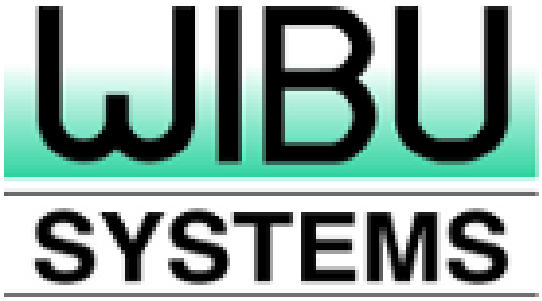
ACCU Conference 2012

Conference Sponsors



Conference Organization

Conference Chair

Jon Jagger

Conference Committee

Astrid Byro
Francis Glassborow
Olve Maudal
Roger Orr
James Slaughter

Event Manager

Archer Yates Associates Ltd
Threshers Yard, West Street
Kingham, Oxon. OX7 6YF
Phone: +44 (0) 1608 659900
Fax: +44 (0) 1608 659911
Email: julie at archer-yates.co.uk

Schedule

Tuesday 24 April (Pre Conference Tutorial Day)

10:00	The C++11 Standard Library - New Features, new Tools, new Patterns, new Pitfalls Nico Josuttis	The Art of Garbage Collector Tuning (for the SUN/Oracle HotSpot JVM) Angelika Langer, Klaus Kreft	Advanced IT Design and Architecture Tom Gilb
17:00			

Wednesday 25 April

Room	University	Cherwell	Blenheim	Charlbury	Wolvercote
09:30	Project Patterns: From Adrenalin Junkies to Template Zombies Tim Lister				
10:30	Coffee break				
11:00	The best & worst new features of the C++11 Standard Library Nico Josuttis	Git in the enterprise Charles Bailey	Lambdas in Java 8 Angelika Langer	What is code simplicity? Arjan van Leeuwen	Business patterns for software developers Allan Kelly
12:30	Lunch WIBU Systems symposium 13:00 - 13:45 (**University**)				
14:00	C++11 for everybody Dietmar Kühl	Parallel Architectures Detlef Vollman	Unit testing and mocking in C++ Ed Sykes, Rajpal Singh	Ericsson Finland Agile transformation Henri Kivioja, Henrik Taubert	The Victorian Exploring Expedition of 1860: reflections on project management and leadership from a real-life deathmarch project Jim Hague
15:30	Coffee break				
16:00	Plenty People Programming with the C++ 11 Standard Library Nico Josuttis	Common objections to TDD and their refutations Seb Rose	An introduction to Userspace Filesystem Development Matthew Turner	Database development using TDD Chris Oldwood	Surprise Surprise
17:30	Break				
18:00	Lightning Talks organized by Ewan Milne				
19:00	Break				
19:30	Atlassian Welcome Party				

Thursday 26 April

Room	University	Cherwell	Blenheim	Charlbury	Wolvercote
09:30	The Congruent Programmer Phil Nash				
10:30	Coffee break				
11:00	C++11 for the rest of us. Simpler code with more power - Part 1 Peter Sommerlad	Go, D, C++ and the Multicore revolution Russel Winder	Developing a walking skeleton using TDD Paul Grenyer	Lean Quality Assurance: Much more cost-effective Quality Assurance methods than Testing Tom Gilb	DSLs from the perspective of extensible languages Didier Verna
12:30	Lunch Electric Cloud symposium 13:00 - 13:45 (**University**)				
14:00	Algebraic data types for C++ Andrew Stitcher	How to write a testable state machine Matthew Jones	Testing a la carte Klaus Marquardt	Real Teams Henrik Berglund	Is C going the way of the Dodo? Dirk Haun
14:45	When only C will do Andrew Stitcher	Groundhog day of a team leader Zsolt Fabok		Is Scrum incompatible with your brain? Henrik Berglund	Open Source is good for you Dirk Haun
15:30	Coffee break				
16:00	C++11 for the rest of us. Simpler code with more power - Part 2 Peter Sommerlad	Dataflow, actors, and high level structures in concurrent applications Anthony Williams	Making Jenkins better Jez Higgins	Refactoring to functional Andrew Parker, Julian Kelsey, Steve Freeman	Surprise Surprise
17:30	Break				
18:00	Lightning Talks organized by Ewan Milne				
19:00	Break				
19:30	Red Gate Software XBox Kinect Challenge (Hotel bar)				

Friday 27 April

Room	University	Cherwell	Blenheim	Charlbury	Wolvercote
09:30	Requiem for C Robert Martin				
10:30	Coffee break				
11:00	Functional programming you already know Kevlin Henney	C++11 allocators Jonathan Wakely	TDD inAssembler Olve Maudal	The impact of virtualization on software architectures and lifecycles - A practical approach Arno-Can Uestuensoez	Devops, Infrastructure-as-code and Continuous Deployment Gavin Heavyside
12:30	Lunch Red Gate Software symposium 13:00 - 13:45 (**University**)				
14:00	SOLID deconstruction Kevlin Henney	Building Generic Libraries in C++11 with Concepts Andrew Sutton	Version control done right Pete Goodliffe	"If it was hard to write it should be hard to read" - some musings on code readability and re-use Roger Orr	Agile anti-patterns! Yes your agile project can and will fail too! Sander Hoogendoorn
15:30	Coffee break				
16:00	C++11 pub quiz Olve Maudal	Complexity Thinking? Or Systems Thinking++? Jurgen Appelo	An Introduction to Scout, a Vectorizing Source-to-Source Transformator Olaf Krzikalla	'Date::IsBusinessDay()' Considered Harmful! John Lakos	Surprise Surprise
17:30	Break				
18:00	Lightning Talks organized by Ewan Milne				
19:00	Break				
20:30	Conference dinner				

Saturday 28 April

Room	University	Cherwell	Blenheim	Charlbury	Wolvercote
09:30	Lightning Keynotes organized by Ewan Milne				
10:30	Coffee break				
11:00	C++11 concurrency tutorial Anthony Williams	Embedded Linux Overview Detlef Vollman	API usability: what it means and why you should care Giovanni Asproni	From Crappy to Classy: Legacy Code Resuscitation Gil Zilberfeld	Variance in Generic Types in Java and C# Robert Stanforth
12:30	Lunch				
14:00	C++ Dataflow Parallelism sounds great! But how practical is it? Let's see how well it works with the SPEC2006 benchmark suite Jason McGuinness	Coding Dojo Emily Bache	UMLGraph and the Declarative Drawing of Diagrams Diomidis Spinellis	C will live forever but in the 21st century it looks like this... Bernhard Merkle	Surprise Surprise
15:30	Coffee break				
16:00	TDD for Embedded C James Grenning	Getting into git Pete Goodliffe	The importance of readability in code Michel Grootjans	Ethics and professionalism. The current state of affairs Giovanni Asproni	Paralysis by Parallelism Russel Winder, Schalk Cronje
17:30	Wrap up				

Preconference

The C++11 Standard Library - New Features, new Tools, new Patterns, new Pitfalls (Nico Josuttis)

One day

C++11, the new C++, is a new approved ISO Standard now and it will have an enormous impact on C++ application programmers.

New libraries but also new language features will provide more features, give more power, and, to some extent, change the way you program in C++. But these new features also introduce new challenges, new pitfalls, and new surprises.

In this tutorial Nicolai Josuttis, the author of "The C++ Standard Library" will present issues you should know about the new Standard from the point of view of an application programmer using the C++ Standard Library. The material will focus on all unexpected and remarkable features he found out while writing the second edition of his library book.

