Equinor Developer Conference 2018

Day 1 – Tuesday September 25

09.00	**Keynote** **Simon Brown** **Five things every developer should know about software architecture**	
10.15	**Languages**	
	Brainfuck	Markus Fanebust Dregi
	Lisp	Erik Parmann
	Elixir	Jørn Ølmheim
	Haskell	Jørgen Kvalsvik
	Visualising software architecture	Simon Brown
12.15	**Developer Survey QA**	
13.00	Lunch	
14.15	**Contributed lecture**	
	Data engineering in Omnia	Tahir Ali
15.00	**Miniworkshops**	
	Machine learning basics	Kristian Flikka and Eivind Sjaastad
	3D printing	Carsten Falk Hammershøj
	Omnia API workshop	Øyvind Rønne
	Design thinking	Jon Jaatun
	The model-code gap **Visualising software architecture**	Simon Brown
17.30	Leisure and activities	
19.00	Dinner and quiz, delicious tapas and drinks	
21.00	Meet your colleagues	

Equinor Developer Conference 2018

Day 2 – Wednesday September 26

09.00	**Keynote** **Olve Maudal** **Feedback driven development**	
10.15	**Workshops**	
	µPython	Sebastian Roll
	Unreal Engine	Jens Gåsemyr Magnus
	A Tour of Modern C++	Olve Maudal
	Intermediate Python	Sixty North v/ Robert Smallshire
	Radix	Lars Kåre Skjørestad
	Google Design Sprint	Kantega v/ Kristin Wulff
13.00	Lunch	
14.15	**Workshops continued**	
	µPython	Sebastian Roll
	Unreal Engine	Jens Gåsemyr Magnus
	A Tour of Modern C++	Olve Maudal
	Intermediate Python	Sixty North v/ Robert Smallshire
	Agile	Knut Kvarme
	Google Design Sprint	Kantega v/ Kristin Wulff
17.30	Leisure and activities	
19.00	Dinner	
21.00	Meet your colleagues	

equinor

Equinor Developer Conference 2018

Day 3 – Thursday September 27

09.00	Workshops	
	µPython	Sebastian Roll
	Unreal Engine	Jens Gåsemyr Magnus
	Hackathon	
	Intermediate Python	Sixty North v/ Robert Smallshire
	Agile	Knut Kvarme
	Google Design Sprint	Kantega v/ Kristin Wulff
12.15	**DCOE and CIT QA**	Moderated by Kristian Flikka
13.00	Lunch	
14.15	Summary	
15.00	EOC	

(intentionally left blank)

Equinor Developer Conference 2018 detailed program

September 25 — 27

Day 1

Keynote: Five things every developer should know about software architecture

Simon Brown

The software development industry has made huge leaps in recent years; with agile, lean, software craftsmanship, evolutionary design and microservices being just a few of the buzzwords we throw around. Despite this, software development teams are often more chaotic than they are self-organising, with the resulting code being more of a mess than was perhaps anticipated. Successful software projects aren't just about good code though, and sometimes you need to step away from the IDE for a few moments to see the bigger picture.

This session is about that bigger picture and is aimed at software developers who want to learn more about software architecture, technical leadership and the balance with agility. This talk will debunk some of the common myths as we look at five things every developer should know about software architecture; a guide to software architecture on modern software projects that's pragmatic rather than academic and lightweight rather than *"enterprisey"*.

Languages

Introduction to Elixir

Jørn Ølmheim

Elixir is a dynamic, functional language designed for building scalable and maintainable applications. Elixir leverages the Erlang VM, known for running *low-latency*, *distributed* and *fault-tolerant systems*, while also being successfully used in web development and the embedded software domain.

This talk will give an introduction to the language, the interactive execution environment, as well as the scripting and applications modes that Elixir can be used in. This will be an introduction and is intended for beginners, thus no prior knowledge of Elixir or functional programming is required.

Crowdsourced Haskell

Jørgen Kvalsvik

Welcome to a hands-on session where we will crowdsource programs *solving real problems in Haskell*, a wonderful non-strict purely functional programming language. Both Haskell and mob programming is a great way to apply yourself, and to pick up a few refreshing ideas on computation and programs on the way.

It will be an interactive session where we together solve problems and write programs, through discussions and suggestions from the participants.

Bringing a laptop is optional (all programs will be compiled and run on the instructor's computer), but you're free to bring your own computer to try out snippets and programs yourself.

Schemy listy lispy lisps

Erik Parmann

We will look into Lisp in several of its beautiful variants. After this little crash course you should be able to uphold polite smalltalk about some of the different lisps, a bit of their history, and how to program in them. You will be able to make snarky remarks to Pythonists about their lack of macros and how *Python is actually just a poor man's Lisp* without homoiconicity, and you will learn what that word means.

