice for new projects. It is highly regarded and is gaining a reputation as a solid foundation for innovative solutions.

Contents of One-Day Training Course	
<p>Target Audience System architects and senior software engineers who need to rapidly get up to speed with .NET Core CLR programming.</p> <p>Prerequisites General understanding of high level .NET Core concepts.</p> <p>Attendance at our <i>C# Language</i> course or equivalent experience is needed.</p>	<p style="text-align: center;">Overview</p> <p>Multiple implementations of .NET .NET Standard is a specification that mandates how all should work .NET Framework: stable, mature (20yrs), slowly-evolving/Windows only, in OS .NET Core: new, Windows/macOS/Linux, ships with your app (so faster updates) For modern codebases, choose .NET Core</p> <p style="text-align: center;">CLR Architecture</p> <p>Feature-rich runtime for many languages What capabilities does it provide?</p> <p style="text-align: center;">Assemblies</p> <p>An assembly is a (DLL-like) delivery & management unit for CLR types Strong names / versioning Modules and assemblies Locating assemblies</p> <p style="text-align: center;">Types</p> <p>Importance of type / CLS & CTS Types and different languages System.Object & identity</p> <p style="text-align: center;">Advanced Types</p> <p>Methods – virtual, abstract, overloaded Type hierarchy / relationships / generics Properties, enumerations, constants, fields</p> <p style="text-align: center;">Instances</p> <p>Types & instances Reference type and value type Finalisation / IDisposable Garbage collection / stack and heap Calling methods/anonymous methods Callback techniques</p>
	<p style="text-align: center;">Compiler Options</p> <p>Native compiler vs. JIT - contrast Using .NET Native .NET on macOS and Linux</p> <p style="text-align: center;">Nuget</p> <p>Using .NET’s package manager Creating your own packages</p> <p style="text-align: center;">.NET Core 2.2 Framework</p> <p>Extensive quality class library Solves problems with .NET Framework Portable and small</p> <p style="text-align: center;">Config</p> <p>Externalised Configuration Handling configuration data Format of config files & custom settings Application and machine configuration</p> <p style="text-align: center;">Custom Attributes</p> <p>Advantages of using attributes in IL Pseudo-custom attributes Creating & detecting attributes</p> <p style="text-align: center;">Metadata & Reflection</p> <p>What data is stored along with code Manifest & metadata tables Peeking inside an assembly Discovering types</p> <p style="text-align: center;">Security</p> <p>Identity issues & overview of .NET Core security architecture Certificates & permissions Command-line and UI security tools</p> <p style="text-align: center;">The Unmanaged World</p> <p>P/Invoke How to call C libraries</p>

.NET LINQ, Expression Trees And Rx Lambda/Query Expressions, LINQ-To-Objects, Trees, Providers, Rx Observ[able|er], Subjects, Operators

.NET's LINQ (Language INtegrated Query) is an innovative querying capability built into the .NET Framework and languages such as C#. Querying plays a major role in most applications. For .NET and C# to pay particular attention to how code can query a variety of data sources (e.g. SQL databases with EF Core, XML documents, observable streams with Rx, collection classes, management objects, ..) can be most beneficial.

Once .NET application developers learn the basics of LINQ they can then explore how to extend it and how to expose custom data sources as LINQ targets.

Think of .NET [expression trees](#) as an AST of a query. They are used in LINQ and many custom solutions. When LINQ providers talk to remote data stores, expression trees are at the heart. Expression trees can automatically be created by a .NET compiler and also directly interacted with by application code.

Rx (Reactive extensions) is “an API for asynchronous programming with observable streams” [[link](#)]. It is based on the observable pattern ([one of the GoF patterns](#)). Rx is available for many languages (e.g. Angular uses it extensively), this course covers Rx+C#.

Contents of One-Day Training Course	
<p>Target Audience Software engineers wishing to learn about the foundation of LINQ, Expression Trees and Rx.</p> <p>Prerequisites Attendees are expected to be experienced C#/.NET developers with some awareness of data querying, collections and abstract syntax tree usage.</p>	<p style="text-align: center;">Overview: LINQ, Expression Trees and Rx</p> <p>Introduction to each, how they fit together and how they might be used in applications Functional programming ideas</p> <p style="text-align: center;">Review of C# Features</p> <p>Modern features of C# that are of interest Lambda expressions, anonymous types, extension methods, var, flexible object initialization, partial classes/methods Func<> and Action<></p> <p style="text-align: center;">LINQ Concepts</p> <p>Immediate & deferred execution Query Syntax vs. Method Syntax What are operators</p> <p style="text-align: center;">Use Query Expressions</p> <p>Query Expression Syntax SQL-like format Typing</p> <p style="text-align: center;">Standard Query Operators</p> <p>Tour System.Linq namespace Exploring provided operators Adding custom query operators Inputs and outputs Moving to saying what you want Lambda, delegates and expression trees IEnumerable<T> vs. IQueryable<T></p> <p style="text-align: center;">Designing LINQ Providers</p> <p>How is the network involved? How does data flow? Where, and when, are expressions run? Creating a custom LINQ Data Provider Programmatically querying custom data</p>
	<p style="text-align: center;">Expression Trees</p> <p>Introduction to expression trees Expression trees and IQueryable<> Dynamically compiling expression trees Review of important expression types System.Linq.Expressions The LambdaExpression type Factory methods</p> <p style="text-align: center;">Debugging</p> <p>How to debug LINQ code Writing your own debug visualizer for expression trees</p> <p style="text-align: center;">ReactiveX for C#</p> <p>Observable and Observer Enumerable vs. observable Rx operators Subject Disposables How to use Rx within application code</p> <p style="text-align: center;">Advanced Rx</p> <p>Rx and .. Handling time Threading Subscriptions Notification Materialization Cold vs. hot Scheduling Exceptions</p> <p style="text-align: center;">Custom Rx</p> <p>Building your own observers, observables and operators Managing subscription lifetimes</p>

Technology Faculty / Data School

Storage Subsystem

- Fundamentals Of Storage

Data Access

- RDF-OWL-SPARQL
- SQL
- XML

ORM

- .NET Entity Framework Core 2.2 Using C#

Format

- PDF Programming

Fundamentals Of Storage

Disk Hardware, Connectors, OS Storage Subsystems, File Systems, RAID, Tx, Caching, Hashing, B-Trees

This course provides a comprehensive tour of the important storage fundamentals that all technologists need to understand in order to build, provision and operate modern IT solutions that involve storage (which of course means all such solutions). There's a lot more to storage than the hard disk on a PC or simple APIs such as fopen() / fwrite() that developers might use. In this course we explore storage end-to-end and see how it interacts with remote disks, hypervisors, the cloud and various in-memory representations. We see how the distinction between storage, database and memory is being blurred. We see how system-wide reliability and

performance demands impart on storage. We see how modern OSes and their powerful storage stacks allow rich journaling file systems and databases to be built that deliver a wide variety of enhanced storage features. We also explore specific technical approaches to transactioning, caching, hashing and B-Trees.

A good understanding of storage fundamentals along with clear knowledge of the storage-related technical options available to choose from helps all involved in engineering to make the optimum design decisions.

Contents of One-Day Training Course	
<p>Target Audience Developers, devops, IT professionals, engineering managers – all of whom need an understanding of the core building blocks of storage platforms.</p> <p>Prerequisites Good all-round knowledge of modern computing infrastructure at a technical level.</p>	<p style="text-align: center;">Storage Concepts</p> <p>Latency Throughput Reliability “bit-rot” and other issues</p> <p style="text-align: center;">Modern Storage for ...</p> <p>File Object XML SQL Graph</p> <p style="text-align: center;">SCSI – Talking To Storage</p> <p>Small Computer System Interface (SCSI) is a comprehensive protocol / command / interface between controller and disk</p> <p style="text-align: center;">Disk hardware and connectors</p> <p>Traditional disk SATA / SAS SSD and Flash NVMe and PCI Express RAM disk</p> <p style="text-align: center;">Tape, Optical</p> <p>Managing larger amounts of data Tape and optical formats used for backup, transporting and long-term archiving</p> <p style="text-align: center;">DAS, NAS, SAN</p> <p>Directly Attached Storage Network Attached Storage Storage Area Network</p> <p style="text-align: center;">Storage Subsystem In An OS</p> <p>Storage and networking are the largest subsystems inside any OS Exploring an OS storage architecture</p>
	<p style="text-align: center;">Modern File Systems</p> <p>Journaling 128-bit sizes Virtual File Systems And Client File APIs Detailed look at internals of a modern file system – ZFS</p> <p style="text-align: center;">RAID</p> <p>Costs and benefits of various RAID types Stripping When errors occur in hardware and in software</p> <p style="text-align: center;">Transactioning</p> <p>Transaction principles Two phase commit Hierarchical transaction groups Savepoints Resource manager Enlistment</p> <p style="text-align: center;">Caching</p> <p>Putting data close to where it is used Cache lifetimes Keeping data in cache fresh Caching architecture</p> <p style="text-align: center;">Identity And Hashing</p> <p>How to identify storage objects Role of hashing algorithms Importance of hashing for de-duplication and other aspects of storage</p> <p style="text-align: center;">B-Tree</p> <p>B-Tree is the most important data structure for storage architectures Performance characteristics of B-Trees Variations of B-Tree layout</p>

RDF, OWL And SPARQL

Ontologies, Standards, Triples, Datatypes, RDF, RDF Schema, OWL, Entities, SPARQL, Reasoning

This course provides a guided tour of the world of knowledge representation and reasoning, from an ontology perspective. We will see that a graph is the most scalable, extensible and distributed form of knowledge representation and facilitates knowledge reasoning to incorporate additional facts.

Resources are identified with internationalized URIs (IRIs). We start with an agreed set of basic datatypes (number types, strings, URI, dates), then use them to define simple statements in the form of triples (subject-predicate-object), then build groups of such statements into graphs, and then combine multiple graphs into datasets.

W3C has defined a layered set of standards related to defining and querying ontologies. We will discover how each standard builds on a foundation to add more ontology-related functionality and together they define a comprehensive solution to ontology management.

We add a query engine in front of such datasets to make them accessible to distributed clients. We can define rules to add more control over ontology interaction. We support knowledge reasoning via inferencing.

Contents of One-Day Training Course	
<p>Target Audience Software engineers and knowledge engineers who wish to learn more about creating and interrogating ontologies using the latest W3C standards</p> <p>Prerequisites A software engineering background with some experience of creating semantic models</p>	<p style="text-align: center;">Ontologies</p> <p>Representing knowledge The knowledge graph Description Logics (DL) Reasoning about knowledge Tour of the world of ontologies</p> <p style="text-align: center;">Ontology Standards</p> <p>W3C has been very active in defining a suite of standards related to ontologies Review of W3C layered standards</p> <p style="text-align: center;">Protégé Tooling</p> <p>Stanford University has created the Protégé tool (http://protege.stanford.edu) - "A free, open-source ontology editor & framework for building intelligent systems"</p> <p style="text-align: center;">XML Datatypes</p> <p>The common primitives (int, string, date) for ontologies and the knowledge graph Value space vs. lexical space Facets for specialization</p> <p style="text-align: center;">Introduction to RDF</p> <p>Role of Resource Description Framework Statement: subject, predicate, object Managing triples</p> <p style="text-align: center;">RDF Schema</p> <p>Extending the RDF vocabulary Defining classes and their properties Reification</p> <p style="text-align: center;">Introduction to OWL</p> <p>Web Ontology Language (OWL) expands the vocabulary for representing knowledge Tour of OWL capabilities – axioms, etc. Literals/datatypes/dataranges/expressions</p>
	<p style="text-align: center;">OWL Entities</p> <p>Named Individual Class Datatype Object Property Data Property Annotation Property</p> <p style="text-align: center;">SPARQL Introduction</p> <p>SPARQL is to ontologies what SQL is to a relational database – a flexible language to query and update knowledge graphs</p> <p style="text-align: center;">Advanced SPARQL</p> <p>Result formats Update Federated queries</p> <p style="text-align: center;">Existing Datasets</p> <p>Exploring available big datasets, e.g.: - Dbpedia (http://wiki.dbpedia.org/) - Wikidata (https://www.wikidata.org)</p> <p style="text-align: center;">Reasoning & Rules</p> <p>Inferencing over ontologies Discovering new relationships Defining rules using RIF</p> <p style="text-align: center;">Ontologies & Machine Learning</p> <p>Some folks think of ontologies as competing with ML – we look at this question and how to use them together</p> <p style="text-align: center;">Ontology Design</p> <p>Review of how to create ontologies using what we have learnt</p> <p style="text-align: center;">Project</p> <p>Designing a large software solution incorporating ideas explored in this course</p>

Structured Query Language (SQL)

Relations, SQL Overview, The Table, DML, Joins, DDL, Stored Procedures, Transactions, ORM

Structured Query Language (SQL) is the very popular purpose-built query language for relational databases. A mathematical relation forms the logical underpinning for a relational table. A table consists of an unordered set of rows; each row consists of an unordered set of columns. Columns are named and are associated with a datatype. Some columns has specific characteristics (e.g. primary/foreign keys). Indices can be attached to some constructs which greatly improve performance.

Whether building cloud, enterprise or mobile solutions, devs should at least consider and often select SQL as

the basis for their data management needs. It offers a wide gamut of capabilities and is massively supported across all computing platforms.

There are many SQL products in the marketplace. This course explores standard SQL. Each database vendor adds its own nuances to this standard language, but all support a large core set of SQL functionality, and it is this core that we cover in this course. After this foundational course developers will be ready to explore the product-specific documentation for their selected SQL engine & selected app programming language.