The Art of Garbage Collector Tuning (for the SUN/Oracle HotSpot JVM) (Angelika Langer, Klaus Kreft)

One day

The HotSpot JVM developed by Sun (now owned by Oracle) has been refined and revised with every release of the JDK since the advent of Java in the mid 90ies. Today, Java developers face an abundance of GC algorithms - from plain and simple serial stop-the-world collectors with a single reaper thread to highly parallelized collectors that run several GC threads concurrently with application threads. Each of these collectors can be configured and tuned in various ways in order to control pause times or increase throughput. The number of choices a Java developer has for configuring the JVM's garbage collection for his application is overwhelming. Hence, garbage collector tuning for the SUN/Oracle JVM is a daunting task. The workshop aims to shed light onto the garbage collection strategies in the Sun/Oracle JVM by first explaining all algorithms (including Java 7's "G1" collector). We will then take a look at the collectors' tuning parameters and will practice GC tuning in a hands-on session using various tools. Attendants are asked to bring their notebooks with a Java 7 SUN/Oracle HotSpot JVM. Instructions for installation of additional tools will be provided in due time. Basic knowledge of Java is required; substantial knowledge of garbage collection is not required.

Advanced IT Design and Architecture (Tom Gilb)

One day

Intended for: People who have some design or architecture practice and responsibility today, and who want some new and powerful tools for designing. This is the beginning of a study leading to competence as IT Design Engineer, or IT Architectural Engineer.

Background: Most IT design and IT Architecture is currently based on non-engineering paradigms. There is almost total absence of quantified requirements for performance, qualities and costs, as a design basis. There is as good as no practice of estimation and measurement of the multiple impacts of a design or architecture on these requirements. In short, design is practiced in an intuitive manner. Nice words but no justification or responsibility. This is highly unprofessional and is damaging to our community. This course will expose, and make freely available, the set of tools necessary to practicing real IT Design/Architecture Engineeering.

Syllabus:

- Overview Lecture: "Real Architecture"
- Quantified Requirements as Primary Design Drivers: Planguage as a basic requirements language.
- Specification Quality Control: Numeric evaluation of requirements and design specifications.
- Design (Architecture) Specification: Advanced levels of detail as prerequisite for evaluation.
- Design Impact Estimation: on multiple design drivers of quality and cost.
- Evo: Getting early and continuous feedback on design attributes.
- Dynamic Design to Cost: how to meet performance and quality targets within budgets.

and deadlines by dynamic learning and adjustment.

Limitations: this one day course cannot train most people to be design engineers. However, the best participants will have the basis for beginning the advanced practice and teaching themselves more. We will be very concrete about the methods, the case study practices, and give free access to deeper textbook material. Your awareness of real IT design engineering will be dramatically raised. Hopefully you will wish to continue these studies and practices.

Keynotes

Project Patterns: From Adrenalin Junkies to Template Zombies (Tim Lister)

60 mins

We all talk about “best practices” but a tiny minority of organizations actually practice them all. But not to worry, think of "best practices" for human health. We know all about them, but very few of us actually practice them all. Maybe if someone did arduously practice all health practices they would forget to have a life. Tim has come to believe that project patterns are stronger than best practices. They are the habits, the decision practices, and the corporate culture, the unstated rules, which dominate office life. The first key is to identify your own organization’s patterns. If they are positive, how can you perpetrate them across all projects? If they are negative, how can you break the habit? Tim will start the talk with some examples from the book project. He will then let the audience offer up some of their own patterns.

The Congruent Programmer (Phil Nash)

60 mins

Why do we do what we do? Are we doing what we want to do? The way we want to do it? How can we use the answers to these questions to get better at whatever it was we were doing in the first place? We don't work in a vacuum. The way we fit all the pieces together matters and can make a difference. We spend a lot of time designing how our code fits together. It's time we did the same to ourselves.

Requiem for C (Robert Martin)

60 mins

For nearly five decades, it's been C. Regardless of what else has been tried, the new language paradigms that have been explored, the new semantics that have been probed, C was at the heart of the successes. C++, Java, C#, Ruby, all took their cue from C. Indeed C's dominance has been so great that years ago Dick Gabriel was moved to declare that C was the last programming language. But all things end, and so it is for C. It's ascendancy as passed, it's demise is accelerating, and the end is near. Alas, for C, we knew it well.

Sessions

Agile anti-patterns! Yes your agile project can and will fail too! (Sander Hoogendoorn)

90 mins

The popularity of agile software development processes and methodologies is imminent and fast growing. Many organizations and projects turn towards agile to help solve the problems of traditional software development. Scrum, extreme programming, test driven development, and lean are no longer the new kids on the block. However, with the rising popularity of agile, mainly due to lack of experience, or management over-expecting results, the coming years many agile projects will fail miserably. Agile is not the silver bullet.

In his enthusiastic style speaker Sander Hoogendoorn, global agile thought leader at Capgemini and involved in agile projects since the mid-nineties, will demonstrate the differences in traditional and agile projects, and show why agile projects will fail – independent of the process used. Sander will elaborate on a series of agile anti-patterns that people will recognize immediately. Think of the Scrumdamentalist, Agile-In-Name-Only, the Pseudo-Iteration, Guesstimation, the Bob-the-Builder Syndrome, Parkinson's Law, the Agile Project Manager, and Student Syndrome. Of course with many embarrassing examples and anecdotes from real-life projects.

Algebraic Data types for C++ (Andrew Stitcher)

45 mins

With the ratification of C++11, C++ has a few more features that are usually found in functional programming languages (lambdas, and "auto" type inference for example).

Algebraic data types are a powerful feature of many functional languages that can be thought of as a combination of C or C++ style unions and structs - according to some they are "unions done right". When combined with pattern matching constructs they enable a very efficient and expressive programming style.

We'll explore some ways to implement both algebraic data types and their matching operations in C++ and the trade-offs involved.

An Introduction to Scout, a Vectorizing Source-to-Source Transformator (Olaf Krzikalla)

90 mins

We present Scout, a configurable source-to-source transformation tool designed to automatically vectorize C source code. Scout provides the means to vectorize loops using SIMD instructions at source level. The tool vectorizes a wide range of loop constructs and can target various modern SIMD architectures. The respective SIMD instruction set is easily configurable by the user.

Scout has become an industrial-strength vectorizing preprocessor and it is used on a day-to-day basis in the software production process of the German Aerospace Center. The tool is published under an Open Source license and available via http://scout.zih.tu-dresden.de

I start the presentation with a short comparision of the tool-based and library-based approach respecting vectorization. After that I introduce Scout and describe its capabilites and the underlying technology. I demonstrate the basic usage of the GUI and the integration of the CLI tool in the make process. Then I explain the vectorization of various forms of loops by example using the GUI. This includes the handling of conditions, partial vectorization, outer-loop vectorization, register blocking, handling of C++ code aso. In addition, I take a closer look at Scouts' configurability, as well as examine the ways in which it can be utilized to define gather/scatter operations, and to vectorize complex idiomatic expressions (e.g. MIN, MAX). I conclude with a presentation of achieved performance results and talk about issues limiting the expectable speedups of SIMD vectorization.

Reference: Krzikalla, O., Feldhoff, K., Müller-Pfefferkorn, R., Nagel, W.: Scout: A Source-to-Source Transformator for SIMD-Optimizations. In: 4thWorkshop on Productivity and

Performance (PROPER 2011). Bordeaux, France (August 2011), accepted for publication.

An introduction to Userspace Filesystem Development (Matthew Turner)

90 mins

Writing a filesystem can be very cool. Alas, writing a filesystem is also very hard. This is mainly because coding in the kernel is hard. Thankfully, most of the pain can be avoided by using a library like FUSE. Such libraries enable filesystems to be expressed as simple userspace programmes by taking care of all that tedious mucking about in kernel space.

This talk will look at the Why and How of such filesystem development, using FUSE on UNIX. The talk will be very practical, with code on the screen and maybe even written in front of your very eyes (if I'm feeling brave).

There will be a short recap of how the VFS works on UNIX and then we'll dive into writing a filesystem with FUSE.

I'll go over my experiences of developing such filesystems - architectural patterns, testing, performance, etc. There will also be a section on the behaviours and gotchas of the libraries involved.

API usability: what it means and why you should care (Giovanni Asproni)

90 mins

Programmers, explicitly or implicitly, when working on complex systems, end up designing some APIs to accomplish their tasks, either because the product itself is some kind of general purpose library or because they need to write some libraries and packages to put some common code of their applications.

