Advanced Programming

ACSE-5: Lecture 10

Adriana Paluszny

Overview: Where do we go from here?

- C++20
- Roles in programming teams
- Programming methodologies
- Agile
- Introduction to UML
- Design patterns

C++20

```
template<typename T>
concept EqualityComparable = requires(T a, T b) {
    { a == b } -> std::same_as<bool>;
    { a != b } -> std::same_as<bool>;
};
```

- Before: C++ 14, C++ 17

History [edit]

Changes applied to the C++20 working draft in July 2017 (Toronto) include:[7]

- concepts (what made it into the standard is a cut-down version; also described as "Concepts Lite"[8])
- designated initializers
- `[=, this]` as a lambda capture
- template parameter lists on lambdas
- `std::make_shared` and `std::allocate_shared` for arrays

Changes applied to the C++20 working draft in the fall meeting in November 2017 (Albuquerque) include:[9][10]

- three-way comparison using the "spaceship operator", `operator <=>`
- initialization of an additional variable within a range-based `for` statement
- lambdas in unevaluated contexts
- default constructible and assignable stateless lambdas
- allow pack expansions in lambda *init-capture*
- string literals as template parameters
- atomic smart pointers (such as `std::atomic<shared_ptr<T>>` and `std::atomic<weak_ptr<T>>`)
- `std::to_address` to convert a pointer to a raw pointer

Changes applied to the C++20 working draft in March 2018 (Jacksonville) include:[11]

- removing the need for `typename` in certain circumstances

In computer science, a **three-way comparison** takes two values A and B belonging to a type with a total order and determines whether A < B, A = B, or A > B in a single operation, in accordance with the mathematical law of trichotomy.

Library [edit]

- ranges (The One Ranges Proposal)[96]
- `std::make_shared` and `std::allocate_shared` for arrays[97]
- atomic smart pointers (such as `std::atomic<shared_ptr<T>>` and `std::atomic<weak_ptr<T>>`)[98]
- `std::to_address` to convert a pointer to a raw pointer[99]
- calendar and time-zone additions to `<chrono>`[100]
- `std::span` , providing a view to a contiguous array (analogous to `std::string_view` but `span` can mutate the referenced sequence)[101]
- `std::erase` and `std::erase_if` , simplifying element erasure for most standard containers[102]
- `<version>` header[103]
- `std::bit_cast<>` for type casting of object representations, with less verbosity than `memcpy()` and more ability to exploit compiler internals[104]
- feature test macros[105]