Contents of One-Day Training Course	
<p>Target Audience Developers who wish to learn SQL by focusing on the large set of SQL features that is common to modern relational databases.</p> <p>Prerequisites Experienced software engineers with some data manipulation background.</p> <p>No previous SQL experience is required.</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p style="text-align: center;">SQL Overview</p> <p>The mathematical relation Representing knowledge via tables Query language for relations Parameterized queries Stored procedures</p> <p style="text-align: center;">Role of SQL</p> <p>Tour of SQL in modern database engines, both client/server & library-based, such as: - Microsoft SQL Server - Sqlite - MySQL - Oracle</p> <p style="text-align: center;">Defining a Table</p> <p>Datatypes Primary key Foreign key Normalization</p> <p style="text-align: center;">Intro to DML</p> <p>SELECT INSERT UPDATE DELETE</p> <p style="text-align: center;">Advanced DML</p> <p>WHERE ORDER BY DISTINCT Aggregates (AVG, SUM, MIN, MAX)</p> <p style="text-align: center;">Joins</p> <p>Combining statements in a join Inner join Left/right join Full join</p> </div> <div style="width: 45%;"> <p style="text-align: center;">DDL</p> <p>CREATE and ALTER DROP TRUNCATE</p> <p style="text-align: center;">Stored Procedures</p> <p>Creating and calling stored procs Parameters and local variables Additional programming constructs (BEGIN/END, TRY-CATCH, etc.)</p> <p style="text-align: center;">ACID</p> <p>Atomicity Consistency Isolation Durability</p> <p style="text-align: center;">Transactions</p> <p>Unit of work All or none Begin Tx and commit or rollback</p> <p style="text-align: center;">SQL And Security</p> <p>SQL injection attack Dangers of dynamically building queries GRANT and REVOKE</p> <p style="text-align: center;">ORM-Object Relational Mapper</p> <p>Many object-oriented programming languages use an ORM to manage SQL queries - how do they work?</p> <p style="text-align: center;">SQL vs. NoSQL</p> <p>Contrast SQL and NoSQL approaches to data storage and data management Suitability for different scenarios</p> <p style="text-align: center;">Project</p> <p>Use of SQL as the basis for data in a comprehensive enterprise solution</p> </div> </div>

eXtensible Markup Language (XML)

XML InfoSet, XML Serialization, Namespaces, DOM, XML Schemas (XSD), XPath, XSLT, XML in App Design

XML is well supported on all technology platforms by many editors, tools and framework vendors. As it has proven its worth in the field, it has become a highly desirable feature to leverage in applications. Hence it is now a mandatory part of the skills set for modern software developers. The W3C has defined a cohesive series of XML standards, covering core information modeling, how data is to be serialized, a Document Object Model defining a programmatic API, how data is transformed and many more standards. XML has become the foundation for whole swathes of functionality in various computing environments.

Document file formats (such as ODF and Office Open XML) have standardized on XML. Many vertical industries (e.g. <http://startndc.iata.org/> or <http://hropenstandards.org>) have defined custom XML schemas. Data exchange is exploiting it. Data delivery over the Internet is using it.

There are three reasons you will benefit from attending this training course. Firstly, you will learn what the XML data format is and its associated standards. Secondly, you will see how to integrate it with your own code. Thirdly, you will explore how XML can help you with app design.

Contents of One-Day Training Course	
<p>Target Audience This training course targets component and web developers who need to know what XML is, how to program it and design support for it into their applications and web services</p> <p>Prerequisites Understanding of document and data storage needs, along with experience of Internet programming</p>	<p style="text-align: center;">XML Overview</p> <p>XML is a metalanguage for describing other data languages Representing data with markup Developer resources</p> <p style="text-align: center;">The W3C XML Standards</p> <p>What are in the standards Layering Introduction to each standard</p> <p style="text-align: center;">The XML Information Set</p> <p>Abstract description Information items Strict rules of XML</p> <p style="text-align: center;">XML Serialization Standards</p> <p>The fundamental XML structure is a tree Each node in tree has a name, attributes & can be a parent of other defined nodes</p> <p style="text-align: center;">Namespaces</p> <p>Avoiding tag ambiguity when using multiple XML schemas Unique identifiers Namespace aliases</p> <p style="text-align: center;">XML Structure</p> <p>Defining what is permissible in XML Logical structure of information Valid & well-formed XML</p> <p style="text-align: center;">XML Schemas</p> <p>Describing metadata using XML Defining schemas Type system – simple and complex Lexical space, value space and facets Modularization Schema Usage</p>
	<p style="text-align: center;">W3C XML DOM</p> <p>A Document Object Model for XML Programmatic Navigation Hierarchy of nodes Fundamental & extended DOM interfaces Alternative: treating XML content as a stream - push (SAX) and pull (.NET)</p> <p style="text-align: center;">XPath</p> <p>Identifying sub-sections of the XML tree XSL pattern matching</p> <p style="text-align: center;">XSLT</p> <p>Extensible Style Language Transforms Transformations for display and into other data descriptions</p> <p style="text-align: center;">Additional Standards</p> <p>XML Base XPointer / XLink XML Query XML & Security</p> <p style="text-align: center;">XML-Based Markup Languages</p> <p>Every industry needs to describe different data and hence need a different schema It is not feasible to have a single complete data description</p> <p style="text-align: center;">XML in Application Design</p> <p>How to design for XML Creating a XML-based data format and programmatically loading and saving it</p> <p style="text-align: center;">Project</p> <p>Case study showing the creation of a complex XML schema, its use within an app for data storage and exchange, and web delivery of data to a browser</p>

.NET Entity Framework Core 2.2 Using C# Models, Annotations, Fluent API, Querying, Saving, Advanced, EF Core+Domain Model, Testing, Project

Handling data is one of the most complex aspects of any enterprise application development project. (The fact that Microsoft's .NET has frequently significantly changed its data story proves this!). Previous approaches have not been entirely satisfactory and many devs thought "There's got to be a better way". Enter EF Core.

EF Core is a modern ORM that has rapidly evolved out of code-first Entity Framework 6 for .NET Framework and provides C# app developers with a rich data access capability, in tune with modern development ideas (domain models, testing, SQL & NoSQL, agile, ..).

EF Core uses LINQ as its querying technology to great effect. LINQ is a radically different approach to querying that integrates object development and data access in a cohesive language environment.

This course covers the [latest release](#), 2.2, of EF Core that is available for production use. The goal is to take C#.NET developers along a journey which will end up where they can competently program against a database or other kinds of data sources from within their .NET applications. We also look at what's coming in EF Core 3.0 (e.g. C# 8 support).

Contents of One-Day Training Course	
<p>Target Audience C# / .NET Core developers wishing to create modern apps that need to access databases using the best ORM available for .NET, that is EF Core 2.2.</p> <p>Prerequisites Practical experience of C#, some previous database programming and SQL knowledge (any database).</p> <p>All demos and labs will be using C#.</p>	<p>EF Core Overview Overview of modern data access A powerful ORM (Object-Relational Mapping) for .NET applications Efficient, configurable, improving rapidly How EF Core fits in with rest of .NET</p> <p>Tour Of Features Use an initial sample app to practically demonstrate main feature set of EF Core Role of DbContext, DbSet, DbQuery, .. Data flows, config Error handling</p> <p>Tooling Development / data environment Connection string Distinct SQL RBMS data providers Non-SQL data providers Specialist data providers Additional developer tooling</p> <p>Models POCO – Plain Old C# Objects From these, auto-gen database schema Conventions / Annotations / Fluent API OnModelCreating Ways to influence generated model</p> <p>Querying Applying usual querying syntax: filtering, relationship following, ordering, aggregate Query types</p> <p>Saving SaveChanges / tracking / concurrency Transactions, cascades, keys Disconnected entities</p>
	<p>LINQ And EF Core 2.2 Deeper look at query language Going beyond basics, how to really use LINQ with EF Core Complex queries (e.g. joins)</p> <p>EF Core and .. Migrations Security Database views Stored procedures GIS</p> <p>EF Core and Domain Model Place domain model and data model in separate assemblies Data model references domain model (not the other way around) Domain model does not reference EFCore Prefer use of Fluent API – why? DDD Bounded Context = DbContext Repositories as interface in domain model (no EF Core) and implementation in data model (can be switched out for alternative)</p> <p>Testing This arrangement greatly helps testing Importance of dependency injection Modern testing approach and EF Core Testing database queries</p> <p>Project Developing an end-to-end layered solution consisting of Angular UI, ASP.NET Core REST API, domain model and EF Core 2.2 data model Looking particularly at last of these</p>

PDF Programming

PDF for developers, Graphics, Imaging, Text, Fonts, Forms, Metadata, Security, Navigation, Project

Portable Document Format (PDF) is a standard representation of pages in a document. It is used for document interchange between document producers (word processors, report generators, CAD programs, specialist editing apps) and document consumers (on-screen viewers, printers, digital signature validators, forms engines, etc).

PDF became popular as the native file format for Adobe Acrobat and has been standardized by ISO, initially as ISO 32000-1:2008 (same as Adobe PDF 1.7) & recently ISO published a major update - [ISO/PRF 32000-2](#).

Pretty much every computing device that has an attached display and/or printer can display/print PDF files. The vast majority of modern document editing environments can output to PDF - hence is is an extremely popular shared representation of a publication.

This course is aimed at developers who wish to gain a low-level technical understanding of PDF and how they can use it programmatically from within their own applications. This course provides an excellent foundation for developers who wish to build their own PDF import/export libraries and those who wish to use one of the existing libraries available for this purpose.

Contents of One-Day Training Course	
<p style="text-align: center;">Target Audience</p> <p>Developers wishing to produce, transform and consume PDF files from their own applications.</p> <p style="text-align: center;">Prerequisites</p> <p>Software developers with experience of creating document editing apps, particularly their file formats.</p> <p>Knowledge of graphics programming (e.g. painter's algorithm) is required.</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <p style="text-align: center;">PDF Technical Overview</p> <p>Describing page representations Applying ink to a page Tour of functionality Relationship between PDF and PostScript</p> <p style="text-align: center;">PDF Concepts</p> <p>Coordinate system Painter's algorithm Imaging model Color management Transparency</p> <p style="text-align: center;">Document and Page Layout</p> <p>Overall file structure How each page is represented Contents of a page Common data</p> <p style="text-align: center;">Text And Fonts</p> <p>Typography in PDF Glyphs Fonts Ways to represent text Text objects</p> <p style="text-align: center;">Graphics</p> <p>PDF supports many 2D diagramming constructs, from line to Bézier curves to various graphical shapes State information Use of 3D Path construction</p> <p style="text-align: center;">Images</p> <p>Bitmaps and their use in PDF Supported imaging formats Rendering images on the page</p> </div> <div style="width: 48%;"> <p style="text-align: center;">Color and Transparency</p> <p>Precisely specific color requirements CIE Masking Clipping</p> <p style="text-align: center;">Forms</p> <p>Forms options in PDF Designing forms Accessing form fields after completion Changes in forms for ISO/PRF 32000-2</p> <p style="text-align: center;">Metadata</p> <p>Attaching additional metadata to PDF Metadata formats Extensibility</p> <p style="text-align: center;">Digital Signatures</p> <p>Signing what uses can see Digitally signing a PDF file Verifying a digital signature Cryptographic options (algorithms etc.)</p> <p style="text-align: center;">Navigation</p> <p>Allowing users to easily navigate to pre-determined destinations within a document (hierarchy & thumbnails) Document outline Flexible inter-page navigation</p> <p style="text-align: center;">Multimedia</p> <p>Embedding video, audio and 3D in a document Formats supported Enabling media playing</p> <p style="text-align: center;">Project</p> <p>Write a project to programmatically create a PDF document</p> </div> </div>

Technology Faculty / Networking School

Protocol

- Fundamentals of TCP/IP Networking
- QUIC And HTTP/3 Protocols

RTC

- WebRTC: Protocol & Programming

API

- REST APIs – Designing, Specifying (OpenAPI) and Generating (OpenAPI Generator)
- ASP.NET Core 2.2 REST API Development

Orchestrator

- Kubernetes

Fundamentals Of TCP/IP Networking

Layering, Networking Architecture, Addressing, IPv4&IPv6, TCP, UDP, DNS, DHCP, HTTP, Security

TCP/IP is the dominant networking protocol suite and all involved in delivering modern technology solutions need a clear understanding of how it fundamentally works. Developers, DevOps and system admins, along with many specialists (such as those involved in database and security) could all benefit from attending this course as it will clarify the fundamentals of networking with the TCP/IP suite.

Networking touches on development, security, network traffic, identity, messaging, data transfer, resilience, middleware and lots more.

We examine the entire set of protocols in the TCP/IP suite. Even though the suite name mentions just two protocols, it is actually a much larger collection of protocols, with important protocols above and below the TCP and IPv4/IPv6 protocols.

In any modern OS, its networking stack is a large segment of the codebase and exposed API. By investigating what is actually happening between one app sending & another app receiving a message (by examining net traffic using appropriate monitoring tools), we can expand our understanding significantly.