There is plenty of information available about how to write clean and maintainable code, but not a lot about writing usable APIs. The two things are related, but they are not the same. In fact, clean code is code that is clean from the point of view of its maintainers, usable APIs, on the other hand, refer to code that programmers (other than the original author) find easy to use. We'll see how usable APIs help in writing clean code (and vice-versa).

In this session I will introduce the concept of API usability, explain its importance--e.g., impact on productivity and defects--and show its relation with clean code, as well as some (sometimes surprising) research results from literature. I will also give some practical advice on how to start writing more usable APIs.

Building Generic Libraries in C++11 with Concepts (Andrew Sutton)

90 mins

Every working generic library is inherently based on concepts: requirements on template arguments that specify its intended use within the template. These concepts may be represented in the library documentation, comments in the source code, specialized programming frameworks, or (as is too often the case) simply in the head of the programmer. Unfortunately, concepts cannot be directly expressed in the C++11 programming language.

In this talk, we look at the design and implementation of Origin, a collection of generic libraries designed specifically for C++11. Over the past five years, Origin has served as a testing ground for exploring new programming idioms and ideas using the emerging C++11 programming language. More recently it is being used to evaluate the design of concept libraries for real programs.

In particular, this talk will focus on the Origin Concept library, which provides facilities for defining concepts and using them to check type requirements and implement concept-based overloading. We will also discuss the semantic aspects of concepts and how they can be used in testing. We will use Origin's Algorithm, Iterator, and Range libraries as examples of what the next generation of C++ generic libraries could look like.

Business patterns for software developers (Allan Kelly)

90 mins

Patterns, patterns everywhere. Patterns exist not just in the software code but around the code. In the same way patterns can explain software design they can explain business design. Some patterns reappear again and again in software companies big and small. To the entrepreneur, or growing business, these patterns offer an opportunity to learn from others. Software architects can benefit too by better understanding the business environment the software exists within.

In this talk Allan Kelly will describe some of the patterns in his new book, show how they connect together in pattern sequences and show how budding software entrepreneurs can make use of these patterns. He will also preview some related patterns not found in the book.

C will live forever but in the 21st century it looks like this... (Bernhard Merkle)

90 mins

There is nothing like C when it comes to the bare metal and real embedded development. It is still one of the most popular languages and unlikely to be replaced in the near future. However there are quite a few problems in the language and especially for developers in the embedded area. Problems are: missing encapsulation concepts, unsafe operation and types, no physical units and quantities, no support for common concepts in the embedded area like tasks, messages or state-machines. In this session I will show how it is possible to build modular languages which special emphasis for developing software for embedded systems. (The principle however can I domain independent).

We show how to extend the C Programming Language e.g. we will add: real module/encapsulation concepts, support for interface/implementation and component base development. Furthermore typesafe physical units and quantities as a C language extension will be shown.

Embedded systems often support state-machines so there will be direct support for programmes with states, triggers, events and actions as first level concepts. Also different syntax (like textual, graphical, tabular) can be mixed here.

I will present the power of modular languages and illustrate and show the idea with an Open Source Projectional Language Workbench MPS in a real-world embedded software development scenario. Modular languages and Projectional workbenches enable us to grow and extend languages easily.

Despite the usual drawbacks of internal DSLs we will build "first class" language extensions with full IDE comfort and debugging support. Combined with the tool support, it is really easy to combine modular languages based on the user needs and build convenient IDE support at the same time.

The samples will run on a Lego/NXT robot, running a OSEK/RTOS with C as programming language. In the talk I will demonstrate the new approach, while in the tutorial the idea is that people build themselves new language extensions a try them out with the Lego/NXT robot or a similar embedded real device.

C++ Dataflow Parallelism sounds great! But how practical is it? Let's see how well it works with the SPEC2006 benchmark suite (Jason McGuinness)

90 mins

Maximizing the performance of various types of software on increasingly large multi-core architectures means programmers could have to exploit the exposed thread-level parallelism using the threading API of the OS, which has been and still is non-trivial. Considerable effort has gone into investigating alternative models for expressing parallelism in code, which have been commonly investigated by their effect upon the performance of the SPEC2006 benchmark suite. Of these techniques, the dataflow model, such as that expressed in C++11, just::threads, boost.threads amongst others, has re-emerged as an interesting technique.

A well designed dataflow library could overcome these issues, which should:

- Provide efficient parallel algorithms.
- Guarantee, via library design, that race conditions and deadlocks would not be present.
- The program would execute an efficient parallel schedule due to the design of the library.
- Address the issues of debugging such parallelised programs.
- Co-exist with other thread libraries such as OpenMP or Intel Thread Building Blocks.

This presentation will examine how successful was the application of a data-flow model of parallelism in C++ to the SPEC2006. This model has been implemented in a library called (Parallel Pixie Dust), that the author created.

C++11 allocators (Jonathan Wakely)

90 mins

The new C++11 standard revised the allocator model used by the containers in the standard library. This talk will start with a recap of C++ allocators and their limitations, look at what has changed, why those changes were made, and whether allocators are any more useful in C++11 than "that last template parameter that everyone ignores".

C++11 concurrency tutorial (Anthony Williams)

90 mins

In this tutorial-style session, I will introduce the C++11 concurrency library in depth, with live practical examples. This is for both those who've never used concurrency before, but want to start, and those who have used other concurrency libraries but want to learn how to take advantage of the C++11 library. I will cover the details of the library, along with practical guidelines about how best to avoid deadlock, race conditions and other concurrency problems.

C++11 for everybody (Dietmar Kuhl)

90 mins

C++ 2011 is now officially adopted and compiler vendors are adding support to implement the standard. There are many changes to make programming with C++ more effective. This presentation concentrates on the many features useful in everyday programming, excluding changes primarily targeted at better support for generic programming: although templates are certainly part of my usual toolbox it seems many people stay away from them. There is still plenty to talk about: constexpr, final, overridden, defaulted, or deleted functions, inherited or delegated constructors, strongly typed enums, new integer and character types and their literals, uniform initialization, r-values and move semantics, noexcept declarations and expressions, automatic types, decltype, new-style function declaration, and lambda functions, and range-based loops. Each one of these changes is intended to make C++ simpler in some form although the mere presence of them makes C++ bigger. Most of these extensions can be used in isolation i.e. you can choose to only use those parts which make your life easier. The presentation will introduce the various features with their objectives and explaining how to use them. Where applicable it will point out known pitfalls. Knowing about the many new aspects increases the toolbox available allows you to make an informed choice of what you want to use. C++ 2011 is coming: get ready to take advantage of it!

C++11 for the rest of us. Simpler code with more power - Part 1 (Peter Sommerlad)

90 mins

With the publication of the new C++11 ISO standard one might ask what that will mean to current or past skills in that language. In addition one could recognize that even Microsoft leans back to use native C++ instead of the .NET languages for some new developments to get more power from smaller hardware.

This tutorial will show how to employ C++11's new features and some already existing ones to write simpler and faster code to the one you might have been used to write in C, C++, C#, or Java.

It especially addresses C++-ish means of abstraction that go beyond the classic OO-style of programming with virtual member functions and inheritance. If you are already using STL algorithms instead of loops, are familiar with lambdas, functors, function binders, async and futures, and know about universal initialization syntax, (variadic!) templates, rvalues and the new meaning of auto, this tutorial might not bring you many news, except for the fun of solving puzzles on your own employing all that stuff during its practice sessions.

As a side effect, if you employ Eclipse CDT for your exercises you might try some new features for TDD, unit testing and refactoring developed by Peter's students and assistants.

Peter will be on the way of preparing his new C++11 book and lecture when ACCU will take place, so expect to be his guinea pig for some of the exercises he intends to make his students solve from fall 2012 on.

Target Audience
Programmers with an interest in C++11, especially those who are more familiar with OO-programming as it is given in Java,

(old) C++, or (simple) C# than the more functional style allowed by templates and the STL.