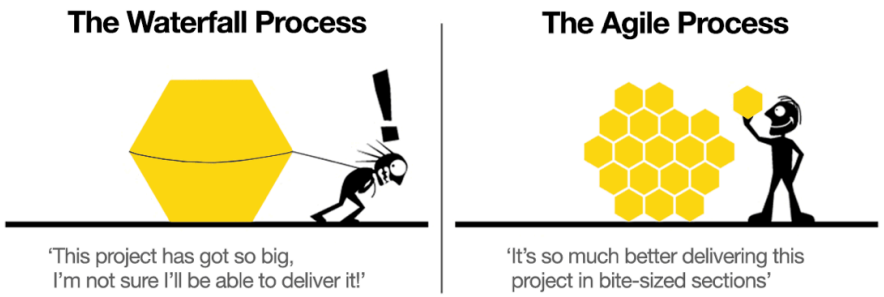
Checkout: https://cppcon.org

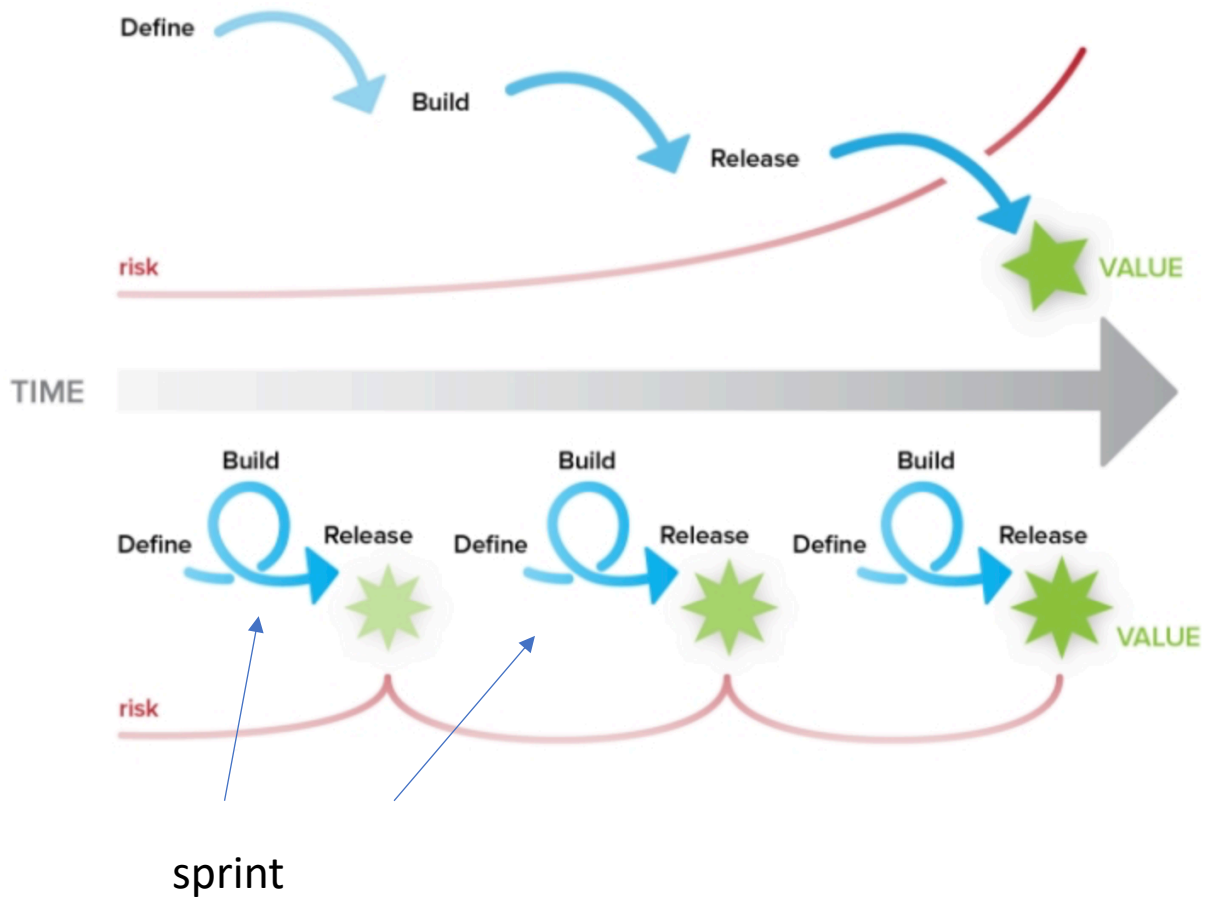
Roles in programming teams

- Architect
- Tech Lead / CTO
- Software Developer Junior/Senior
- Scientific Developer
- Back end/ Front end Developer
- Full Stack Developer
- Product Manager/Lead
- Tester / Quality Control Manager

Programming/Development methodologies

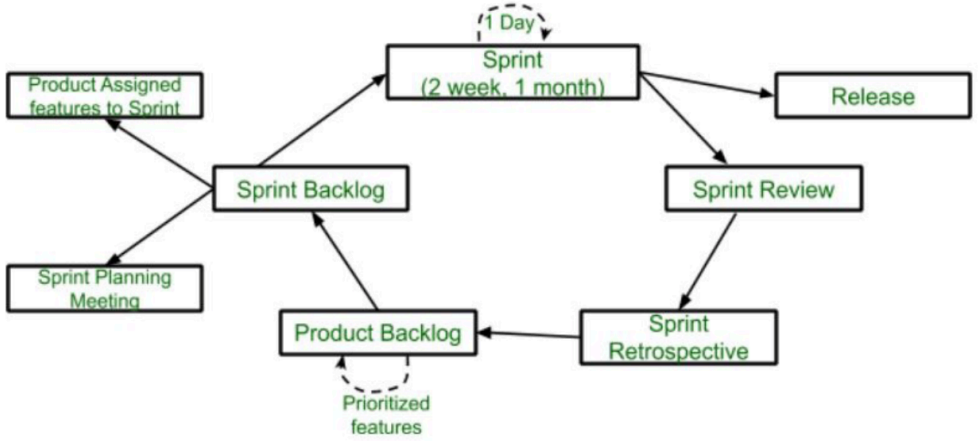
- Advanced programming for software engineering
- Waterfall, Prototype, Agile





Agile

- Based on constant improvement (iterations/sprints)
- Iterative, dynamic, long vs. short nested cycles (30day/24hr cycles)