Contents of One-Day Training Course	
<p>Target Audience Everyone with a technical background who is interested in how computer networks actually work.</p> <p>Prerequisites Experience of working on software projects, including development, deployment and ongoing service provision.</p> <p>No previous network programming or infrastructure experience is required, though any such knowledge would be beneficial.</p>	<p style="text-align: center;">Layers, Protocols and Devices</p> <p>Overview of networking participants Networking big picture Organization of the Internet Protocol development process (RFCs)</p> <p style="text-align: center;">Use of Wireshark</p> <p>Eavesdropping on the network Extremely useful tool for low-level monitoring of protocol traffic</p> <p style="text-align: center;">TCP/IP Family Of Protocols</p> <p>Layering of protocols Distribution of responsibilities Packet structure (encapsulation etc.)</p> <p style="text-align: center;">Data Link and Network Links</p> <p>How octets are physically communicated technologies used at data and network Ethernet</p> <p style="text-align: center;">Addressing</p> <p>Approaches to network addressing Subnetting ARP Addressing for IPv6</p> <p style="text-align: center;">IPv4 & IPv6</p> <p>Protocol exchanges Packet structures MTU Internet Control Message Protocol (ping)</p> <p style="text-align: center;">TCP</p> <p>Enhanced services Headers Windowing SYN packet and triple-handshake Protocol exchanges</p> <p style="text-align: center;">UDP</p> <p>Comparison with TCP “Best effort” datagram service Protocol exchanges</p> <p style="text-align: center;">DNS and DHCP</p> <p>DNS architecture DNS headers Record types (e.g. SRV records) DHCP architecture</p> <p style="text-align: center;">Internet Building Blocks</p> <p>Routing protocols Border Gateway Protocol (BGP) QoS AS</p> <p style="text-align: center;">HTTP 1.x & 2</p> <p>HTTP basic messaging patterns Using Fiddler to examine HTTP traffic Message chunking Multiple concurrent channels in HTTP/2</p> <p style="text-align: center;">SMTP, POP and MIME</p> <p>Email protocols family Message stores Representing emails</p> <p style="text-align: center;">Security And Networking</p> <p>Firewalls NATs IPSec SSL/TLS</p> <p style="text-align: center;">OS Networking Stacks</p> <p>Windows Linux Exposed C APIs (sockets etc.) Networking APIs inside managed runtimes</p>

QUIC And HTTP/3 Protocols

Transport/App Protocol, Connection, Packet, Frame, TLS 1.3, Multiplexing, Flow-Control, Implementation

QUIC and HTTP/3 are respectively the much anticipated next-generation transport protocol and next generation application protocol. HTTP/3 runs on top of QUIC. Both are currently undergoing standardization through the IETF.

Both protocols are being designed in unison to work extremely well together to achieve an elevated level of performance and security. They offer a number of very interesting new features and continue the work we first see in HTTP/2 of adding multiplexing for HTTP connections.

“The QUIC transport protocol incorporates stream multiplexing and per-stream flow control, .. By providing reliability at the stream level and congestion control across the entire connection, it has the capability to improve the performance of HTTP compared to a TCP mapping. QUIC also incorporates TLS 1.3 at the transport layer, offering comparable security to running TLS over TCP, but with improved connection setup latency. [\[link\]](#)”

At the end of this course, attendees will understand how both protocols work and why they are important.

Contents of One-Day Training Course	
<p>Target Audience Networking professionals and senior software engineers who require a deeper understanding of these new protocols that will play a very significant role in the future of the web and the Internet</p> <p>Prerequisites Programming experience in any low-level language. Attendees will develop server and client-side implementations of both protocols as part of the labs.</p> <p>Good all-round networking knowledge; attendance at our <i>Fundamentals Of TCP/IP Networking</i> course or similar experience.</p>	<p style="text-align: center;">Next Gen Protocols</p> <p>What are we trying to achieve? Can we not reach these goals with TCP and HTTP/1.1 or HTTP/2? What do QUIC and HTTP/3 offer?</p> <p style="text-align: center;">QUIC Overview</p> <p>General tour of how QUIC works Based on UDP (a foundation that is already supported everywhere) QUIC plays role of TCP in protocol stack Message flows Connection set up and tear down Intro to security Available QUIC implementations Extensions</p> <p style="text-align: center;">Connections</p> <p>Reasons for low-latency connection setup What impact this has? Connection migration Error correction Packet layout</p> <p style="text-align: center;">QUIC Streams & Multiplexing</p> <p>Unidirectional and bidirectional streams How multiple streams are multiplexed onto a single connection</p> <p style="text-align: center;">Flow Control</p> <p>Connection flow control Stream flow control</p> <p style="text-align: center;">QUIC And Security</p> <p>Modern TLS 1.3 built into QUIC Security is not an add-on option Review of security architecture Responding to NAT rebinding</p>
	<p style="text-align: center;">Frames</p> <p>PADDING, RST_STREAM, [CONN APP] _CLOSE, MAX_DATA, MAX_STREAM_DATA, MAX_STREAM_ID, PING, BLOCKED, STREAM_BLOCKED, STREAM_ID_BLOCKED, NEW_CONN_ID, RETIRE_CONNECTION_ID, STOP_SENDING, ACK, PATH [CHALLENGE] RESPONSE], NEW_TOKEN, STREAM, CRYPTO</p> <p style="text-align: center;">HTTP/3 Overview</p> <p>Overall architecture HTTP/3 endpoints Options for discovery Types of streams (control, push, reserved) Available HTTP/3 implementations</p> <p style="text-align: center;">HTTP Framing</p> <p>DATA, HEADER, PRIORITY Settings Framing architecture</p> <p style="text-align: center;">HTTP Message Exchanges</p> <p>Message has one HEADERS frame and a number of DATA frames and optionally a concluding HEADERS frame Message flows</p> <p style="text-align: center;">Connection Management</p> <p>Cancellation, Compression, Prioritization Server Push Error management</p> <p style="text-align: center;">Impact On Application Design</p> <p>How these new protocols will influence application design (esp. multiplexing)</p> <p style="text-align: center;">Project</p> <p>Create simple implementations of QUIC and HTTP/3 and see how application code could benefit</p>

WebRTC: Protocol And Programming

Protocol: Establishment, Codecs, SDP, Security, NAT Programming: API Tour, PeerConnection, Project

Real time communication (RTC) is intended for person-to-person communication in real time. WebRTC is built into modern standards-compliant web browsers, mobile devices and IoT devices. WebRTC supports RTC without needing plug-ins. WebRTC enjoys very wide-spread vendor support. Forms of communication supported include audio, video and data exchange (e.g. application data or docs).

WebRTC is the basis for RTC within many interactive apps that are widely used. What is less known is that WebRTC is a programming platform, and apps running

in the browser & on mobile/IoT devices can interact with it. App developers striving to add unique features to their apps would be well advised to consider WebRTC, as once an understanding of how it works has been gained, it is not that time consuming to program. The results are quite powerful and appreciated by users.

This course starts with an introduction to WebRTC, then looks at the protocol in detail, and then looks at how to program it. Topics covered include codec selection, SDP, addressing, security, identity, specialist services, data exchange and lots more.

Contents of One-Day Training Course	
<p>Target Audience Networking professionals and senior software engineers who require a deeper understanding WebRTC – both the protocol and how to program with it.</p> <p>Prerequisites Programming experience in JavaScript and/or TypeScript.</p> <p>Good all-round foundational networking knowledge; attendance at our <i>Fundamentals Of TCP/IP Networking</i> course or similar experience.</p>	<p style="text-align: center;">WebRTC Intro</p> <p>What is is trying to achieve Widespread adoption A community A set of standard protocols A set of standard APIs - no plugins needed An API for programming An open source project</p> <p style="text-align: center;">Architecture</p> <p>Major components for audio, video, data How transport works Identity JavaScript and C++ APIs Layering</p> <p style="text-align: center;">Foundational Technologies</p> <p>RTP – Realtime Transport Protocol SDP – Session Description Protocol Interaction with SIP (VoIP) Future: QUIC and WebRTC?</p> <p style="text-align: center;">WebRTC Protocol</p> <p>Connection Establishment Message flows Codec selection Buffering networking</p> <p style="text-align: center;">WebRTC and SPIN/ICE</p> <p>Issues with some networks (NAT) How to use NAT with WebRTC Forwarding</p> <p style="text-align: center;">WebRTC And Security</p> <p>How to identify participants Securing comms links Regulation: comms, GDPR, police, ..</p>
	<p style="text-align: center;">W3C Standards</p> <p>W3C is working on many WebRTC stds API written in WebIDL, so are familiar to any JavaScript/TypeScript app dev Review of important ones, e.g.: WebRTC 1.0: RTC Between Browsers and Media Capture and Streams</p> <p style="text-align: center;">Connection Programming</p> <p>How to programmatically set up connections with remote parties Configuration Security Transport RTCPeerConnection</p> <p style="text-align: center;">Media API</p> <p>Passing track info to remote parties RTCRtpSender RTCRtpReceiver</p> <p style="text-align: center;">MediaStream API</p> <p>MediaStreamTrack interface MediaStream interface</p> <p style="text-align: center;">Data API</p> <p>Exchanging app data over WebRTC links RTCDataChannel</p> <p style="text-align: center;">Application Issues</p> <p>Developer environment setup for WebRTC Debugging Error handling with WebRTC WebRTC as part of a large application suite</p> <p style="text-align: center;">Project</p> <p>Detailed project exploring how modern UI apps can be shared via WebRTC’s data channel</p>

REST APIs – Designing, Specifying (OpenAPI) and Generating (OpenAPI-Generator)

Design API, Write Spec, Autogen client/server code

A REST API is the most practical way of connecting clients and servers in a distributed heterogeneous world.

We start by discovering what is involved in creating high-quality REST APIs and how to best go about designing them, in a contract-first manner. Later in the course we also explore advanced API design topics.

Then we need to specify the API. [OpenAPI](#) is essentially a well-written standard for a messaging schema, describing the messages and their headers and body contents. “The OpenAPI Initiative (OAI) was

created by a consortium of forward-looking industry experts who recognize the immense value of standardizing on how REST APIs are described.”

Once we have written our OpenAPI schema, the final task is to generate code to produce and consume messages that comply with that schema. For this, we use the OpenAPI Generator open source project. “[OpenAPI Generator](#) allows generation of API client libraries (SDK generation), server stubs, documentation and configuration automatically given an OpenAPI Spec”. It supports dozens of languages/frameworks.

Contents of One-Day Training Course	
<p>Target Audience Software architects and senior developers tasked with efficiently creating and/or consuming REST APIs</p> <p>Prerequisites Sound understanding of at least one client programming environment and one server environment (from the OpenAPI Generator list of supported languages)</p> <p>Some previous experience of network programming would be useful.</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <p>Need for REST APIs Representational state transfer Principles of REST architecture Reliably connecting distributed pieces of functionality</p> <p>Review of JSON and YAML How we represent information on the wire JavaScript Object Notation (advanced) “YAML Ain't Markup Language” What enhancements YAML brings</p> <p>Tasks To Build REST APIs Need to consider design of API Need to specify API Need to generate client library to call API Need to integrate client library with UX Need to generate server stubs to host API Need to integrate server stubs with DB, etc Need to comprehensively test everything</p> <p>Designing a REST API What to take into consideration Functionality discoverability Identifying resources and selecting verbs Integration with HTTP methods</p> <p>OpenAPI Overview OpenAPI Initiative (Linux Foundation) The OpenAPI specification (https://github.com/OAI/OpenAPI-Specification/tree/master/versions)</p> <p>OpenAPI Usage Contract Driven API (create contract first) Handling paths Defining components Creating API documentation</p> </div> <div style="width: 48%;"> <p>OpenAPI In Detail Detailed look at important constructs in spec file and how best to use them Boilerplate layout and sections</p> <p>OpenAPI Generator Overview A modern open source community API Generator toolset for building REST APIs Compliant with OpenAPI Spec v3 (fork of Swagger Codegen) A set of useful tools for code generation for client libraries and server stubs e.g.: OpenAPI Generator, from version v.3.3.2 onward, generates Angular 7 code</p> <p>OpenAPI Generator Features Setup and tooling Integrate with build Useful command line arguments List of options and their uses Exploring OpenAPI Generator source tree</p> <p>Practical REST API Creation Security (avoiding code injection) Testing Working with CI/CD Error handling Documentation generation</p> <p>Style Guide How best to structure OpenAPI files Naming conventions Preparing for future API version evolution</p> <p>Project Building an end-to-end application involving all steps in REST API design, specification, implementation and usage</p> </div> </div>

ASP.NET Core 2.2 REST API Development

Foundations, Creating REST APIs, Controllers, Domain, OpenAPI (Swagger), Testing, Architecture

ASP.NET Core 2.2 is the modern approach to developing web UI and REST API applications using .NET Core 2.2. This exciting technology richly supports OpenAPI-based REST development, test-driven development, domain models, custom URL routing schemes, flexible viewing and separation of concerns.

This course covers in-depth the foundations of ASP.NET Core 2.2 and how to use it to create REST APIs. Important: this course does not cover ASP.NET MVC user interface development (e.g. use of Razor).

ASP.NET Core 2.2 comes with a range of features to build REST APIs: both the HTTPS interface and what is behind it (e.g. domain model, database access via EFCore)+[enhanced OpenAPI](#) support.

Modern REST API developers use C#, .NET Core and ASP.NET Core 2.2 to build server-side implementations, then use OpenAPI (formerly called Swagger) to describe an API and tools such as [OpenAPI-Generator](#) to auto-generate client-side stubs (e.g. for Angular 7 UI in the browser).