C++11 for the rest of us. Simpler code with more power - Part 2 (Peter Sommerlad)

90 mins

See above

C++11 pub quiz (Olve Maudal)

90 mins

Join us for a pub quiz on C++. You will be working in groups where I present interesting code snippets in C++ and you will discuss, reason about and sometimes need to guess what the code snippet will print out. There will be many educational snippets where we elaborate on the basics of C++, but some of the snippets will be really hard with surprising answers and where we explore the dark and dusty corners of the language. There will be several C++11 based questions. The motivation for doing the quiz is to learn from each other while having fun. Solid experience with C++ is essential and some knowledge of C++11 is useful.

Coding Dojo (Emily Bache)

90 mins

The coding dojo is a place coders retreat to in order to practice their coding skills. You can play with new tools and languages, or concentrate on improving your practice in a specific area. In all cases, we aim for a safe, collaborative environment where we learn from each other.

Can you refactor? In really small steps? Can you turn some, frankly, ugly code into a paradigm of elegant, readable, extensible design? In this dojo, we'll give you the chance to do some deliberate practice. You'll also get to try out "text-based" testing - a technique I've found particularly helpful when refactoring legacy code.

I've got some slightly hairy but working code for you to improve, with versions of it in various programming languages, (Java, Ruby, Python, C# and C++), so you can choose one you're comfortable with. (See it on github). I'll go through the problem and how to run the tests before we start coding. We'll work in pairs, refactoring and cleaning the code. The idea is not to re-write it from scratch, but rather to practice taking small steps, running the tests often, and incrementally improving the design.

We'll spend the last part of the session on a retrospective. Amongst other things we'll discuss the designs people have ended up with, and how the choice of programming language affects things. We'll also discuss text-based testing and its applicability to legacy code like this.

If you're going to get the most out of the session, it would help to install some things in advance, (or plan to pair with someone else who has). Firstly you'll need a development environment for the language(s) you'd like to code in. Secondly, the tests I'll provide will be text-based, and you'll find it convenient to use the tool "TextTest" to run them. There are installation instructions on http://texttest.org

Common objections to TDD and their refutations (Seb Rose)

90 mins

This is not a session about how or why to practice TDD. Based upon research conducted, I will outline the most common objections to TDD and describe in detail, with examples, how to refute, avoid or mitigate each of them. The coverage will acknowledge that there are risks inherent to all techniques and will not promote the idea that TDD is some kind of silver bullet.

The session will combine the formal presentation of slideware with active engagement of attendees to provide further objections and to contribute to the discussion of how to maximise the value that TDD (and automated unit testing in general) can deliver.

Complexity Thinking? Or Systems Thinking++? (Jurgen Appelo)

90 mins

People have been using the term Systems Thinking for a few decades. But nowadays we sometimes hear the term Complexity Thinking. Some claim that comparing complexity thinking to systems thinking is like comparing Einsteinian physics to Newtonian physics. Others claim that complexity thinking is nothing more than systems thinking in a fashionable jacket.

In this session we discuss using models (metaphors, analogies, mathematics, patterns) to try and make sense of the world. We have a look at complexity theory and we will see that it doesn't go well with the traditional scientific method. In social systems, such as software projects, the standard approaches of reductionism, prediction and control don't work very well. Even systems thinkers make the mistake of trying to apply the traditional scientific method to social systems.

It is better to use complexity thinking, which is all about complex systems, complexity absorption, diversity, narratives, context, reflexivity, exploration, uncertainty and interaction.

And what does this all have to do with Lean and Agile software development? Do theoretical debates on terminology really help us to better manage our business? The answer is yes. Because "There is nothing as practical as good theory" (Kurt Lewin) and "Without theory there is no learning." (John Seddon).

Database development using TDD (Chris Oldwood)

90 mins

The modern day RDBMS is a complex product that offers so much more than just data persistence. The SQL language, with its vendor specific variants such as T-SQL, provides the ability to develop code in various forms to read, transform & write that data efficiently. This code requires constant testing right from its inception through its various incarnations until it is finally retired.

TDD is a technique that promotes writing those tests at the front of the development process, whether that be because you're writing new code or changing existing code. The knock-on effect of this approach is that your client-based perspective opens your eyes to potential variations in the implementation, and that is where the second 'D' in TDD turns from Development into Design. With a solid automated test suite and Continuous Integration under your belt too the world of refactoring opens itself up so that your database design can safely evolve.

This session looks at applying the same principles and disciplines used in other areas of system development to tame the ever increasing complexity that has arisen from the maturity of the RDBMS.

Dataflow, actors, and high level structures in concurrent applications (Anthony Williams)

90 mins

In this session I will give an overview of various high level approaches to concurrency, including dataflow architectures and actor libraries. Examples will be drawn from several languages and platforms, including C++ and Groovy

'Date::IsBusinessDay()' Considered Harmful! (John Lakos)

90 mins

Professional developers create software in response to perceived business needs. All too often, the stated requirements come with extra baggage that would unreasonably constrain proper factoring of the problem and therefore its solution: "Write me a 'Date' class that tells me whether today is a business day." Given such malformed specifications, it is incumbent on developers to distil the true requirements.

Thoughtful, fine-grained factoring of software offers many important practical advantages. Individual components, being small and focused, often fall into one of a handful of familiar class categories, making them easier to understand, document, test, and maintain. Packaged properly, these carefully crafted, stable sub-solutions can be reused individually and collectively to address other business needs without modification, leading to a powerful form of reuse that we call "hierarchical reuse".

In this talk, we begin by reviewing our component-based development methodology, including basic physical design concepts. We then proceed to analyze the valid business need of efficiently implementing the substantial machinery necessary to address the domain of dates and business days alluded to above. Along the way, we apply our knowledge of physical design and fine-grained factoring skills to arrive at a fully-factored solution -- distributed across many distinct components having acyclic physical dependencies -- in which each component is easy to understand and use, and the resulting collection of stable components is also hierarchically reusable.

Developing a walking skeleton using TDD (Paul Grenyer)

90 mins

Starting with an (almost) clean IDE Paul will develop a Walking Skeleton. The walking skeleton was described by Alistair Cockburn as "... a tiny implementation of the system that performs a small end-to-end function. It need not use the final architecture, but it should link together the main architectural components. The architecture and the functionality can then evolve in parallel." It is also one of the theme's in Freeman & Pryce's Growing Object Orientated Software Guided by Tests.

In this session Paul will start with an (almost) clean IDE and develop a walking skeleton for a simple application and demonstrate how Test Driven Development (TDD) can be used even at the system level to test features.

Devops, Infrastructure-as-code and Continuous Deployment (Gavin Heavyside)

90 mins

How often do you deploy new versions of production code? Once per quarter? Once per month? Just as agile software development methodologies bring short iterations and always-working code, Devops methodologies and techniques such as infrastructure automation and continuous deployment can enable a business to release production code as often as several times per day. Frequent releases of incremental functionality mean lower risk at each release, and functionality being delivered to customers more quickly.

The DevOps movement is changing the way tech companies manage their infrastructure. Code is put into production faster and more safely by breaking down the barriers between Developers and IT Operations.

To achieve frequent safe releases one of the key things you need is automated, repeatable infrastructure. Open source tools like Chef and Puppet enable scripted, testable, repeatable server configuration every time, on physical hardware and virtualised or cloud services. Checking your infrastructure configuration into source control brings reproducibility, traceability, repeatability, brings developers and operations closer together

In this talk we will discuss the principles, methods and practices for achieving a state of continuous deployment, including developer testing, CI, integration testing and infrastructure. We will see how automated configuration management tools such as Chef can enable teams to accelerate their deployments and launch new services in minutes. I will highlight specific examples from my own experience of building a robust, automated data collection and analysis service for the insurance industry, using Chef and AWS in a startup.

DSLs from the perspective of extensible languages (Didier Verna)

90 mins

Creating a domain-specific language (DSL) is inherently a transversal activity: it usually requires knowledge and expertise in both the application domain and in language design, two completely orthogonal areas of expertise. The difficulty is that either one needs to find people with such dual competence, which is rare, or one needs to add manpower to the project by means of different teams which in turn need to coordinate their efforts and communicate together, something not easy to do either.