Minimalistic languages and their implications

Markus Fanebust Dregi

We will start this session by giving an introduction to the extremely small language Brainfuck. We will then discuss the technical implications of Brainfuck being Turing complete, as well as ponder over its connection to life, Game of Life, amoebas and emerging intelligence (artifical or not).

To finish off with something concrete we will *pair program Brainfuck*. If you bring your best and work hard, you might be able to add two integers by the end of the session (no kidding).

Bringing a laptop is beneficial. But to be honest, development time will be king. So you will not lose that much time by emulating on paper.

Miniworkshops

3D printing — a practical guide to getting started

Carsten Falk Hammershøj

What is *3D printing* and where do you start?

Which desktop printers exist and what do you need to consider before you purchase one? What materials are needed for printing. Which maker communities for 3D printing are there? What are the opportunities for Equinor?

How do we design for 3D printing? We will see tools and do a practical exercise.

Visualising software architecture with the C4 model

Simon Brown

It's very likely that the majority of the software architecture diagrams you've seen are a confused mess of boxes and lines. Following the publication of the Manifesto for Agile Software Development in 2001, teams have abandoned UML, discarded the concept of modelling and instead place a heavy reliance on conversations centered around incoherent whiteboard diagrams or shallow *"Marketecture"* diagrams created with Visio. Moving fast and being agile requires good communication, yet software development teams struggle with this fundamental skill. A good set of software architecture diagrams are priceless for aligning a team around a shared vision and for getting new-joiners productive fast.

This session explores the visual communication of software architecture and is based upon a decade of my experiences working with software development teams large and small across the globe. We'll look at what is commonplace today, the importance of creating a shared vocabulary, diagram notation, and the value of creating a lightweight model to describe your software system using the "C4 model", which I created as a way to help software development teams describe and communicate software architecture, both during up-front design sessions and when retrospectively documenting an existing codebase.

The model-code gap

Simon Brown

When we're having an architecture discussion, we'll talk about abstractions, using terms like component, module and layer. These abstractions reflect our mental model of a software system, which are useful when describing our architectural ideas and intent. These same abstractions don't typically exist in the programming languages we use though. There's no layer keyword in Java, for example. To implement these abstractions, we need to use a combination of the constructs available in our programming languages; such as classes, interfaces, objects, functions, packages, namespaces, files and folders. In many cases, the code that is written doesn't end up reflecting the model. This is *the model-code gap*.

The model-code gap manifests itself in a number of ways. In obvious cases, the code doesn't match the architecture diagrams. Sometimes the problems are more subtle though. This session is about the model-code gap, and particularly how it relates to applications that are built from a single monolithic deployment unit. Regardless of how we choose to structure our code (layers, vertical slices, ports and adapters, etc), our best design intentions can be destroyed in a flash if we don't consider the intricacies of the implementation strategy. The devil is in the implementation details.

Introduction to machine learning

Kristian Flikka & Eivind Sjaastad

- A very basic *introduction to machine learning*.
- What is it?
- Why do we do it?
- How do we do it (in Python)?

There will be some examples shown, we recommend (but don't require) that you bring your computer with Python installed, so that you can try some things out for yourself.

API workshop

Øyvind Rønne

In this hands-on session we will *create a simple API* in Node.js/Express. We'll discuss design considerations and concepts like naming, versioning, life-cycle management, discoverability, authentication, etc. But mostly we will code and have fun!

Please make sure `Node.js` is installed on your computer beforehand.

Data enginering in Omnia

Tahir Ali

The amount of data collected and analyzed has increased rapidly, which has led to an increase in the demand for skills and tools in data processing. With the growth of both the number and size of big data teams, specialized roles begin to be defined. One of these roles is *data engineer*, which focuses on ensuring that quality data is available for data scientists and analysts to analyze.

This talk will give you an introduction to;

- Data Engineering
- Data engineering project(s)
- How we do data engineering in Equinor
- A demo of Azure Data factory V2

Design thinking

Jon Jaatun

What is Design Thinking?

Jon, from the mobility team, gives us a fun introduction to Design Thinking. With practical examples, first one that you can experience yourself in Lego, and later from him and his team's work for the *field of the future*.

Day 2: Workshops

Keynote: Feedback driven development

Olve Maudal

For any non-trivial project: Software development should be considered a continuous learning process and a cooperative game of communication between professionals. Effective software development can be achieved through frequently repeating cycles of *preparing, changing, observing, reflecting, and learning.*

While the statement above is obvious to many, it is easy to miss the key points. For instance, you must make sure that you facilitate the learning process by implementing effective feedback mechanisms, do frequent iterations and be willing to re-plan the project continuously. You must also implement information radiators, enable osmotic communication, and get rid of things that hinders communication (yes, I am a fan of Alistair Cockburn). But first of all, you must assume that your developers are professionals that know what to do given a vision, trust and enough information. You should certainly not treat your developers as mere resources that need to be directed and told what to do and how to do it.