Contents of One-Day Training Course	
<p>Target Audience REST API developers wishing to learn about the latest approach to creating REST APIs using .NET Core 2.2 technologies.</p> <p>Prerequisites Attendees are expected to be experienced C# developers with some understanding of the HTTP protocol.</p> <p>Attendance at our <i>.NET Core 2.2 CLR Programming</i> course or equivalent experience.</p> <p>No previous experience of ASP.NET Core required, as this course explores it from the foundations upwards.</p>	<p>Overview of ASP.NET Core 2.2 “ASP.NET Core is an open-source and cross-platform framework for building modern cloud based internet connected applications, such as web apps, IoT apps and mobile backends.” [link] Goals for ASP.NET Core Advantages and challenges Web UI and Web API unified</p> <p>Review Of REST API Pipeline Developer setup required (Visual Studio) What components are needed to deliver REST solution via ASP.NET Core Quick Tour of ASP.NET Core Focus in on server-side implementation HTTP request pipeline New features of ASP.NET Core 2.2 What’s coming in ASP.NET Core 3.0</p> <p>First REST API App Walkthrough of a simple app introducing all the major features Solution layout Project template in Visual Studio 2017 ASP.NET Core 2.2 can run on both .NET Core and .NET Framework, but ASP.NET Core 3.0 will only run on .NET Core</p> <p>Important Concepts Attributes (Produces, Route, ..) Use of async HTTP methods (POST, GET, ..) Accessing parameter information Conventions Analyzers</p>
	<p>Controllers And Actions Plays coordination role ControllerBase ApiController Action results Different results for different needs Parameterizing actions Formatting results</p> <p>Domain & Data Models Domain-driven design Tour of DDD concepts (e.g. repository) In ASP.NET Core context, model is a domain model (not a data model) Domain model define repository, which data model implements (e.g. EF Core 2.2)</p> <p>Architecture Designing ASP.NET Core solutions Common design problems</p> <p>OpenAPI (Swagger) Use of add-on OpenAPI capabilities with .NET Core 2.2 to create portable representation of API and documentation</p> <p>Security Web application security Using ASP.NET Core application services Various threats</p> <p>Testing Approaches to testing web apps Unit testing and Mocking Debugging tools (e.g. Fiddler, Postman)</p> <p>Project A demonstration of how to use ASP.NET Core for more substantial workloads</p>

Kubernetes

How Everything Works Together, etcd, API Server, Scheduler, Controller, Kubelet, CRI, CNI, CSI, Project

Once you go beyond running a few containers on a single machine, you need to think about how to manage multiple nodes and make them work together at scale and that is where Kubernetes comes in. Kubernetes is the leading orchestration system for container clusters.

At first glance Kubernetes is confusing, because it involves multiple components running demanding / varying workloads on different nodes in a network. It needs to offer resilience in the face of all kinds of failure, very tight security, efficiency, timely delivery of application updates and many more features.

However, after we first review the big picture for Kubernetes, when we examine each of its components in turn, we see they each performs a focused task (storage config, change data, schedule a container, start a container). Each component on its own is relatively simple, and so we gradually build up a deeper understanding of how Kubernetes actually works.

This intensive course brings engineering professionals who already know about containers and microservices up to speed with the world's leading open source container orchestration platform.

Contents of One-Day Training Course	
<p>Target Audience Developers, devops personnel and system administrators wishing to provision large clusters of containers using Kubernetes.</p> <p>Prerequisites Attendees to this course must have already attended our <i>Microservices and Containers</i> course or have similar experience.</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <p>Big Picture Concepts What is a container cluster, a master node, a worker node, schedulers, controllers, a container, a pod, kubelet, etc. How everything works together</p> <p>Setting up a Container Cluster Set up Kubernetes so that master talks to kubelet on each node, which manages a set of pods on that node, and each pod contains one or more containers</p> <p style="text-align: center;">etcd</p> <p><u>etcd</u> is a highly reliable distributed database engine that optimally stores and allows observing of config data Like what you have in /etc config files but managed by daemons (hence name) RAFT Protocol</p> <p style="text-align: center;">API Server</p> <p>Offers a REST API to remote clients to configure and monitor a cluster Clients could be command-line interface (CLI) tools or admin consoles or scripts Writes settings into etcd Let's examine APIs on offer</p> <p style="text-align: center;">Kubectl</p> <p>A CLI that talks to API Server Main route for devops people to directly interact with API Server</p> <p style="text-align: center;">Scheduler</p> <p>Watches etcd for changes to configuration (e.g. via API Server) and makes scheduling decisions, writes these to etcd Scheduling algorithms and settings</p> </div> <div style="width: 48%;"> <p style="text-align: center;">Controllers</p> <p>Multiple controller types are supplied to manage node lifetimes, replication, endpoints and account details Cloud controllers</p> <p style="text-align: center;">Kubelet</p> <p>Makes a worker node part of the cluster Watches settings in etcd for this node When change detected in etcd config, makes it so in what runs on worker node</p> <p style="text-align: center;">Container Runtime Interface</p> <p>CRI is an interface between kubelet and actual container runtime Popular options: CRI-o or cri-containerd Containers are usually run via OCI's runc</p> <p style="text-align: center;">CNI</p> <p>Container Network Interface (CNI) is a spec and implementation for networking functionality in the world of containers</p> <p style="text-align: center;">CSI</p> <p>Container Storage Interface (CSI) is a plugin architecture for storage volumes</p> <p style="text-align: center;">Labels and selectors</p> <p>Attaching key/value pairs to objects Selecting many objects based on labels</p> <p style="text-align: center;">Kubernetes Source Tree</p> <p>To build up deeper knowledge, we explore the source trees for Kubernetes and related projects, such as etcd (all written in Go)</p> <p style="text-align: center;">Project</p> <p>Deploying a real world application to a Kubernetes cluster</p> </div> </div>

Technology Faculty / UX School

Client UI

- Fundamentals Of User Interaction
- HTML5.3 And DOM5.3 Using TypeScript
- CSS – Cascading Style Sheets
- Angular 7.2 Fundamentals
- Advanced Angular 7.2
- Angular Material 7
- PWA using Angular 7.2 Service-Worker
- Angular 7.2 Internationali[z|s]ation
- NgRx
- Ionic 4
- Web Components/Angular Elements/Microfrontends

Server UI

- Node.js 11 PUG And Express UI Programming Using TypeScript

Fundamentals Of User Interaction

User Centered Design, Usability Principles, Design Process, Identifying the User, Tools, Usability Tests

Usability is an intangible software feature - more noticeable by its absence than its presence in an application. This course helps you design it into your products. User interaction design is a branch of software engineering – and needs to be approached in an organized manner. We need to get to know the user, to discover the user interaction requirements, to build the app, and then to test that we have done it correctly. This is both similar and different to how we would code up an algorithm or implement a database schema. Every workitem on a software dev team’s project plan should be examined to see if it enhances usability of the app -

if not, it needs to be reconsidered. There is a need to collect as much information as possible from the user – via product demonstrations, usability testing, and after launch, analyzing the helpdesk queries. Just like advertising, 50% of software features are never used – you will need to find out which, and more importantly why – perhaps the unused features are too difficult, perhaps the user does not know they exist (inaccessible), or perhaps they are simply not needed. Designers and developers will benefit from attending this course by gaining an increasing awareness of the processes used to create incredible user-friendly apps.

Contents of One-Day Training Course	
<p>Target Audience This course is aimed at software designers and senior developers who need to create highly intuitive user interfaces</p> <p>Prerequisites Experience of programming graphical user interfaces is needed along with an appreciation of usability issues</p>	<p style="text-align: center;">User-Centric Design</p> <p>Placing the user at the center of the software design process Every developer needs to think of the user experience – not just those who directly create the user interface</p> <p style="text-align: center;">Usability Design Principles</p> <p>Terminology and metaphors Consistency Task invocation and navigation Functionality discovery Continuous feedback Controllability Selection and activation Validation of user input</p> <p style="text-align: center;">Concepts</p> <p>User – application communication Norman’s Model From CLI to GUI to CBI Command-Based Interface used from multiple sources (GUI, wizards, macros, Automation, CBT/Help, undo/redo)</p> <p style="text-align: center;">Usability Design Process</p> <p>Identifying the user Task analysis Story-boarding Prototyping Usability testing An iterative cycle</p> <p style="text-align: center;">Direct Manipulation</p> <p>Direct interaction In-place editing Drag and drop</p>
	<p style="text-align: center;">The look of the application</p> <p>Font, color, etc. Use of certain controls Layout of forms Task-centric design Icons and cursors Benefits of a “minimalist” interface</p> <p style="text-align: center;">Types of GUI</p> <p>OOUI MUI Handling complexity Handling large data sets</p> <p style="text-align: center;">Usability for different apps</p> <p>Database (transactions, locking, tables) Graphics (dirty bit, perspective, selection) Components (object display, activation) Networking (node selection, time delays)</p> <p style="text-align: center;">User Assistance Tools</p> <p>Help system/online mentoring wizard Context sensitive computer based training</p> <p style="text-align: center;">Error Avoidance</p> <p>Why errors happen Engineering user errors out of applications Users never make errors-only designers do</p> <p style="text-align: center;">Usability Testing</p> <p>The “five minute” user test Collecting information from users User interaction engineering Making it part of the development project and what internal doc/models are needed.</p> <p style="text-align: center;">Usability Engineering Project</p> <p>Complete walk through of how to design user interaction for a complex project</p>

HTML5.3 and DOM5.3 Using TypeScript

Text, sections, forms, tables, the DOM, events, the canvas and lots more inside modern browsers

The web platform is undergoing a rapid pace of innovation and its primary standards, HTML and its DOM, are central to how modern browsers work. The HTML markup pre-populates the tree and then the DOM can be used to dynamically edit the tree's content. After a quick review of the fundamentals of HTML and the DOM, this course explores many of the new and expanded capabilities offered by HTML5.3 & DOM5.3. These latest specs are well supported across modern web browsers from different vendors and offer an increasing range of rich functionality that significantly improve over the simple markup apps of the past.

Among the many enhancements explored in this course are the shadow DOM, the HTML canvas, better table functionality, improved event handling architecture, the idea of an HTML application with a manifest, better navigation and plenty more.

In the past developers using higher level frameworks were sometimes isolated from directly accessing HTML and the DOM, but there has been a trend in more recent frameworks to re-introduce developers to direct low-level access to the full power of HTML and the DOM - we will investigate the real benefits of this approach.

Contents of One-Day Training Course	
<p>Target Audience Experienced developers who already know the basics of HTML (HEAD, BODY, P, DIV, etc.) and now wish to get up to speed with the latest standards for markup and programmatically editing HTML content.</p> <p>Prerequisites For the DOM5.3 part of the course, all demos and lab exercises will be in TypeScript, so attendees need to know that language.</p>	<p>HTML5.3 overview WHATWG (whatwg.org) and W3C "Living standard" vs. numbered spec Central role of HTML's 1200-page spec in standardizing the web platform Collection of specs Metadata, elements, attributes, encoding</p> <p>Tour Of Elements Exploring common elements, some we are familiar with, some new to HTML5 Page lifecycle / Uses for URIs The execution context(s) Content models</p> <p>Resources, URI and IRI Identifying resources Internationalization with IRI</p> <p>DOM Overview Correspondence between HTML markup elements and DOM tree elements (they are similar, but not exactly the same) How to interact with a tree of elements Options for parsing / serializing DOM Role of WebIDL in defining web APIs</p> <p>Sections & Grouping Sections - body, article, section, aside, h1..6, hgroup, header, footer, address Grouping - p, hr, pre, blockquote, ol, ul, menu, li, dl, dt, fig[ure caption], main, div</p> <p>Text a, em, strong, small, s, cite, q, dfn, abbr, ruby, rt, rp, data, time, code, var, samp, sub/sup, kbd, i/b/u, mark, bdi,bdo,span,br, wbr</p> <p>Modern Tables Evolution of HTML tables - table, caption, colgroup, col, tbody, thead, tfoot, tr, td, th</p> <p>Forms Submitting to the server form, label, input, button, select, datalist, optgroup, option, textarea, output, progress, meter, fieldset, legend</p> <p>HTML Templates Chunks of reusable markup Defining and instantiating</p> <p>Event Handling Event bubbling Defining and using events Popular DOM events Event listeners</p> <p>Shadow DOM Isolating elements for web components attachshadow() and shadow root</p> <p>Specialist HTML iframe, dialog, summary, ins/del Session history Location</p> <p>HTML Canvas Drawing context 2D immediate mode graphics Other graphics and media handling OffscreenCanvas</p> <p>HTML Applications Application Cache and manifest Offline web applications Linking and link types The Markdown DSL</p>

CSS – Cascading Style Sheets

Handling Styling, Selectors, Descriptions, Specificity, Priority, Pseudo-, CSS for ..., SCSS

CSS is not a markup language (unlike HTML5) used to represent content. CSS is not a programming language (unlike JavaScript/TypeScript), used to dynamically change content at run time. Rather, CSS is a styling language used to concisely represent styling information for web content.

be applied to elements. An important consideration is to maintain the same “look & feel” across multiple pages, Thus a site-wide “house style” is often defined. Also CSS allows the same content to be optimally rendered on output devices with differing capabilities (size, resolution, interactivity) and suitably for people with differing interaction needs.

CSS is used to declaratively describe the initial hierarchical set of styles that are to be applied to HTML elements which later may be manipulated by TypeScript/JavaScript code. CSS offers considerable flexibility in how styles are specified and how they can

This course looks at the role of CSS in the world of modern web development, provides a detailed tour of CSS features and explores how you can best deploy it for styling your own web solutions.