One additional complication is that being an expert developer in one specific programming language does not make you an expert in language design -- only in using one of them. DSLs, however, are most of the time completely different from the language in which the embedding application is written, and a general-purpose programming language (GPL), suitable to write a large application, is generally not suited to domain-specific modeling, precisely because it is too general.

As a consequence, it is often taken for granted that your application's DSL has to be completely different from your application's GPL. But what if this assumption was wrong in the first place ?

The need for designing a DSL as a completely new language often comes from the lack of extensibility of your GPL of choice. By imposing a rigid syntax, a set of predefined operators and data structures on you, the traditional GPL approach gives you no choice but to implement your application's DSL as a different language, with its own lexical and syntactic parser, semantic analyzer and possibly its own brand new interpreter or even compiler. A much less widely accepted view, however, is that some GPLs are extensible, or customizable enough to let you implement a DSL merely as

either a subset or an extension of your original language. The result is that your final application, as a whole, is now written in a completely unified language. While the end-user does not have access to the whole backstage infrastructure and hence does not really see a difference with the traditional approach, the gain for the developer is substantial. Since the DSL is now just another entry point for the same original GPL, there is essentially only one application written in only one language to maintain. Moreover, no specific language infrastructure (parser, interpreter, compiler etc.) is required for the DSL anymore, since it is simply expressed in terms of the original GPL. The already existing GPL infrastructure is all that is needed.

The purpose of this presentation is to envision the process of DSL design and implementation from the perspective of extensible GPLs, and to show how the capabilities of the original application's GPL can have a dramatic impact on DSL conception. More precisely, the objective of is twofold:

1. showing that by using a truly extensible GPL, implementing a DSL is considerably simplified, to the point that it may sometimes boil down to writing a single line of code,

2. exhibiting the most important characteristics of a programming language that make it truly extensible, and hence suitable for DSL'ification. Such characteristics are most notably dynamicity (not only dynamic typing, but in general all things that can be deferred to the run-time), introspection, intersession, structural or procedural reflexivity, meta-object protocols, macro systems and JIT-compilation.

Embedded Linux Overview (Detlef Vollman)

90 mins

More and more embedded systems grow out of a simple self-written basic system and require a full OS. Linux is a rising star in this domain and is used in many completely different kind of systems. But an embedded Linux is not exactly the Linux you know from your PC. Embedded systems have their constraints, and they have different requirements in terms of hardware (memory types, devices), functionality and reliability. As many embedded systems have realtime requirements, but Linux per se is not a realtime OS, there exist various realtime extensions for embedded Linux.

This talk will present all pieces that make up an embedded Linux system (from bootloader to watchdog, from interrupt service routine to application). It will present the various approaches to target specific requirements of embedded systems like flash memory, MMU-less micro-controllers, space restrictions and realtime response times. And it will discuss some broader topics like build systems, software updates and support and legal aspects. This talk goes not into technical details, but presents an overview of many aspects of using Linux in embedded systems.

Ericsson Finland Agile transformation (Henri Kivioja, Henrik Taubert)

90 mins

Ericsson Finland started the Agile transformation in 2008 with the first Scrum Team. Since then we have scaled up to 30+ teams and set up a complete e2e organizational setup supporting Agile. This transformation has been (and still is) a profound change in organizational thinking and culture. We are now delighted to share our experiences and learnings from our journey also externally. This is a mature presentation that has already been presented in different conferences with success.

After receiving good feedback from many sessions we have decided to make this presentation even more interesting by adding different dimensions into it. We would like to present this change with real persons from different viewpoints (2-4 persons). This will add interaction and more energy into the presentation.

Organizational view
How agile change is seen in organizational context. High level goals mapped into individual and team context. How new product development is seen in mature organization.

Developer view
What are the essential skills for a C++ developer. How to ensure and improve the pace when moving into new problem domains. How does agile and scrum change the needed skillset for a developer. What changes in the toolset have been necessary to support the agile way of developing software..

Coach/Team view
What is needed to make the change smooth in teams, individuals and organization. How do I need to change to make the change happen. How do the team/coach make sure the full potential of the team(s) is utilized.The importance of really having well-balanced cross functional teams. How to turn the

pain of tool and environment problems into something positive.

Product view
What challenges does 10 Years legacy bring to the picture.
What are the telecom characterictics that need to be taken into account. What is needed from testing.

Ethics and professionalism. The current state of affairs (Giovanni Asproni)

90 mins

We know how important software is in the modern world. Almost every thing we use nowadays has some software running inside.

However, writing code is not a profession--there aren't laws (in most countries) that require training or any formal qualification to be allowed to do that--and, as a result, programmers are not bound to abide to any code of ethics, and are rarely legally responsible for damages caused by their code.

Also, according to some, software projects are riddled with unethical behaviour (see http://drdobbs.com/architecture-and-design/229402654 and the "The Dark Side of Software Engineering" book by Robert Glass).

This session aims at exploring this subject further, at shedding some light at what the current situation really is, and at proposing some possible ways to alleviate the problems we are facing.

From Crappy to Classy: Legacy Code Resuscitation (Gil Zilberfeld)

90 mins

Many times we're handed someone else's code. "It's working code; the last guy who worked on it said so", says our boss. Then he leaves the room. After close examination, we conclude there isn't a single test in sight. No proof it works, or a safety net to make our changes. Just "working" code.

As craftsmen we vowed never to write crap. We intend to try to do what we can to turn this very smelly piece of code to a rose garden. But where do we start? How do we proceed?

This is not a theoretical refactoring session. It's hands-on code resuscitation. We're going to take a "working" piece of C++ code, and carve it, fix it, extract it, test it and add functionality.

We're going to give the code a makeover. We're going to be proud of it, and the "last guy" won't recognize it. But he'll be able to add features to it, fix bugs and know he's not damaging the system.

This session is intended for C++ developers and development team leaders who would like to learn about tools and practices to make code production ready. We'll talk about why the code smells, and how to help it expose its inner beauty.

Functional programming you already know (Kevlin Henney)

90 mins

From JVM to .NET languages, from minor coding idioms to system-level architectures, functional programming is enjoying a long overdue surge in interest. Functional programming is certainly not a new idea and, although not apparently as mainstream as object-oriented and procedural programming, many of its concepts are also more familiar than many programmers believe.

This talk examines functional and declarative programming styles from the point of view of coding patterns, little languages and programming techniques already familiar to many programmers.

Getting into git (Pete Goodliffe)

90 mins

Git is becoming increasingly popular as a source control system, both in the wild yonders of the open source world, and in corporate environments, where it's gaining traction just as subversion did just a few years ago. This talk will explain what all the fuss is about. This talk provides:

- An overview of distributed version control, to understand how it differs from "traditional" models, what problems it solves, and what problems it creates.
- An overview of git, explaining it's particular model of usage and idiomatic workflows.
- An introduction to basic git usageworkflows.
- How to work with other git users.
- How to branch and merge effectively.
- How to use git as a "better subversion" and how to use it to integrate with an existing subversion repository.
- Tools that'll make your life with git easier.
- Routes into more advanced information.

You'll leave with an understanding of git, and will be able to start using it immediately.

Git in the enterprise (Charles Bailey)

90 mins

Git is a popular distributed version control tool. It was written for and is used to maintain the Linux kernel and has been adopted by many open source projects.

Suitable for commercial environments?
Commercial organizations have requirements for their version control systems that differ from those of open source projects. Git can be used in ways that meet many commercial requirements and many of the strengths that appear to apply specifically to open source environments can also have value in a commercial environment.

Installation choice, authentication and access control.
While Git is a distributed tool, most workflows require one or more shared access repositories for interaction between developers. There are a number of options to be evaluated such as hosted solution (e.g. GitHub), a "vanilla" install on a private server or a potentially more feature-rich third party solution such as gitolite, gitosis or gitorious. Each choice of "server" has implications for the implementation of authentication and access controls.

Moving from a centralized version control tool.
There are obvious differences between a centralized version control tool and a distributed one but there also some less obvious differences that will affect which workflows and practices are most productive.