MicroPython

Sebastian Roll

MicroPython is a Python implementation for embedded processors. This two day workshop will cover the MicroPython language and how to use it for your very own IoT project. Each participant will be provided with a *powerful ESP32 microcontroller* and a wide range of fun components to use. Cooperation is encouraged!

Some examples of what we might create together:

- mp3 player
- remote controlled car
- gamepad
- NFC tag reader
- handheld web server
- MQTT-connected sensor

From the experienced to the aspiring, this workshop should suit everyone. Please bring your laptop.

Unreal Engine

Jens G. Magnus

Let’s make games with Unreal Engine!

This workshop is a combination of a tutorial and hackathon. We will go over everything you need to make simple games.

This includes

- The Unreal Engine Editor
- Programming in Blueprint (Unreal’s visual programming language)
- Important gameplay classes
- Materials

During the tutorial section of the workshop we will learn by introducing new gameplay elements to template games provided by the engine. When we’re done with that it’s time to get creative! We will use what we’re learned to create our own games.

Unreal Engine 4 is a powerful tool. The engine can be modified, extended and scripted with C++; something we might touch upon if there is time. It supports VR/AR with all mainstream hardware.

Workshop requirements:

- Somewhat beefy computer, the editor is heavy (but the standard laptop works)
- Unreal Engine license, it’s free, you just need to register at https://www.unrealengine.com
- Preferably with the Unreal Engine installed
  - Do this before the workshop as the installation is at minimum 7GB
  - If you run Linux, you need to build the engine from source, this requires a newer version of make, clang, and mono.

Intermediate Python

Robert Smallshire

This two day course is designed for developers who already know the fundamentals of Python. This course will get more ”*under the hood*” and introduce the students to powerful tools and techniques that go beyond the basics. There are a lot of intermediate topics in Python, and this course can be customized based on need. The class will focus on Python 3 unless Python 2 is specifically requested.

- We start immediately with working programs.
- Testing is integral to our approach.
- Taught on Windows, Linux or Mac OS X.
- Knowledge level of Python for Programmers course is assumed

Topics

- Function and class decorators
- Closures
- Creating context managers
- Packaging and distribution of Python packages
- Callable objects, lambdas, and extended argument syntax
- Properties, class methods, and static methods
- String representations of objects
- Specialized numeric and scalar types
- Functional-style programming tools
- The iteration and iterable protocols
- Multiple inheritance, method resolution order, and super()
- Collection protocols and implementing collections
- Advanced error handling with exceptions
- Introspection

Computer Setup:

This workshop requires that you bring a laptop with the following software installed:

- Python 3.3 or greater installed.

- An editor for Python code. Attendees can use whatever editor they prefer, but we recommend PyCharm which is a full-featured Python IDE
- The ability to either connect to the Internet or accept USB flash drives so we can distribute course materials on the day.

A Tour of Modern C++

Olve Maudal

In this fast-paced course we will start from scratch and relearn *C++ with modern syntax and semantics*.

Among other things you will learn (at least something) about:

- rvalues and move semantics
- how to write and understand templates
- function objects and lambda expressions
- decltype, auto and type deduction in general
- exception handling and exception safety
- ”mystical” stuff like ADL, RAII and SFINAE
- futures, promises and higher-order parallelism
- concepts and type traits
- iterators, smart pointers and object lifetimes
- using the standard library effectively
- misc *do’s and don’ts in modern C++*
- modern design principles and how to write solid code
- C++11, C++14 and new stuff coming with C++17 and later

This course is aimed at experienced programmers that would like to learn how to write, or at least understand, modern C++. Ideally you should have some experience with either C, old-school C++, Python and/or Java.

Google Design Sprint

Kristin Wulff

Get to know a practical, and *time boxed implementation of Design Thinking* from Google Ventures. The methodical five days approach, forces the team to explore a vast space of ideas, and assist you in quickly narrowing them down to the one you really want to test on your users. An agile approach to the design phase, that allows your team to fail fast and shortcuts the idea$\rightarrow$learning feedback loop. Do you think we could be more curious and creative in the earlier stages of our projects. Do you think that our projects are agile in name, but not always in practice. Or have you at some point simply experienced that the code you were writing, was an excellent answer, but sadly for the wrong question. Then join in for three$\times$three hours of fun, a new perspective on the design phase, and a pocket full of ideation activities and practices for you and your team.

Primarily for: Everyone

Participant requirements: Their head and hands.

Radix half day Workshop

Lars Kåre Skjørestad

TODO: Insert description here!!

Agile Workshop

Knut Kvarme

TODO: Insert description here!!