Contents of One-Day Training Course	
<p>Target Audience All developers wishing to gain a comprehensive understanding of the least known of the three pillars of web development (the other two being HTML and JavaScript/TypeScript).</p> <p>Prerequisites Knowledge of HTML is required along with some graphics background.</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <p>CSS Overview</p> <p>Growing set of styling standards Defining CSS rules Style document layout Declaration block Internal CSS using <style> External CSS using <link> and .css file Could be used to “render” speech, but mostly used for visual representations</p> <p>CSS 1 / CSS 2 / CSS 3 / CSS 4</p> <p>Evolution of standards (CSS modules) Expanding set of layered standards What is [well less well] supported</p> <p>Selector</p> <p>Identifying categories of markup elements Element type Id Class Element hierarchy</p> <p>Declaration Blocks</p> <p>Collection of declarations in blocks Property and value (separated by colon) Representing values using custom syntax suitable for each value type</p> <p>Inheritance</p> <p>Importance of style ancestry Styling descendants</p> <p>Pseudo-class/pseudo-element</p> <p>Additional ways to select elements Styling after particular actions (e.g. click)</p> <p>Combining selectors</p> <p>Importance of selector order Priority</p> </div> <div style="width: 48%;"> <p>CSS for Layout</p> <p>Defining where elements appear on page Replacement for HTML tables (sometimes) Multi-columns and flexible layout</p> <p>CSS for Color</p> <p>Defining color scheme for elements Element color Background Ways to define color (e.g. gradient)</p> <p>CSS for Font</p> <p>Selecting font types for text Styling such fonts (italics, bold) Sizing fonts</p> <p>CSS for Spacing</p> <p>Margin Border Internal text spacing</p> <p>CSS For Text</p> <p>Alignment Paged media Translations</p> <p>CSS For Content Handling</p> <p>Web animation Transitions Generated content</p> <p>SCSS</p> <p>A CSS pre-processor that simplifies CSS syntax and extends its capabilities “allows you to use variables, nested rules, mixins, inline imports and more” with CSS</p> <p>Project</p> <p>How to use CSS in a large enterprise solution to provide flexible styling</p> </div> </div>

Angular 7.2 Fundamentals

Tour, CLI, Building Components, Directives, Databinding, HTTP Client, Forms, Bootstrapping

Angular 7.2 is the most advanced web client framework available for production use. It provides a very solid foundation for development of modern web applications and is increasingly being selected by start-ups, cloud vendors and global enterprises for their large scale solutions with demanding needs and tight development time frames.

Angular is a vibrant open source project and is built by a large team from Google and elsewhere. They work to further evolve the framework and they regularly add interesting enhancements. This course covers the latest

Angular 7.2 that can be used to build web desktop, web mobile, native desktop, native mobile and even server applications (as it has plenty of non-UI functionality).

Angular is a big project, best approached in stages by developers wishing to start building apps. Before this course app developers should learn TypeScript (Angular itself is written in TypeScript, as are most Angular apps). Then app developers should attend this course as their first contact with Angular, optionally followed by our *Advanced Angular 7.2* course, which looks in more detail at some specialist Angular topics.

Contents of One-Day Training Course	
<p>Target Audience Developers seeking to quickly get up to speed with the best web framework in the world.</p> <p>Prerequisites Developers experienced with the TypeScript language and web programming in general.</p> <p>No previous Angular experience required.</p> <p>All demos and lab exercises will be in TypeScript.</p>	<p style="text-align: center;">Angular Framework Tour</p> <p>Collection of packages that work together to deliver a wonderful web framework Overview of how it works Introduction to each major module Many parts to an Angular app</p> <p style="text-align: center;">Angular CLI 7.2</p> <p>Command line interface to creating, building, serving and testing Angular apps Angular CLI automates the creation of a good boilerplate source tree for your app that you can later enhance What you might like to customize (e.g. versions in package.json) CLI Prompts (new in Schematics in v7)</p> <p style="text-align: center;">Building Components</p> <p>Exploring how we build components Event handling – firing and listening Input and output properties Metadata for components</p> <p style="text-align: center;">Angular Template Syntax</p> <p>Enhancing HTML syntax with custom directives and expressions Which HTML concepts not permitted Interpolation Expression syntax Attribute directives ngModel</p> <p style="text-align: center;">Structural Directives</p> <p>ngFor / ngIf ngSwitch Microsyntax ng-template / ng-container</p> <p style="text-align: center;">Databinding in-depth</p> <p>Event, Property, Attribute, Class, Style, Two-Way</p> <p style="text-align: center;">HTTP Client</p> <p>How to use the various HTTP request types Role of in-memory-web-api for testing Use of services in Angular app architecture Asynchronous stream of events (objects) delivered to your components (RxJS)</p> <p style="text-align: center;">Animation</p> <p>Transitions States & Triggers</p> <p style="text-align: center;">Advanced Components</p> <p>Styling for Angular components Lifecycle hooks Pipes Deeper look at how you build services</p> <p style="text-align: center;">Introduction to Forms</p> <p>Template-driven forms Error handling Change tracking Structuring form handling code Reactive forms</p> <p style="text-align: center;">Validation</p> <p>Validating forms input Correctly displaying error information Built-in and custom validators Role of CSS in reflecting control status</p> <p style="text-align: center;">Bootstrapping</p> <p>How an Angular app bootstraps Intro to how rendering works Use of platform-browser</p>

Advanced Angular 7.2

Routing, Libraries, Rendering, Platforms, DI, NgModule, Universal, DevKit, Schematics

It is in the more advanced capabilities of Angular that we see it distancing itself from simpler frameworks and results in it being more and more selected for large-scale important projects that needs a stable, powerful framework as the basis for long-term innovation.

Angular offers a well thought out architecture, its configurable platforms means alternative rendering approaches may be supported (e.g. from a web worker), dependency injection means components can be swapped in and out over time and a wonderful routing engine provides browser-side navigation for views.

Developers already familiar with using Angular to build UI apps will find this advanced course of particular interest as it comprehensively explores how to leverage the rich feature set of the Angular Framework to build more innovative applications that distinguish themselves from the competition in rendering performance and the flexibility of what they offer users.

Attendees will also benefit from this course's coverage of more specialist Angular topics, such as NgModules, Angular Universal and Angular Dev Kit (including Schematics).

Contents of One-Day Training Course	
<p>Target Audience Angular and TypeScript developers wishing to explore more deeply how to leverage the more advanced capabilities of the Angular Framework.</p> <p>Prerequisites Attendees should have attended our “<i>Angular 7.2 Fundamentals</i>” course or have equivalent experience.</p> <p>All demos & labs are in TypeScript.</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <p>Angular System Programming Closer look at low level details of how Angular apps actually run Use of zones in Angular</p> <p>Dependency Injection Excellent for testing Hierarchical DI built into Angular Provides greater flexibility in how a well structured app can evolve into future</p> <p>Angular Router Fundamentals Browser-based editing of address bar URI Setting up routes using the Router Service Router outlet Catch-all entry</p> <p>Advanced Angular Router State management Handling routing events Use of guards</p> <p>Lazy Loading For large applications, loading everything at startup is costly How can we postpone loading some functionality until/unless it is needed</p> <p>Advanced Angular CLI What is a CLI workspace? In-depth look at workspace layout ng add Build tooling Multiple projects</p> <p>Building Angular 7.2 Libraries ng-packagr ng g library Sub-dividing large projects into libraries</p> </div> <div style="width: 48%;"> <p>NgModule Detailed look at what NgModules are and how they are used entryComponents vs. bootstrap imports & exports</p> <p>Platforms & Rendering Rendering pipeline in v7 (Render 2) and upcoming v8 (Render 3/“Ivy”) Customizing rendering Logging rendering information Role of platforms Web workers:[Client Service]MsgBroker</p> <p>Schematics Idea behind schematics Creating for your own projects Examining sample schematics projects</p> <p>Angular Language Service “The Angular Language Service is a way to get completions, errors, hints, and navigation inside your Angular templates”</p> <p>Angular Universal Running Angular on the server Good for search engines (SEO) Good for fast first load of page in browser</p> <p>Angular 7.2 DevKit Set of dev tools and libraries for ecosystem Schematics - “generators that transform an existing filesystem” -used by Angular CLI Core, Build & Architect packages</p> <p>App Architecture Domain model for Angular apps</p> </div> </div>

Angular Material 7

Material Design, Angular Material Tour, Navigation, Layout, Popups, Datatable, Flex Layout, CDK, Project

After developers get up to speed with the fundamentals of Angular and TypeScript programming, their attention turns from the basics of creating web UIs to more substantial concerns – such as the need to create visually appealing, logically organized and easily navigable applications that responsively react to the devices used by end-users.

Such modern web applications need to be competitive in the marketplace and for this consistent styling, layout and component architecture is needed – hence the need for material design (<https://material.io>) and its imple-

mentation for Angular 7.2 - Angular Material 7.

Hundreds of millions of users worldwide see material design everyday when using Google Search, Gmail, Youtube and Android; hence it makes sense to adopt it for your own website too.

The three main parts of Angular Material are a well crafted set of components, the flex layout engine and CDK (for building your own components). All of these are covered in this specialist course as we explore how best to create modern web UIs that look well and work well.

Contents of One-Day Training Course	
<p>Target Audience Developers interested in efficiently bring material design to their Angular 7.2 applications</p> <p>Prerequisites Experienced Angular developers with a flair for UI design.</p> <p>Note: This is not an introductory Angular course - so attendees must already be familiar with the Angular framework.</p> <p>All demos and labs will be in TypeScript, so attendees need to know TypeScript.</p>	<p>Purpose of Material Design Google’s design language Style guide++ Principles of material design</p> <p>Tour of Material Design “Material Design is a unified system that combines theory, resources, and tools for crafting digital experiences.”</p> <p>Overview of Angular Material Implements material design for Angular Exploring its capabilities What it brings to modern UI projects Review of Angular’s forms architecture</p> <p>Delivering UI Capabilities Pagination & navigation Typography Layout Useful role of schematics</p> <p>Form Controls Autocomplete, checkbox, input, radio button, select, slider, slide toggle</p> <p>Navigation Menu Sidenav (creation using schematics) Toolbar</p> <p>Layout List, Grid, Card, Tabs Virtual Scrolling (new in v7) - large lists & fast UI Drag and Drop (new in v7)</p> <p>Buttons, Indicators & Icons Button, button toggle, chips, icons, progress spinner, progress bar</p> <p>Popups And Modals Dialog, Tooltip Snackbar</p> <p>Data Table Table and its data source Paginator and sort header Creation using v6 schematics</p> <p>Tree Tree root and hierarchical nodes Creation using schematics Event handling</p> <p>Introduction to Flex layout Building responsive user interfaces Flexbox usage for various screen sizes The benefit of a grid and column spans Responsive API & Media query Coding custom layout How flex layout works under the hood</p> <p>Architecture of CDK Separate Component Development Kit "general building blocks for UI components decoupled from the visuals of Material Design" New foundation for Angular Material, can also be used separately</p> <p>Using CDK Github location: material2/src/cdk Creating your own components based on CDK for a range of capabilities without too much extra development effort CDK data table</p> <p>Project Using Angular Material in a large project to see how to best utilize its rich capabilities</p>

PWA using Angular 7.2 Service-Worker

PWA Ideas, AppShell, Manifest, Service Worker spec, CLI's --service-worker, Implementation, Project

Web developers have been enviously glancing over at native app developers and admiring all the shiny extra toys they have – the sometimes connected app, instant app start, app store for discoverability, notifications, etc. (of course, web developers do not forget the unique advantages they have – a ubiquitous platform, use of URLs for deep linking, avoiding version hell, etc.).

With the arrival of Progressive Web Applications (PWA), now web developers have the best of both worlds. PWA makes it easy to build web apps that run in modern browsers and behave like native apps.

A PWA is a web app that uses three key technologies – the manifest, service workers and optionally the app shell. The best way to build PWAs is to use Angular 7 & its *Service-Worker* package. The project source tree can be created as normal using Angular 7.2 CLI and then run this to add PWA support: `ng add @angular/pwa`

The aim of this course to to bring Angular 7.2 developers up to speed with how a PWA works, to review the underlying spec and then to dive deeply into how to implement a powerful PWA using Angular's Service-Worker package.

Contents of One-Day Training Course		
<p>Target Audience Web developers wishing to build powerful PWA apps using Angular 7.2 and its <i>Service-Worker</i> package.</p> <p>Prerequisites Good experience of Angular 7.2 programming.</p> <p>This is an advanced course and before attending, attendees should already be quite familiar with Angular 7.2 and TypeScript programming.</p>	<p style="text-align: center;">PWA – What are we trying to achieve</p> <p>All three ideas are important:</p> <ul style="list-style-type: none"> - progressive - web - applications <p>The idea of a client-side mini-proxy server and transient network connections</p> <p>The fact we gain native app capabilities does not mean we lose web app capabilities</p> <p style="text-align: center;">PWA Technologies</p> <p>Manifest Service Worker AppShell</p> <p>Review of how these technologies work</p> <p>Message flows</p> <p>Service workers are quite distinct from web workers – not to be confused</p> <p style="text-align: center;">Intro to PWA with Angular 7.2</p> <p>Angular and Progressive Web Apps</p> <p>What Angular 7.2 offers to PWA app devs</p> <p>Exploring the Service-Worker package</p> <p>Angular CLI and Service-Worker apps</p> <p style="text-align: center;">AppShell</p> <p>An approach to initially offer a minimalist UI that can be cached</p> <p>Gradually add more content</p> <p>Very fast rendering of first view</p> <p style="text-align: center;">Manifest</p> <p>What's in a manifest.json file?</p> <p>Generating default</p> <p>Start url, scope, display, etc.</p>	<p style="text-align: center;">Service Workers Spec</p> <p>Review of W3C Service Workers Spec – we explore how service workers perform</p> <p>Main artifacts</p> <p style="text-align: center;">Angular CLI Generated Code</p> <p>Add PWA using: <code>ng add @angular/pwa</code></p> <p>What does it do?</p> <p>Add new packages: ServiceWorker & PWA</p> <p><code>angular.json/configurations/serviceWorker</code></p> <p><code>ServiceWorkerModule.register</code> call</p> <p style="text-align: center;">ngsw-config.json</p> <p><code>assetGroups</code> <code>InstallMode (prefetch)</code> <code>UpdateMode</code> <code>Resources</code></p> <p style="text-align: center;">Working With Service Workers</p> <p><code>swUpdate</code>: deciding on an update strategy</p> <p><code>swPush</code>: service worker's push notifications</p> <p>Other aspects of service worker programming and configuration</p> <p style="text-align: center;">Tooling</p> <p>Google's PWA site Google LightHouse</p> <p>Debugging and instrumentation</p> <p style="text-align: center;">Internals</p> <p>The source for Angular's Service-Worker package is well worth studying</p> <p>We also look at source for Angular CLI's <code>@angular/pwa</code> package</p> <p style="text-align: center;">Project</p> <p>Creating an Angular 7.2 application that builds on the ideas explored in this course and that scores 100 in LightHouse</p>

Angular 7.2 Internationalization

Regional settings, i18n, i18n Pipes, ngx-translate, Unicode, CDLR, Country Packs

Do you know the wonderful(ly complex) Japanese writing system has three kinds of symbols: kanji, hiragana & katakana. Do you know in Germany street addresses have the house number at the end rather than the beginning (so “7 Main Street” is written as “Hauptstraße 7”). Do you know in Bahrain, the currency has three subunits rather than the usual two (BD 123.456 rather than \$123.45). Do you know the social rules of formality and politeness are much more complex in many cultures compared to English-speaking lands. Do you know that 96% of the world’s population do not live in the USA. All this goes to show

the world is a big place with fascinating differences and well worth visiting. The aim of this course is to help your Angular 7.2 apps travel well.