Team and repository size.
Git repositories can grow very large but this is not usually the first cause of scalability issues. The most acute scalability issues come from the frequency with which pushes are attempted to any single branch. This is usually affected by a combination of team size and the level of activity on a project. Often symptomatic of organizational complexity, these scalability issues can usually be mitigated by choosing a

sensible division of projects and responsibilities and by taking advantage of Git's powerful and flexible branching options. Sometimes tackling the practical Git issues can provide surprising insight into the organizational issues.

Supporting developers.
Git has features that can help developers and can be used to support project standards and practices. Git can be integrated with automated test tools to provide per-commit or per-push verification on submitted code. Git hooks can be used to perform checks on both submitted code and commit metadata.

Go, D, C++ and the Multicore revolution (Russel Winder)

90 mins

C++11 is now with us, but is it too little too late? Can the 10+ year old D finally take the limelight as the successor of C++. Can the 18 month old language Go sweep aside C, C++ and D. Certainly Go has the might of Google, and other major players, behind it as the native code language for working with the Web and the Cloud. Will C++ evolve fast enough not to be discarded into the dustbin of programming language history?

This session will look at the concurrency and parallelism features of these three languages to see if C++ can survive by pulling on its HPC roots, or whether the C++11 standard marks the beginning of the end of that language by having focused on low level infrastructure issues at the expense of the high level features required for application in the Multicore Era — models that Go and D already have and are exploiting well.

Groundhog day of a team leader (Zsolt Fabok)

45 mins

Lean thinking and Kanban usually approach an organisation from the executive or management level, but the people who are working in development and testing see things a bit differently. It takes a long journey until a developer feels the weight of an service level agreement, or understands the different prioritisation methods. The team leader is the person who guides the developer and tester on their journey. In my presentation, I intend to show the team leader's perspective of Lean thinking and its implementation.

Guiding a team in an organisation which follows the Lean principles isn't that easy, especially as a team leader. Unfortunately, it isn't enough to say that "from now on, we are going to have a limit on the tasks in development phase...", because this is not enough. People need to get out of their comfort zones, they must understand the new principles and they must be more disciplined than in an agile environment. Additionally, the team leader should report to the management, and ensure that the Lean and/or Kanban transition will not have a bad effect on the organisation, for example missing delivery dates, or quality issues.

During the last three years, I've observed up several different interesting behaviour patterns and scenarios in what people did on their Lean journey. In my presentation, I'm going to show the most interesting ones together with the actions I took in order to save the situations (e.g. gemba walk, motivation techniques), and help my teams be better at their profession and be better Lean thinkers.

Uniquely, I'm also going to prepare several cases where I did wrong, like in the movie 'Groundhog Day'. One have try over and over again until he or she finds the right path to follow.

How to write a testable state machine (Matthew Jones)

45 mins

Anyone can code up a state machine, but can you make such a machine fully testable? Can you prove that it is? Can you do this repeatably?

This talk presents a way of designing and coding state machines that ensures separation between the state transition logic, and the application controlled by state machine. This separation makes testing very easy because the code under test no longer has a dependency on the application. Such dependencies can seriously impede automated unit testing.

Part GOOS, part design patterns, part TDD; the approach adds tests for expected actions, given incoming events, and these drive out new states and transitions. Given state transition logic that is fully testable, layers of tests can be built: individual state transitions, or sequences of transitions which in essence test the use cases of the state machine.

The talk will finish with some real world examples to prove the technique can be applied to more than trivial cases.

If it was hard to write it should be hard to read - some musings on code readability and re-use (Roger Orr)

90 mins

The notorious "Real Programmer" first depicted in the online article "Real Programmers Don't Use Pascal" coined a number of phrases to describe the activity of many programmers. There are many more - for example see http://www.suslik.org/Humour/Computer/Langs/real_prog2.html

One such phrase is "Real programmers don't comment their code. If it was hard to write, it should be hard to read." Using this as a starting point I am going to talk about code readability. Is this statement true? Should it be true?

What makes code hard to read, and does this matter? I will look at the interaction between code readability and code reuse and how it also impacts the 'not invented here' syndrome.

I will illustrate my talk with a variety of examples of code; some of it is going to be hard to read and some of it will (I hope) be much easier.

The examples, reflecting my own experience, are likely to be in C++, Java and C# with perhaps some other languages thrown in for good measure. However, in most cases the issue is not so much with obscrure details of the specific computer language itself but with the overall way the code is written.

Is C going the way of the Dodo? (Dirk Haun)

45 mins

Triggered by my employer's hand-wringing search for C programmers, I was wondering: Where do new C programmers actually come from?

C itself is still very much in use in many areas, yet CS students only seem to be learning higher level languages at university these days. While a competent programmer will be able to pick up pretty much any language in a short time, C does have some peculiarities - like pointers - that no other programming language has and that are a common cause of problems.

So what can we do to ensure a steady supply of experienced C programmers well into the future? Not claiming to have all the answers, I'd also like to invite the audience to help in brainstorming.

Is Scrum incompatible with your brain? (Henrik Berglund)

45 mins

When you start using Scrum, interesting things start happening in your brain. Scrum exposes a lot of problems, and you brain tries to make you feel better by automatically sweeping them under the carpet.

Unfortunately this also means that improvements achieved will be limited and actually it will make you feel worse rather than better. This talk will show you how to handle this and other basic human issues that occur when you start working on changing and improving. After the talk you will have new skills that you can apply to make progress on any problem you care about.

Audience: Mainly developers, but also humans in general

Lambdas in Java 8 (Angelika Langer)

90 mins

Java 8 will introduce elements of functional programming to the Java programming language - the so-called "lambda expressions" (formerly known as "closures"). The language extension will include functional types, lambda expressions, extension methods, method/constructor references and local variable capture. The tutorial will explain the new language features along with their purpose.

Numerous JDK abstractions will be reengineered to use the new language feature. The most radical overhaul will affect the JDK collections. Bulk operations (also known as "filter-map-reduce for Java") will be added to the collections. Their implementation will optionally offer parallel execution by multiple threads using a fork-join thread pool.

Lean Quality Assurance: Much more cost-effective Quality Assurance methods than Testing (Tom Gilb)

90 mins

QA usually means testing and only testing. Yet testing is a very expensive and ineffective method of assuring the needed quality levels. Testing is better used as a last resort, desperate attempt to assure quality. Quality Assurance is far more than 'test', and it can be far more cost-effective. 'Quality' is far more than 'bugs'. If you want real competitive quality, you probably have a lot to learn.

You will be introduced to a set of methods that are 10x more cost effective than test, and you will learn to perform an Agile Inspection.

Topics: All are Lean Methods

- How to perform an Agile Inspection
- Measure Quality Levels in Specifications with Inspection
- Defect Prevention Process
- Designing Quality in
- Designing to meet quality within cost
- Quality is far more than Bugs
- Stakeholders Decide Qualities
- Quality Quantification
- Numeric Quality Gateways
- Value Management Process with frequent feedback and change

Lightning Keynotes (organized by Ewan Milne)

60 mins

Lightning Keynotes - kind of like the lightning talks, except these speakers just won't shut up. The lightning keynotes is a 60 minute sequence of fifteen minute talks given by four speakers on a variety of topics. There are no restrictions on the subject matter of the talks, the only limit is the maximum time length.

We will select the keynote speakers and topics at the conference itself.

Lightning Talks (organized by Ewan Milne)

60 mins

Each session of lightning talks is a sequence of five minute talks given by different speakers on a variety of topics. There are no restrictions on the subject matter of the talks, the only limit is the maximum time length. The talks are brief, interesting, fun, and there's always another one coming along in a few minutes.

We will be putting the actual programme of talks together at the conference itself. If you are attending but not already speaking, but you still have something to say, this is the ideal opportunity to take part. Maybe you have some experience to share, an idea to pitch, or even want to ask the audience a question. Whatever it is, if it fits into five minutes, it could be one of our talks. Details on how to sign up will be announced at the event, or simply collar Ewan whenever you see him in the hallway.

