Software life cycle, high assurance & types
Stages in software development

- Requirements
- Specification
- Design
- Implementation
- Validation
- Maintenance
Waterfall model versus agile methods
Waterfall model

- Requirements
- Specification
- Design
- Implementation
- Validation
- Maintenance
Agile methods

Requirement → Specification → Design → Implementation → Validation → Maintenance → Requirement
Different projects require different methods

- Implementing an AES (Advanced Encryption Standard) component
  - Requirements and specification is not going to change significantly
  - Predictability and correctness are paramount

- Implementing a social media website
  - Requirements are initially very vague
  - Web users are well accustomed to half-baked features and a little downtime
Again, we need to be flexible

• We need to be able to **trade quality for reduced effort**

• We need to be able to **trade predictability for agility**
The scope of this course
Logical program properties

• Our tool of choice for specifications

• They are flexible
  
  1. They can directly be used for testing
  
  2. They can directly be used for formal verification

• They are fundamentally connected to types
Type-driven development

• We will use types for all four stages of software development

1. Specification — types can encode arbitrary properties

2. Design — types structure code

3. Implementation — types guide and sometimes imply implementations

4. Validation — types can be automatically checked
Types provide flexibility

- **Singleton types** are perfectly precise
  \[ n : SInt(n) \]

- **Bit-size types** track an important implementation constraint
  \[ n : BInt(w) \]

- Types as we know them
  \[ n : Int \]

- **Dynamic types**
  \[ n : Dynamic \]
By making types more precise...

• We refine the specification

• The type checker requires us to justify our implementation in more detail

We gain quality, but also have to spend more effort
By making types less precise...

- We simplify experimentation
- We will have to perform more testing, or accept defects

We avoid fixing too many details of the specification
Lambda calculus in a nutshell
Haskell

• A practical, strongly-typed functional programming language
  ‣ Widely used in research, industry & education
  ‣ Mature, highly optimising compiler with interactive environment
  ‣ Over thousands of open-source libraries and tools

• Named after the logician Haskell B. Curry

http://haskell.org/
Why Haskell?

- Functional languages are based on the lambda calculus
  - Semantics of programs is fairly precisely defined
  - This simplifies formal reasoning about these programs

- Functional languages can dramatically increase productivity
  - Factor of four has been cited for Erlang versus C++

- Haskell has a very sophisticated type system

- Haskell has controlled effects