Critically important i18n aspects that Angular 7.2 app developers need to consider include how to represent strings (in memory, on-screen and in data files); how to input characters in the user interface; how to display numeric, financial and date data according to local customs; how to manage dialog and other resource values and how to develop custom tools to help with localization. This course covers all these and lots more.

Contents of One-Day Training Course	
<p>Target Audience Experienced Angular developers with an interest in preparing their Angular 7.2 apps for international markets</p> <p>Prerequisites This is not an introductory Angular 7.2 course - so attendees must already be familiar with the Angular framework.</p> <p>All demos and labs will be in TypeScript, so attendees need to know TypeScript.</p> <p>No prior internationalization experience required.</p> <p>PLEASE NOTE: the first half of this course covers general internationalization programming and the second half covers internationalization with Angular 7.2.</p>	<p style="text-align: center;">Overview</p> <p>Importance of internationalization (i18n) we sell in your language/buy in our own i18n concepts - globalization, internationalization, localization</p> <p style="text-align: center;">Review of how i18n works</p> <p>Common source tree Multiple localization assets Helping the translators Thinking of internationalization from the beginning – not as an afterthought Externalizing assets we need to change</p> <p style="text-align: center;">i18n Tooling</p> <p>Setting up a suitable i18n dev / test bench Need to get fully local OS installed (not just English OS with international strings) XLF files and xlifffmerge</p> <p style="text-align: center;">i18n and the modern browser</p> <p>How different browsers handle regional settings and language selection Tools in the browser (e.g. dev tools) Languages in use Google Input Tools For Chrome https://www.google.com/inputtools/chrome</p> <p style="text-align: center;">Unicode</p> <p>Character set / scripts Text direction & text layout Inputting complex languages</p> <p style="text-align: center;">CDLR</p> <p>“The Unicode CLDR (Common Locale Data Repository) provides key building blocks for software to support the world's languages, with..repository of locale data”</p>
	<p style="text-align: center;">Angular’s Built-in i18n tools</p> <p>https://angular.io/guide/i18n Overview of Angular and i18n The i18n attribute – translatable strings ng x i18n</p> <p style="text-align: center;">i18n Angular Pipes</p> <p>i18n and DatePipe, CurrencyPipe, DecimalPipe and PercentPipe</p> <p style="text-align: center;">Intro to ngx-translate</p> <p>http://www.ngx-translate.com This modular library provides:</p> <ul style="list-style-type: none"> - A service - A directive - A pipe <p>Good for dynamic and static content</p> <p style="text-align: center;">Advanced ngx-translate</p> <p>Idea of loaders Review of provided loaders Comparison with Angular’s built-in i18n tooling</p> <p style="text-align: center;">Country Packs</p> <p>Often need country specific additions (e.g. for local regulations) Building (lazily loaded) country packs for country-specific add-on functionality Using Angular CLI 7’s ng g library Integrating country packs with main app</p> <p style="text-align: center;">Project</p> <p>A larger sample Angular project showing the right way and wrong ways to manage product development aimed at global markets</p>

NgRx 7

Reactive + State, State Store, Side Effects, DevTools, Entity, Schematics, Architecture, Project

We use the term “state” to describe nuggets of data whose lifetime outlives that of a single call to an event handler (e.g. auth token, contents of shopping cart, custom color selection for sidenav). An Angular app is composed of a hierarchy of components. Sometimes state that is only used by a single component can be stored within that component; state shared between related components (near each other in the hierarchy) can be passed among them directly. Using services with dependency injection is also an option; but for more substantial applications with many components, managing state needs more attention. Enter NgRx ...

NgRx is a well organized suite of packages to manage application state in a RxJS observable cache. It is very popular, because it comprehensively solves the state management issue that every large Angular 7.2 app ultimately will face. The three key participants in NgRx are actions, state and reducers (which literally reduce an existing state and an action to a new state). Though it can be used on the server, in the real world NgRx is mostly used on the client (on the server, data usually ends up in a database). In this detailed course we explore the world of NgRx 7 and see how it can be of real benefit to larger Angular 7.2 apps.

Contents of One-Day Training Course	
<p>Target Audience Developers wishing to manage client-side state in their Angular 7.2 apps using NgRx 7</p> <p>Prerequisites Experienced Angular developers building larger applications who need to more seriously consider their client-side state management architecture.</p>	<p>Reactive Meets State What problem are we trying to solve The observable pattern and its uses Parts of RxJS that are of interest to us Lets look at state within an Angular app When to use NgRx (and when not to)</p> <p>Overview of NgRx 7 Exploring the NgRx Platform Important packages and their interactions Getting it installed and running Adding to development environment</p> <p>Concepts State (and the idea of immutability) Actions (state changes) Reducers</p> <p>@ngrx/store Introduction OnPush change detection strategy Boilerplate code for NgRx Creating reducers</p> <p>Advanced @ngrx/store State composition Selectors ngrx-store-localstorage</p> <p>Meta-Reducers Meta actions & meta reducers Wrapping a reducer with a meta reducer Parameter to <code>StoreModule.forRoot()</code></p> <p>@ngrx/effects What is an effects model? Isolating change, making pure components Feeding actions into state cache Event sourcing architecture <code>@Effect()</code> decorator</p> <p>@ngrx/router-store Combining Angular Router & <code>@ngrx/store</code> ROUTER_NAVIGATION StoreRouterConnectingModule</p> <p>@ngrx/entity What is an entity collection? CRUD operations on entity collections Uses for type-safe adapters/entity selectors</p> <p>@ngrx/store-devtools <u>Redux DevTools</u> Instrumentation tooling for the store Instrumentation options</p> <p>@ngrx/schematics Idea of schematics A scaffolding library that integrates NgRx with Angular CLI Available blueprints</p> <p>Case Study: NgRx & Shopping Cart NgRx is very often used to build the shopping cart feature in eCommerce apps Let's see how to do it properly</p> <p>NgRx Internals Tour of NgRx source code https://github.com/ngrx/platform NgRx source is managed as a monorepo Discover how it all fits together</p> <p>Architecture Considerations Exploring the architectural issues that need to be considered in order to successfully leverage NgRx rich capabilities in an app</p> <p>Project Combining what we have learnt in the course to use in a larger NgRx project</p>

Ionic 4

System Model, Ionic/Core, Ionic/Angular, CLI, Stencil, JSX, Capacitor, PWA Toolkit

Ionic 4 is a complete re-imagination of how Ionic works based on W3C Web Component standards, while retaining most of the existing Ionic API used by apps.

Most modern web frameworks are recently adding significant Web Component support (e.g. [Angular Elements](#)) and with v4 we see this in the Ionic world.

User interface developers have for decades been creating apps by weaving together their own code with pre-built blocks of functionality – these are sometimes called widgets, custom controls or just components. With [Web Components](#), this approach is now becoming a reality for the Web Platform also. We can now have, say, a small Ionic component consumed by a larger React component hosted by a plain JS application.

This course brings web developers with little or no previous Ionic experience up to speed with Ionic 4 programming. We will see how Ionic 4 is a powerful framework that enables developers use TypeScript and familiar web technologies to build apps for mobile devices. Ionic contains a set of mobile-friendly UI components, a library to interact with native services, a CLI and a toolkit for creating progressive web apps.

Contents of One-Day Training Course	
<p>Target Audience Web developers who wish to build either PWA or native apps for mobile devices using the latest web technologies.</p> <p>Prerequisites Experience of web development, in particular Angular. During the course we discuss the relationship between Ionic v4 and Angular (with Ionic v3, use of Angular was mandatory, with v4 it is optional).</p> <p>No previous Ionic experience needed – we start with the Ionic v4 fundamentals.</p> <p>All demos and labs will be in TypeScript, so attendees need to know TypeScript.</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <p>W3C Web Components Ideas behind this W3C set of standards Custom Elements Shadow DOM HTML Templates</p> <p>Ionic 4 Overview Developing for mobile – smaller screens, use of touch, limited device features How Ionic delivers a set of capabilities to facilitate web developers building cross platform mobile apps</p> <p>Concepts Major architectural subsystems and how they work together. Building your first Ionic 4 app The toolchain</p> <p>Ionic CLI (v4) A command-line interface to generate, serve, build and package Ionic apps Important commands and their options</p> <p>Ionic 4 Core Based on Stencil UI components Themes Utils Controllers Configuration</p> <p>Ionic 4 Angular Optional Angular integration Injectable controllers Directives for virtual Interaction with Angular routing IonicModule (an NgModule)</p> </div> <div style="width: 48%;"> <p>Stencil Ionic Stencil is a compiler that makes Web Components Use of JSX The idea of a virtual DOM Async rendering Data binding</p> <p>Capacitor A mobile app will often need from time to time to use services of the native OS – we see Ionic Capacitor delivers capability (Capacitor takes over the role of Cordova) The Capacitor API and its usage How it works on different substrates</p> <p>Ionic PWA Toolkit Principles of progressive web apps that Ionic PWA Toolkit implements Service workers / push notifications Configuration / routing</p> <p>Ionic 4 Internals Tour of Ionic 4 source code https://github.com/ionic-team Explore how the various projects mesh together and can be extended</p> <p>Architecture Considerations How to elegantly design mobile apps Handling differences among device types Issues to pay attention to</p> <p>Project Bringing together all the ideas discussed in the course into a larger project to create a mobile app that will be competitive in the app stores</p> </div> </div>

Web Components/Angular Elements/Microfrontends

Custom Elements, Shadow DOM, NgElement, createCustomElement, App Shell, Architecture

Full stack developers have been successfully using microservices server-side for a few years and now they would like to apply that architecture client-side, an approach known as microfrontends. In some ways microfrontends are similar to microservices: a clear need to break up a monolith application, allow different parts of a large app to evolve and be deployed at their own pace, perhaps using distinct foundational technologies. In other ways, they are different: microservices can run separately in data centers on a (e.g. Kubernetes) container cluster, whereas we wish a set of microfrontends to run isolated in a web browser

and yet appear to end-users to be a single integrated interactive application, with some shared capabilities. In this specialist course we first review W3C Web Components - a set of standards that allow components to be produced and consumed by different web frameworks (or different versions of the same framework). Then we look at Angular Elements, which allows the construction of web components using Angular. Then we explore microfrontends - what are they, how to best build them using Angular Elements, how to host them (app shell) & how to design large-scale web applications using them.

Contents of One-Day Training Course	
<p>Target Audience Experienced Angular developers working on large Angular projects who wish to compose them out of microfrontends.</p> <p>Prerequisites Full stack developers with good all-round experience of Angular 7.2.</p> <p>Awareness of role of containers and microservices for server-side development highly relevant.</p>	<p>Overview What are we trying to achieve? Dividing an app into dynamic components * Standards - W3C Web Components * Implementation - Angular Elements * Design approach - Microfrontends</p> <p>W3C Custom Elements Create your own HTML elements Attribute, properties, events New CustomElementRegistry How we can use them for microfrontends</p> <p>W3C Shadow DOM Shadow tree and light tree Angular ViewEncapsulation.ShadowDOM (note: Native is deprecated)</p> <p>Support Features: Slots, HTML Templates/Custom Events Relevant additional HTML/DOM features that modern browsers support</p> <p>Composability in Angular Angular dynamic components Lazy loading & extensibility Specialist use of NgModules</p> <p>Angular Elements Intro NgElement, ngBootstrap() createCustomElement() Content projection (new in Elements v7)</p> <p>Advanced Angular Elements Build process Managing element lifecycle / evolution Loading a library containing elements Review of Angular Elements' source tree Importance of Render3/Ivy [Angular 8]</p> <p>Microfrontend Architecture What microservices brings to server app How best to apply idea client-side Microfrontend = large (somewhat contained) slice of interactive app Critical to make group of them appear as as a single integrated app</p> <p>Building Microfrontends Structuring microfrontends Source layout Expected classes and interfaces Design patterns</p> <p>App Shell Microfrontends have to live within a shell Custom app shell hosts microfrontends What capabilities it could offer: notification – recently used-quick links-app discovery</p> <p>Microfrontends and .. Routing Internationali[z]s]ation Security / Styling Sharing widgets</p> <p>Browser Containers Interesting idea: containers in the browser containers = namespaces + cgroups How best to bring idea to browser [polyfill] Hosting microfrontend in browser container</p> <p>Project Bringing together everything covered in this course, we conclude with a review of a substantial project that uses microfrontends, based on Angular Elements</p>

Node.js 11 HTTPS/Express 4.16/PUG Using TypeScript Web UI app, REST API app, Node HTTPS, Express, Generator, Routing, PUG, Project

Because of its ease of development, similarity with web client-side programming, maturity along with its expanding framework and tooling story, a Node-based approach to server-side development is very compelling. Since TypeScript is usually used client-side, it make sense to also use it server-side.