Making Jenkins better (Jez Higgins)

90 mins

Jenkins is a widely used and extremely capable continuous integration server. While it's been available since 2007, under it's original name of Hudson, in popularity seems to have really taken off in the past year or so. One of the primary reasons for its success is it's extremely flexible configuration. Jenkins has a quite a small core, with most of its functionality provided through plugins. Jenkins plugins provide access to different source code control systems, a wide variety of build tools, test result tracking and charting, static analysis tools, and so on. Nearly every aspect of Jenkins can be customised via a plugin. At time of writing there are over 400 different Jenkins plugins available.

Four hundred is too few.

Over the past two years, we've from dabbling with CI to Jenkins forming part of our core toolset. Jenkins builds on checkin, yes, but also deploys builds into development environments. It runs performances tests and records the history. It matches tells us which build contains which bug fixes. It also does our release builds - tagging the repository, building from the tag, writes release notes telling us which work packs have been updated, pushes the build up onto the live server, and emails Ops to say everything is ready to go. The standard plugins provide the foundation, but the our own plugins have put Jenkins at the heart of our development process.

If you want to get the most from Jenkins, you really should write your own plugins. This session will explain why you should, what you can change or add to Jenkins, and how to do it.

Open Source is good for you (Dirk Haun)

45 mins

While open source applications are widely accepted and used, both at home as well as at work, using open source components as part of their software is something that companies still seem to shy away from. Why is that?

In this talk, we're going to look at some of the reasons for that reluctancy, e.g. legal reasons or fear of the GPL, and check if they're valid and how much of a problem they really are in practice.

As a counterpoint, we’re going to look at the benefits of having open source components. Also on the agenda: How to find an open source component that fits your requirements and some thoughts on giving back to the open source community without having to give away all your company’s secrets.

Parallel Architectures (Detlef Vollman)

90 mins

Concurrency is one of the big challenges of computer programming for a number of years and will continue to be a major challenge in the near future. But "concurrency" is not a single feature or mechanism, but it's a whole domain of mechanisms, problems, aspects, pitfalls, opinions and believes in an environment that's continously changing.

This talk will provide an overview of concurrency, both in hardware and software. It will present existing hardware and current trends in computer architecture that cause concurrency and common mechanisms in software to leverage and to manage and control this concurrency.

Examples in software will be shown in C++, but should be generally understandable without detailed C++ knowledge. No specific library or API will be presented, but only general mechanisms.

This talk is not for programmers who want to hear about the latest thread library in C++. This talk is for programmers and designers of projects that have to deal with concurrency. No knowledge of C++ is required.

Paralysis by Parallelism (Russel Winder, Schalk Cronje)

90 mins

Mulit-CPU and multi-core architectures have been with us for a long while now, but are we using them properly: are your programs running effectively on these platforms, or are your processes "paralysed by parallelism". Is your mindset locked by mutexes and do you believe a threadpool is your only hope of multi-tasking? Do you think actors are only found in movies, and that pipelines only carry oil?

In this fun-poking session we explore numerous questions that come up when people try to move on to new concurrency and parallelism paradigms. Using examples in various languages we show the audience pragmatic ways of using actors, pipelines, dataflows, message queues, and possibly other tools and techniques

Plenty People Programming with the C++ 11 Standard Library (Nico Josuttis)

90 mins

What happens if all people in the room solve together some programming problems covering the C++11 Standard Library? We will find out ...

Real Teams (Henrik Berglund)

45 mins

In sports we have teams that win together, loose together and celebrate together. Team members depend on each other. Using their motivation they can make us gasp as they beat opponents with seemingly superior skills.

This is somewhat different from the type of "teams" a lot of companies have been using to develop software. Lately though, a lot of companies have been asking their people to work in some sort of self organizing agile team. This is a very similar idea to the sports team.

This session starts by examining why this major change is happening and then quickly moves on to show how you can make real teams a reality where you work. We will be covering a set of principles that, when put in place, pretty much guarantee that a real team will form. It does not matter if you are a developer, scrum master or a line manager. If you want to, you can make this happen.

You will leave with a set of ideas and tools that you can start applying right away at your workplace. The goal is that you will have more fun as you win, loose and create amazing products together as a team.

Audience: Developers, Scrum masters, managers, ...

Refactoring to functional (Andrew Parker, Julian Kelsey, Steve Freeman)

90 mins

"Those who know no foreign language know nothing of their mother tongue" Goethe

Knowing functional techniques leads to better object oriented code, just as knowing about objects leads to better procedural code. The trick is getting from here to there.

We will take three key features of imperative languages - sequence, selection, and repetition - and look at systematic ways to refactor them to a functional style. These techniques reveal assumptions about imperative programming and help developers be more expressive in their workaday programming languages.

We will discuss the benefits that functional ideas bring (such as expressiveness, modularity, and safety) and show how we try to shoehorn them into (sometimes hostile) imperative languages, in particular Java.

The workshop will include demonstrations and exercises for attendees to try out themselves.

Schedule (90m total)
5m Introductions and setup
10m What and Why of functional
15m Patterns of Repetition (with demo)
15m Exercise
15m Patterns of Selection and Sequence (with demo)
20m Exercise
10m Wrap up/Discussion

SOLID deconstruction (Kevlin Henney)

90 mins

The SOLID principles are often presented as being core to good object-oriented practice. Each of S, O, L, I and D do not, however, necessarily mean what programmers expect they mean or are taught they mean. By understanding this range of beliefs we can learn more about our OO practice than just S, O, L, I and D.

This session starts by going over the SOLID principles, looking at code examples and also different interpretations of the principles themselves. Contradictions and questions are revealed. It is through paradoxes and surprises that we often gain insights and improve our skills. We will leave SOLID slightly more fluid, but having learnt from them more than we expected.

TDD for Embedded C (James Grenning)

90 mins

Rumor has it that TDD cannot be used for developing C, let alone embedded C. The rumor is wrong! TDD can be used effectively for all forms of C. In the session you'll see how to break dependencies right down to the silicon. You probably have some legacy C that is resisting to submit to your test harness. We'll look at some of the techniques for getting your legacy C into the test harness.

Many of the challenges in embedded development stem from the target hardware bottleneck. The bottleneck slows progress of the embedded software development due to many contributing factors including non-existent or buggy hardware, and the inefficiencies of cross-platform development. The session shows how to effectively use TDD and object oriented design techniques to overcome the target hardware bottleneck.

TDD in Assembler (Olve Maudal)

90 mins

You can't do test first in X? Yes you can! While test-driven development is now an established practice in the industry and a very powerful tool used by many software professionals, it is still common to meet developers saying: "TDD? Heard of it, but certainly not useful/applicable/possible for my kind of work". While it is true that TDD is not always the best approach, it is still a technique that is surprisingly useful for nearly every programming task. In this session I will do a live demonstration on how you can start from scratch and use TDD techniques to develop a simple but complete program, including buffered IO routines, in assembler.

Testing a la carte (Klaus Marquardt)

90 mins

Testing has become a cardinal virtue of software developers. And you certainly can become a decent tester from repeated practice, and learning from your text book. However, you might agree that your home cooking (repeated practice and a book on your side) does not yield results similar to a master cook. No offense meant, it certainly applies to my home cooking.

So masters do something different. They have spotted something that makes things not just work, or work better, but delicious. Plus, most are willing to share their insights and guide your next steps.

This session invites fellow ACCU participants to share and present their goodies and insights to the ACCU public. Each good idea will then be discussed in a workshop, and checked for applicability and for beneficial combinations in different settings and domains. Finally, the results will be made available as a flipchart exhibition.

Schedule:

- presentation of insights and goodies: 40 min
- workshop on applicability and combinations: 40 min
- wrapup and exhibition: 10 min

If you are ready to share, please drop me a line

Testing embedded systems (Klaus Marquardt)

90 mins

This session invites fellow ACCU participants to share their experiences in testing embedded systems. The idea is similar to the "Pattern Buffet" that we had in 2009, or the complexity topic in 2011.

However, the session is split and turns into a workshop afterwards, so that participants gain more insights on their particular issues.