This course explores using TypeScript to develop server-side web UI and REST API applications using the latest technologies based on Node.js 11. It covers a mix of technologies that together allow developers to quickly build robust server-wide web solutions.

The Node.js HTTP(S)&HTTP/2 modules are how Node apps talk to the HTTP protocol family. The Node-based Express Application Framework is how rich web UI and web API apps can be built. It exposes powerful routing and middleware capabilities. The PUG template engine (the latest version of what used to be called Jade), transforms an HTML-like template syntax plus supplied data values into HTML which is sent to the web browser for rendering. In some ways PUG competes with Angular on the client – we contrast what they offer and investigate how to use them together in the same solution (often a sensible approach).

Contents of One-Day Training Course	
<p>Target Audience Web developers seeking to build rich web UI and REST API apps using the latest Node runtime, Express application framework and PUG template engine.</p> <p>Prerequisites Attendance at our <i>Node.js 11 Runtime Programming Using TypeScript</i> course or equivalent experience.</p> <p>All demos and lab exercises will be in TypeScript.</p>	<p>Building Web UI apps Review of server-side and client-side UI development options Role of templating and access to data Creating compelling user experiences in a Node world</p> <p>REST API Apps Review of REST API concepts Designing a REST API using Node Evolving a REST API Security issues</p> <p>Node.js 11 HTTP[S] Modules HTTP message flows using Node Use of createServer to create server server.listen and server.on syntax HTTP message content – headers, etc. Error handling considerations (error.code and Exxx values)</p> <p>Introduction to Express (4.16) Application framework based on Node Main components: express(), Application, Request, Response and Router Intro to what Express calls middleware</p> <p>Express Generator Recommended file layout for Node/Express apps Express-Generator constructs boilerplate code and layout quickly Reviewing generated content</p> <p>Express Routing Converting incoming URLs and query syntax to action invocations Defining and configuring a router</p> <p>Implementing an API Testing an API</p> <p>Express Middleware Additional custom steps added to processing pipeline Stack of middleware functions Helpful auxiliary packages such as path, morgan and bodyParser</p> <p>PUG Template engine based on Node & Express Intro to PUG Template syntax Intro to PUG framework - runs on server and transforms syntax + data into HTML</p> <p>PUG Template Syntax Concise representation of markup Consumes data to generate contents Loops and control flow Useful shorthand for common needs</p> <p>PUG API Usage pug.compile[File Client FileClient ClientWithDependenciesTracked] pug.render and pug.renderFile Options interface</p> <p>Angular Client-side UI Server-side UI vs. Client side Why server-wide UI is important (e.g. efficient access to server-side data) even in a world with Angular Integrating Angular on the client with server-side coding</p> <p>Project Developing an integrated project that brings together all the topics covered in the course</p>

Operations Faculty / DevOps School

Shell

- The Bash Shell

Repository

- GIT And Github

Toolchain

- TypeScript/Node/Angular Toolchain
- C/C++ Toolchain

Process

- Agile Development Process

The Bash Shell

Tour, Variables, Control Flow, Functions, Builtins, History, Error Handling, Aliases, Best Practices

Bash is a powerful shell environment that provides excellent interactive and scripting control of the underlying OS, installed software platforms and can be used to build and manage your own custom tools too. Bash is good for repetitive chores, setting up an environment, automating non-trivial workflows, software builds and product installations.

The goal of well-written shell scripts is to allow repeatable / configurable / reliable task execution. The overall aim of this course is to equip attendees with a clear understanding of how to achieve that using Bash.

Professionals skilled in the art of Bash scripting are significantly more productive. Knowledge of Bash is a “must-have” tool in the skillset of every heterogeneous system administrator or developer. Though initially popular on Unix and Linux platforms, Bash is now very widely available, including on macOS and Microsoft 2019, as an optional install). With a little effort you can even get Bash to work on mobile OSes. This wide availability is important because it means the effort you invest to learn Bash well now will pay repeated dividends in future regardless of which OS you use.

Contents of One-Day Training Course	
<p>Target Audience Administrators, developers and power users who wish to use the command line to interact with the system and to create shell scripts to automate such activities</p> <p>Prerequisites Some previous experience of shell usage required.</p> <p>General knowledge of the operating system command layout is useful.</p>	<p>Overview</p> <p>What is Bash good for? Compliance with POSIX Shell Standard Feature tour Setting up Bash – for Linux/macOS, Windows Subsystem For Linux and Git Bash (limited) How Bash compares to PowerShell</p> <p>Control Flow</p> <p>if select / case for break / continue</p> <p>Variables</p> <p>set and unset for local variables env Export to child processes</p> <p>Processes and Command Line</p> <p>Command pipeline Command line arguments Exit code and results Process architecture of executing Bash exec command \$\$ - process id of process executing shell</p> <p>Functions</p> <p>Parameters to functions Viewing declared functions Nested functions Job control</p> <p>Configuring the Environment</p> <p>.bashrc Start up scripts - profiles Login shell</p>
	<p>Scripts</p> <p>Creating/calling/debugging Bash scripts Sourcing – external library of functions Job control History Arranging larger blocks of script</p> <p>Builtin Functions</p> <p>Wide range of builtins Regular expressions readline</p> <p>Alias</p> <p>Listing aliases Role of aliases alias and unalias functions</p> <p>Error handling</p> <p>Error handling in scripts Signal handling trap</p> <p>File Handling</p> <p>How files are represented in Bash general file I/O umask read</p> <p>Scripting Best Practices</p> <p>Small chunks of script Pay attention to debugging Resilience to error situations Similar to/different from regular programming</p> <p>Project</p> <p>Explores the scripting needed for automated workflow for a non-trivial enterprise scenario</p>

Git and GitHub

Working Dir/Staging, Local/Remote, Clone, Push, Pull, Branch/Merge, Monorepo, GitHub Desktop

Source code is by far the most important asset any software company owns. It is more valuable than buildings, brand names, computer hardware, furniture or anything else a software company has. Source code needs to be valued and treated like the very important company asset that it is. Hence the need for a robust source code management system.

Git is the most popular source code management system; GitHub.com is the most popular Git cloud hosting solution. Either Git alone or Git and GitHub can be used to comprehensively manage and protect source.

Even if not using GitHub for their own source, app developers still need to get familiar with it as most of today's popular open source projects are using it and app developers will invariably need to use these.

This course covers both and helps developers gain hands-on experience in how to incorporate both into their development workflow. Many Git-related terms have entered the developer lexicon – push, pull request, cloning, forking, promoting, repo – and this course helps attendees understand each concept and mentally tie everything together to see how they work in unison.

Contents of One-Day Training Course	
<p>Target Audience Software engineers and architects wishing to correctly manage valuable source trees</p> <p>Prerequisites Programming knowledge and some previous hands-on experience of any source control system.</p> <p>For the GitHub part of the course, each attendee will need their own (free) GitHub login to complete the lab work.</p>	<p>Distributed Version Control Using what you might already know Adding distributed influence Organizing teams via Git Strategies for managing source trees Terminology - push / pull, clone, fork, fetch, branching and merging Lifecycle of a single line of code</p> <p>Getting Started With Git Installing and configuration Simple usage We explore where source can be stored (local and remote) Promoting from working dir to staging and beyond What happens during a commit</p> <p>Working Locally Init vs. clone File system layout The .gitignore file Creating, modifying and deleting locally Cancel changes (revert) Logging/history/status</p> <p>Branching / Merging Creating and listing branches Merging a branch Change tracking / diff / Rebase Feature branches vs. trunk development</p> <p>Remote Remote protocols Connecting to remote repo servers Push and pull commands Fetch command</p> <p>Command Line Tooling Porcelain vs. plumbing Beyond the basics - more complete look at advanced command line tools for Git</p> <p>Managing as part of Toolchain Git as part of toolchain Use with other tools Call via scripting (automated test runs, linting, Continuous Integration/Continuous Delivery - CI/CD) What to do with generated info</p> <p>Monorepo Each project need not exist in separate repo Multiple projects can be placed in a single repository – known as a monorepo Practical ideas for using monorepos</p> <p>GitHub Source repositories in the cloud Public (free hosting) & private (fee-paying) Organizations and teams Interacting with open source projects (issues, releases, changelog, pull requests)</p> <p>GitHub Desktop Enhanced (Electron-based) GUI to comprehensively manage Git repositories Easy to set up and use</p> <p>Git Internals The source code for Git itself is an interesting read: https://github.com/git/git What can we learn from exploring it?</p> <p>Project Organizing a large source tree using Git Deciding on project and repo layout</p>

C/C++ Toolchain

Understand, Organize, Build, Write, Document, Test, Optimize, Verify, Debug, Package, Explore, Deploy

Lots of things have to happen correctly in order to efficiently transform source code on developers' computers into commercial digital products in use by customers around the world.

After writing the source, the code has to be built, unit tested, documented for developers, stored in repositories, optimized, debugged, the performance monitored, the source style has to be verified, libraries and components packaged & applications deployed. For C/C++ developers, a toolchain is needed with appropriate tools for each of these tasks.

The C/C++ toolchain often gets less attention than the languages themselves or high-profile libraries (such as STL) but it is vital for highly productive developers to have a deep understanding of the C/C++ toolchain and what it can offer. As projects get larger and there is pressure to deliver updates in shorter time frames but with higher quality expectations, successfully leveraging the C/C++ toolchain and all its rich capabilities makes all the difference.

This course explores a range of useful developer tools that when used together achieves just that.

Contents of One-Day Training Course	
<p>Target Audience Software engineers who wish to more fully explore the toolchain options available for C/C++ projects</p> <p>Prerequisites Experience of C and/or C++ programming. At least some experience of the C/C++ toolchain.</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <p>Tour of C/C++ Toolchain Range of tools needs for high-productivity high-quality C/C++ programming</p> <p>Intro to CMake CMake is the popular cross-platform C/C++ build system Relationship to native build systems Defining the build process with CMake Handling the build & source directory tree</p> <p>VIM Options for editing code across platforms General use of VIM VIM for C/C++ source editing</p> <p>Intro to Unit Testing Benefits of unit tests & overview Designing tests and automatic evaluation Intro to Google C++ Testing Framework</p> <p>Advanced Unit Testing More detailed look at Google C++ Testing Framework and how it can be successfully used in large C/C++ projects Mocking, code coverage, performance</p> <p>Code Documentation Doxygen auto-generating documentation Professional finish</p> <p>Lint, etc. Making sure code complies with styling and other requirements Use of lint and other specialist tools</p> <p>Larger Source Trees Organizing large C/C++ source trees Header files, libs, main</p> </div> <div style="width: 48%;"> <p>C/C++ Compilers The three major compilation toolkits are GCC, LLVM and Visual C++ compiler plus there are many smaller toolkits</p> <p>LLVM Tour of LLVM project CLang Optimizer Standard library</p> <p>Language Service for C/C++ Using LLVM to build C/C++ code Command-line options Compilation passes</p> <p>Debugging Native debugger Extracting run time information Symbol management</p> <p>Package Management Unlike other languages, C/C++ does not have popular package management systems (e.g. no C/C++ equivalent of npm or nuget) We look at what are the packaging options available to C/C++ apps (e.g. chocolatey)</p> <p>Deployment Approaches to packaging C/C++ apps for different platforms Deciding where & how to place assets</p> <p>Real-world Toolchain Usage Exploring the toolchain setup for a large open-source C/C++ project</p> <p>Project Correctly setting up toolchain usage for our own enterprise projects</p> </div> </div>

TypeScript/Node 11/Angular 7 Toolchain

Understand, Organize, Build, Write, Document, Test, Optimize, Verify, Debug, Package, Explore, Deploy

Lots of things have to happen correctly in order to efficiently transform source code on developers' computers into commercial digital products in use by customers around the world.

After writing the source, the code has to be transpiled, unit tested, documented for developers, stored in repositories, optimized, debugged, the performance monitored, the source style has to be verified, libraries and components packaged, minified & applications deployed. For TypeScript developers, a toolchain is needed with appropriate tools for each of these tasks.

The TypeScript/Node/Angular toolchain often gets less attention than the language/runtimes themselves but it is vital for highly productive developers to have a deep understanding of the toolchain and what it can offer. As projects get larger and there is pressure to deliver updates in shorter time frames but with higher quality expectations, successfully leveraging the TypeScript/Node/Angular toolchain and all its rich capabilities makes all the difference.

This course explores a range of useful developer tools that when used together achieves just that.

Contents of One-Day Training Course	
<p>Target Audience Software engineers who wish to more fully explore the toolchain options available for TypeScript projects for Node and/or Angular.</p> <p>Prerequisites Experience of TypeScript programming for Node and/or Angular.</p>	<p>Toolchain Tour Range of tools needs for high-productivity high-quality TypeScript/Node/Angular programming Node 11's Inspector Protocol</p> <p>TSC – TypeScript Transpiler Tsc and its options More advanced uses for configuration file</p> <p>TypeScript Language Service Written in TypeScript, an external process supplying a range of language services How to call from non-TypeScript clients</p> <p>Visual Studio Code Code editor for various languages Use of Visual Studio Code for TypeScript programming, both Node and Angular</p> <p>TypeDoc Code Documentation TypeDoc auto-generating documentation Professional finish</p> <p>TSLint, etc. Palantir's TSLint is used to make sure code complies with styling/maintainability</p> <p>Angular Package Format The Angular Package Format is a well-specified layout used for Angular packages Third-party libraries should follow layout</p> <p>Angular Augury In-depth debugging of Angular content Chrome integration (v1.2x supports ng)</p> <p>WTF - Web Tracing Framework Low-level collection of precise high-volume tracing data Visualizing results</p>
	<p>ts-node Node-based execution environment for TypeScript code</p> <p>Gulp.ts Gulp is a very popular task runner We explore how to write gulp files in TypeScript (requires ts-node) Defining the build process using gulp.ts Handling the build & source directory tree</p> <p>WebPack 4 Role of module bundler Dependency graph and bundles</p> <p>NPM & Yarn Creating packages using Node Package manager (NPM) and Yarn Publishing to NPM Adding extra features to package (e.g. CI)</p> <p>Jasmine & Karma What to test Test syntax in Jasmine Executing tests using karma Karma.config.js</p> <p>Protractor End-to-end testing of angular apps Writing tests for entire integrated app Automated collection of results</p> <p>Real-world Toolchain Usage Project Exploring the toolchain setup for the main Angular project itself – a very large open source TypeScript that is widely deployed Correctly setting up toolchain usage for our own enterprise projects</p>

Agile Software Development

Methodology, Values, Principles, User Stories, Pairing, Individual/Team Dynamics, Test First, Refactoring

Agile software development is a highly productive strategy for organizing project teams. It allows software to be quickly written for today’s needs and that can easily evolve to keep up with an every-changing business environment. It exploits the best of individual and team creative efforts. Agile shuns the heavy “ceremony” of documentation-rich rigid software development processes; and instead advocates a code-centric approach with just enough documentation to genuinely satisfy all stakeholders’ needs. This approach is more satisfying for developers and is more finely aligned with customers’ true requirements. It results in

much richer contributions from all project participants. It advocates short iterations that fulfills the “release early and release often” aspiration. Teams are based on a flat organizational structure with a high-degree of face-to-face communication with a “product owner”. It treats the software development business as an iterative subscription model, rather than a once-off event. It is a simpler approach to writing software – and easier for a team to use. In today’s highly competitive environment, whichever software team can write quality code fastest wins. This course is ideally suited to dev teams making the agile move to gain competitive advantage.

Contents of One-Day Training Course	
<p>Target Audience All members of software project teams – managers, architects & programmers - and the product owner.</p> <p>Prerequisites Good understanding of software development issues</p> <p>Significant project experience (either as a specifier or an implementer) .</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <p>Efficient Light Methodology Current state of play in s/w projects Need for new approach to programming Families of agile methodologies and selecting best techniques from each “An ecosystem that ships software”</p> <p>Values Individuals and interactions over processes and tools Working software over comprehensive documentation Custom collaboration over contract negotiation Responding to change over following a plan</p> <p>Principles Satisfying customers, welcoming changes, delivering working software, work together, projects by motivated individuals, face-to-face communications, progress measured by running code, sustainable development, quality matters, simplicity, self-organizing teams, reflection Individual & Team Dynamics High communication levels Shared knowledge spaces Benefits of higher-skilled developers Handling a creative team Collaborative & competitive</p> <p>Design Documentation Doc strategy – succinct yet sufficient Documentation for all project stakeholders Useful templates for project documents</p> </div> <div style="width: 48%;"> <p>Agile Projects Agile project management Iterations and releases</p> <p>Requirements via User Stories What is a user story and how to create one? Certain number completed each iteration</p> <p>Design Architecture with a little ‘a’ UML as a sketch Design today for today; refactor tomorrow</p> <p>Test-First Written by programmers (unit tests) & customers (acceptance tests) Putting the concept into practice</p> <p>Pair Programming “A driver and a navigator” (think rally car racing) - not “a driver and a passenger” Duties of the navigator Continuous code review</p> <p>Refactoring What happens when software matures? “Improve the design after the software has been written” Business Issues Fixed price/fixed-scope contracts, responsibilities, outsourcing, etc.</p> <p>Tools Continuous Integration/Delivery tooling Unit Test tools Refactoring tools</p> <p>Sample Project Exploration of the use of agile processes in a case study development project</p> </div> </div>

Operations Faculty / Identity School

Security

- Fundamentals of Security
- Angular 7.2 Security and Authentication

Privacy

- GDPR & CCPA – A Developer's Viewpoint

Fundamentals Of Security

Security Concepts, Crypto, Certs, Identity, Trust, Attack Patterns, ISO2700x, Reviews, Processes

The entire software user base - specifiers, management, users, developers, administrators – all demand security and all have a role in delivering it. The goal of this course is to teach participants a common core set of fundamentals that is the first step in achieving this. Security should be treated as part of the expected skill set of every software professional. They need a fundamental understanding of security issues, before considering how to address them in the apps they develop and deploy (or better, as an integral part of the design). Security considerations must be part of software decision making, though they should not

overwhelm it. Most software pros already have some awareness of security issues – this course builds on this basic knowledge and ensures the entire dev and infrastructure teams have a heightened and consistent appreciation of security concepts, along with a deep understanding of the core security standards.

This course focuses on the fundamental concepts and standards behind security. It is independent of any operating system or software environments Those attending will be well placed afterwards to think about optimum implementation strategies for their platforms.

Contents of One-Day Training Course	
<p>Target Audience Software developers and IT. professionals who need a good grounding in all the important security concepts</p> <p>Prerequisites Experience of working on software projects, including development, deployment and ongoing service provision.</p> <p>No previous security programming or infrastructure experience is required, though any such knowledge would be beneficial.</p>	<div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p>Security Services</p> <ul style="list-style-type: none"> Message integrity Authentication Non-repudiation Proof of submission/delivery Confidentiality Privacy Anonymity <p>Security Concepts</p> <ul style="list-style-type: none"> Network authentication, authorization, auditing, ciphers, key exchange, hashing, salting, least privilege, default lockdown-mode, canonicalization, leaks, buffer overflows, attacks (dictionary,mitm) <p>Cryptography</p> <ul style="list-style-type: none"> Symmetric and asymmetric crypto Latest crypto standards AES and SHA-3 Elliptic Curve Cryptography Comparison of performance & robustness Problems with older specs [des/md5] <p>Digital Certificates</p> <ul style="list-style-type: none"> Public Key Infrastructure (PKI) Revocation and CRL Attributes, certificate fields <p>Identity</p> <ul style="list-style-type: none"> Identity management Identity and federation Limiting dispersal of identity <p>Trust Services</p> <ul style="list-style-type: none"> Offloading work to trusted third parties Whom to trust, how, and to what extent? Trust server </div> <div style="width: 45%;"> <p>Common Attack Patterns</p> <ul style="list-style-type: none"> Social engineering Web app attacks and insider attacks Human factors <p>ISO 2700x</p> <ul style="list-style-type: none"> International standards for identifying, documenting and countering threats The proposed ISO 2700x series Purpose of ISO 27001 Information Security Management Systems <p>Security Reviews</p> <ul style="list-style-type: none"> Conducting security reviews Security threats – from inside and outside Building a threat model <p>A Security Development Process</p> <ul style="list-style-type: none"> Integral part of how we write software Best practices as part of dev process Ongoing influence <p>A Security Infrastructure Process</p> <ul style="list-style-type: none"> Security policy in the enterprise Secure deployment and operations Advisories – CERT, vendor-specific <p>Design Patterns for Security</p> <ul style="list-style-type: none"> How to correctly design security features into your software systems <p>Security and ...</p> <ul style="list-style-type: none"> Storage, backups, networking, WiFi, user interface, identity, kernel, etc. <p>Project</p> <ul style="list-style-type: none"> Designing a secure programmable infrastructure for a sample system </div> </div>

Angular 7.2 Security and Authentication

CSP, Contexts, Sanitizers, Schema, XSS, CSRF, CORS, XSSI, Authentication Workflow, Web-Authn

There are many design choices web developers make every day that can positively/negatively impact web app security. Security is not a task just to be left to security experts (though having them on the team is certainly a good idea). No, every web developer needs a strong grounding in both web security in general and the security of the web framework they use in particular. This course supplies both for Angular 7.2 app developers. We start with a thorough review of general browser security and then proceed to see how Angular can help in building secure web applications, including exploring in detail how to build authentication.

Angular has a compelling security story, responding well to potential attack vectors. By default an Angular CLI-generated app is very secure. As code is added, security settings can be carefully adjusted as needed.

Authentication is one of the most complex and yet most important aspect of any substantial Angular application. By breaking it into manageable chunks, attendees will appreciate how a well organized authentication workflow should be, and see how to build this inside an Angular application (using [NgRx 7](#) to store the auth token). We also look at authorization and auditing.

Contents of One-Day Training Course	
<p>Target Audience Angular 7.2/TypeScript developers wishing to gain a deeper understanding of how web security in general and security within an Angular app in particular work. Also those who need to implement authentication within an Angular app.</p> <p>Prerequisites This is an advanced course and as prerequisites attendees must have an understanding of security fundamentals and general experience of Angular development</p>	<p>Overview of Browser Security TLS / SSL Input validation / output encoding Client-side security Open Web Application Security Project Content Security Policy (CSP) CSP feature tour Directives Policy definition Integration with other specifications Strict Transport Security “Defines a mechanism enabling web sites to declare themselves accessible only via secure connections” - RFC Web Cryptography Running cryptographic algorithms inside a browser (AES, RSA, HMAC, SHA, etc.) A W3C recommendation from Jan 2017 Overview of Angular Security Angular’s security best practice XSS CSRF Security contexts & sanitizers “Security risk” marking in doc Cross-Site Scripting (XSS) Idea of malicious code-centric Protecting the DOM Tackling XSS in Angular Sanitizers Types of Angular security contexts Role of sanitizers Custom sanitizers Review of schema for security definitions</p> <p>Cross-Site Request Forgery (XSRF) What browser app needs to prevent XSRF Angular 7.2 HttpClient and XSRF CORS Cross-Origin Request and CORS request A server-side feature / Enabling CORS Cross-Site Script Inclusion (XSSI) JSON APIs and security Preventing execution of JSON responses Dev Tools & Security Review of Chrome Devtools’ Security tab Lighthouse Security audits Angular 7 Authentication Intro Designing auth workflow for Angular app Login / logout UI & status UI API calls for auth and retrieving token Storing JWT auth token / supplying to APIs AuthGuard for routing Implementing Authentication Important decisions / security implications Public vs. secure routing targets Using NgRx 7 to manage the auth token Two-factor authentication (using Twilio) Extending with authorization and auditing Web Authentication API FIDO and W3C have released web-authn Use of web-authn by secure Angular 7 app Bringing together the ideas covered in</p>

GDPR & CCPA – A Developer’s Viewpoint

Privacy Primer, Privacy & Software Platforms, Implementing Consent, SAR, Breach, Compliance

One of the most significant yet understated achievements of the European Union in practically helping its 500 million citizens is the GDPR – [General Data Protection Regulation \(EU\) 2016/679](#). The upcoming California Consumer Privacy Act (CCPA), sometimes called the “[American GDPR](#)”, is substantially similar to the GDPR but has some differences. Though somewhat bureaucratic, when you understand how GDPR and CCPA work, they are actually quite a sensible approach. If you were asked to invent your own data protection framework that genuinely protects citizen data privacy rights, you probably would end up with something

close to GDPR/CCPA. Global companies are being encouraged to [implement GDPR on a world-wide basis](#) (with suitable local adjustments), because it is good for their customers and to be ready for the equivalent rules that are likely to be adopted by governments around the world over the next few years.

This specialist course for software developers explores how the architecture of their global software platforms can be adjusted to accommodate implementing GDPR / CCPA and similar. It focuses in on how to implement software features needed to deliver robust data privacy.

Contents of One-Day Training Course	
<p>Target Audience Experienced software engineers who need to implement GDPR-/CCPA-related functionality in the software platforms they are developing.</p> <p>IMPORTANT: This course is not legal advice. This course is written and presented by a software engineer, not a lawyer. To get a complete all-round explanation of all aspects of GDPR/CCPA and to get legal advice on how their organizations should implement these, attendees also separately need to consult with qualified lawyers licensed for the jurisdiction(s) relevant to their project(s).</p> <p>Prerequisites Good all-round experience of software development, at a CTO, software architect or senior software developer level.</p>	<p style="text-align: center;">Privacy and Data Protection Categories</p> <p>What are we trying to protect? What are the possible rights that could be conferred on citizens/companies? The idea of a data protection framework and how it might work Categories of data / heightened protection for some (e.g. personal health data) Introducing GDPR is like when recycling arrived - a bit of a change at first, soon became embedded in how we behave</p> <p>A layman’s GDPR/CCPA Intro What is the GDPR/CCPA/similar? Organizations have responsibilities Individuals have rights What are data controllers, data processors and data subjects? CCPA vs. GDPR Lawful reasons for processing their data Adequacy Decision & international</p> <p style="text-align: center;">GDPR/CCPA & Software Platforms</p> <p>What are important features of GDPR/CCPA for cloud and enterprise software platforms? GDPR/CCPA as one feature of the wider corporate software infrastructure</p> <p style="text-align: center;">Privacy Model for Software Platforms</p> <p>Recommend a corporate privacy model, with input from engineering, ops, sales, .. Want one clear global set of features / procedures that encompass global privacy requirements and can adapt in future</p> <p>Personal data stored needs to be identified and assigned a category-how to implement Some categories (e.g. health details) are required to have higher protection-how? Consent: Asking For/Recording One lawful reason for storing/processing an individual’s data is they have consented How software platforms can ask use for consent+record the granted/denied consent SAR Handling (access request) Data subject may ask for copy of their data Software platforms should be adjusted to electronically accept SARs and respond How do we know individual making SAR is really the data subject? What data to collect (and what format) How to return result via software platform Handling a Data Breach When data breach occurs, inform data controller + data subject (if at risk) Handling an Audit An audit can check for GDPR compliance – how dev/DevOps best prepare for audit Specialist Issues Personal data appearing in log files What happens when employee joins/leaves What happens with new customer Paying attention to data retention policy GDPR Software Project We examine the practical steps needed to GDPR-enable a software platform</p>