Schedule:

- invited talks: selected participants, 10min each (35min)
- feedback: exploration of practices, and a voting on what would work and what would not (15min)
- consequences for project practices: a workshop on what testing and design techniques need to be in place to support the testing approaches (40min)

The best and worst new features of the C++11 Standard Library (Nico Josuttis)

90 mins

For the second edition of "The C++ Standard Library" covering C++11 (C++0x), I tried hard to understand the new features of both the C++ language and its library. Sometimes I execrated, sometimes I praised what I found (out). So, let's present my top goodies and disappointments.

The impact of virtualization on software architectures and lifecycles - A practical approach (Arno-Can Uestuensoez)

90 mins

During the last years the virtualization of complete physical machines to "programs-only" evolved as a common technology. The application of this techniques until now are considered as systems administration approaches only.

In first evolutionary steps the virtual machines are defined as containers for application specific configurations as so called appliances. These are defined as simple containers for a set of programs to be used in combination. This may comprise the main application and some additional base software. Thus the targeted advance mainly comprises version consistency and flexibility in handling and distribution of "application specific servers and workstations".

The second step evolving from this is the actual usage of virtual machines within private and public networks as dynamic distributable servers and workstations packaged as programs only. These require one interface only as runtime environment - the hypervisor - thus offer basic support for scalability and flexibility of actual execution location.

The next step of evolution now is the application of more granular packaged virtual machines as components to be assembled to a combined feature set. This approach offers advance for multiple aspects such as application modularity and dynamic runtime reconfiguration. One essential feature is here the introduction of nested virtual machines as virtual stacks. This provides a logical vertical tree structure for software components, which is mainly based on network interfaces only and therefore could be dynamically assembled and reconfigured, even redistributed component-wise.

The current limitations due performance impact when using emulators only is going to be eliminated with the now uprising manycore CPUs, which enable the mapping of the logical

structure to a flat array of distributed cores. Thus for execution with almost no performance impact.

The session demonstrates the application of virtual machines as virtual components including their application in cloud based software architectures.

The importance of readability in code (Michel Grootjans)

90 mins

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand." - Martin Fowler et al, Refactoring: Improving the Design of Existing Code, 1999

Objectives of the session: I will show examples of how code can quickly become unreadable. I will show examples of how to achieve readabiltiy by refeactoring existing code.

Content: Code should always be expressed as clearly as possible. Some languages provide flexible ways to enhance code readability. Ruby leads the way. .net has a extensibility points to make our life as developers a lot easier.

The Victorian Exploring Expedition of 1860: reflections on project management and leadership from a real-life deathmarch project (Jim Hague)

90 mins

You know where you're starting from. You know roughly where you want to go to. You have a fair idea of what you're going to have to do to get half way there. But you can only guess what problems you will encounter from that point. Still, you have a wonderful new technology that will make your journey much easier, you have generous funding and the backing of the board, even if some of the detailed objectives keep changing. What could possibly go wrong?

This session tells the tragic story of the Victorian Exploring Expedition of 1860. A generously-funded attempt to chart an overland route to the Gulf of Carpentaria on Australia's north coast, it ended achieving its primary goal but at a terrible cost, both financially - it ended 5x over budget - and otherwise, as 8 men perished on what became, literally, a death march project. We look at the management and leadership failures that doomed a project that came agonisingly close to success but ended in disaster, and reflect on parallels with the human issues in contemporary software development.

UMLGraph and the Declarative Drawing of Diagrams (Diomidis Spinellis)

90 mins

While a picture might be worth a thousand words, the hundreds of mouse clicks required to draw and maintain it often seem like a waste of productive time. Worse, it's awkward to track mouse-drawn pictures through revision control, keep them in sync with other parts of our project, or automatically refactor their contents. Fortunately, computer power and automatic graph-drawing algorithms have advanced so as to allow the automatic placement of graph nodes on a canvas and the routing of the respective edges. So, we can design our models and other diagrams using a declarative textual representation (drawing with words), and then view, publish, and share them in graphical form. Creating diagrams in a declarative, textual notation has several advantages. We can focus on our drawing's essential elements, leverage our code-writing skills, automate the creation process, and use our editor and other text tools to examine the picture's properties and perform large-scale changes.

To draw pictures with words we can use many tools and graphics formats. With the Graphviz system we can draw directed and undirected relations between elements using a simple declarative language. Building on top of it UMLGraph allows the declarative drawing of UML models and the reverse-engineering of such models from Java code. Pic gives us a procedural drawing language that lets us define our own domain-specific drawing language. With Gnuplot we can plot data and functions in a wide variety of 2D and 3D styles. Finally, we can plot geographical data through the Generic Mapping Tools (GMT) or by generating KML files, and we can obtain additional leverage by having one graphics tool or script generate output for another.

Unit testing and mocking in C++ (Ed Sykes, Rajpal Singh)

90 mins

We will be talking about our experiences with two C++ mocking libraries - MockItNow and Hippomocks. We'll also talk about our experience of using TDD and unit testing in c++. We'll give an short explanation of mocking and unit testing before sharing our experiences. We'll also have a look at some other mocking frameworks although we can't guarantee that we'll have used them to help write production code :)

Hopefully this talk will help push these proven modern practices into the C++ world which has been one of the slowest languages to adopt them.

Variance in Generic Types in Java and C# (Robert Stanforth)

90 mins

With the launch of Java 5 in 2004, probably the most attention-grabbing change to the language was the introduction of generics. This major extension to the type system brought with it some startling and often opaque new syntax, with even the innocuous binary search method in the Java collections framework having its signature upgraded to "public static <T> int binarySearch(List<? extends Comparable<? super T>> list, T key)". These so-called 'wildcard' types are provided to solve the conundrum of converting between containers of different but related types, for example List<Cat> and List<Animal>.

In this session we set out the problems associated with such seemingly plausible conversions, motivating the question of variance of generic types in object-oriented languages. We introduce the concepts of covariance and contravariance, and use them to deduce the precise rules that would apply to a type system that properly incorporates generics with variance.

The discourse is illustrated with a detailed examination of Java's solution of 'wildcard' types, which we compare to the rather different approach to the same problem taken more recently by C# 4.0. Although Java's formulation encourages us to think in terms of wildcards, there will emerge some complementary perspectives on what variant generic types denote, helping us to make sense of the austere syntax.

Version control done right (Pete Goodliffe)

90 mins

In these enlightened times, it's no surprise that developers should be using version control. We've been doing it for decades. (Well, some of us have.) However, not all developers use version control as well as they could. In this talk, we'll look at how to use version control **well**, not just as place to dump code, but a sharp tool that is central to the development process.

We'll investigate some history of source control (inasmuch as it will help us understand how it works, and how to select the best kind of tool), look at the kinds of version control tools available, and their integration into our devevelopment tools and processes. We'll then investigate workflows that will turbocharge your personal development regimine and streamline the way you work with other developers.

What is code simplicity? (Arjan Van Leeuwen)

90 mins

Lots of books have been written about refactoring and code design, but what is it we're trying to reach? What is our goal in coding? In this talk and workshop we explore what simple code means at a low level and at a high level, why simple is always better, and how we can create simpler code.

In university we were always taught to write elegant code. I explicitly use the term simple in this talk, because I believe it's less loaded, and it's easier to get to a definition of what simplicity really is, while elegance seems more subjective. Code simplicity is very much related to code readability - by keeping code simple, we try to keep it understandable and maintainable for future developers (and probably ourselves).

One of the things we'll do in this talk is to see if we can get to a definition of simple code that we can all agree with. We'll talk about existing definitions of the opposite term, code complexity, and we'll try to find examples of truly simple code to see if we can determine what it really is that makes that code simple.

Using examples from production code we'll then take a look at different situations where code can be made simpler. Using our definition, we'll try to simplify the code in small groups, and en route see if we can further specify our definition of simple code (perhaps we can even simplify it!).

When only C will do (Andrew Stitcher)

45 mins

We've all grown accustomed to the conveniences of C++ and the so called "scripting" languages like Python or Ruby. The relative paucity of facilities in C means that the common wisdom is to only program in C when you need to improve the performance of your code. But there are other good reason why you might want or even need to code in C. In this session we'll discuss some the most important reasons as well as some of the pitfalls you might encounter on the way.