Scrum	Extreme Programming (XP)
In Scrum framework, team work in iterations called Sprint which are 1-2 month long.	In Extreme Programming(XP), teamwork for 1-2 weeks only.
Scrum model do not allow changes in their timeline or their guidelines.	Extreme Programming allow changes in their set timelines.
Scrum emphasizes self-organization.	Extreme Programming emphasizes strong engineering practices
In Scrum framework, team determines the sequence in which the product will be developed.	In Extreme Programming, team have to follow a strict priority order or pre-determined priority order.
Scrum framework is not fully described. If you want to adopt it then you need to fill the framework with your own frameworks method like XP, DSDM or Kanban.	Extreme Programming(XP) can be directly applied to a team.
Scrum master: ensures the process but does not participate	

(Micro) Introduction to UML

- Standard graphical notation for specifying, designing and documenting software
- Visualise design
- Increase understanding and communication of design within the team, stakeholders and customers
- Use case diagrammes (Behavioural)
- Class diagrammes (Structural)
- Sequence diagrammes (Behavioural)
- State diagrammes (Behavioural)
- Deployment diagramme (Structural)

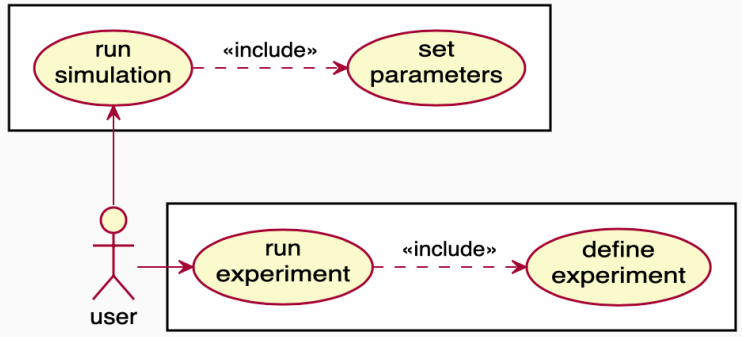
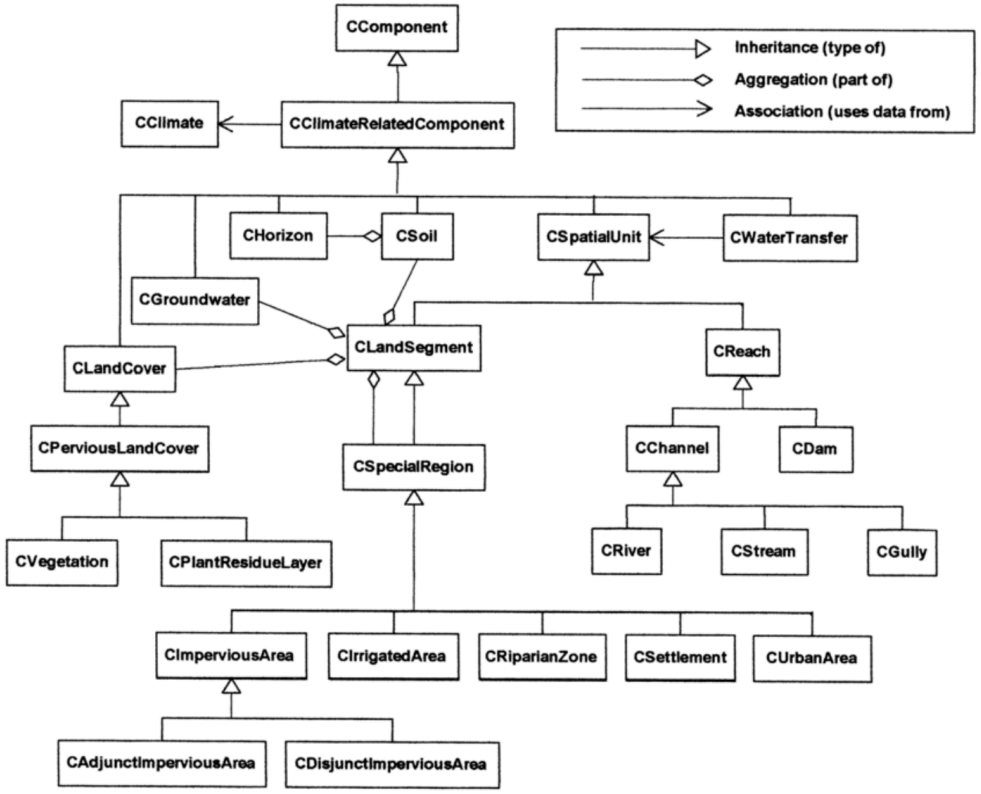


Figure 1: Use-Case Diagram: Interaction with a Simulation Model

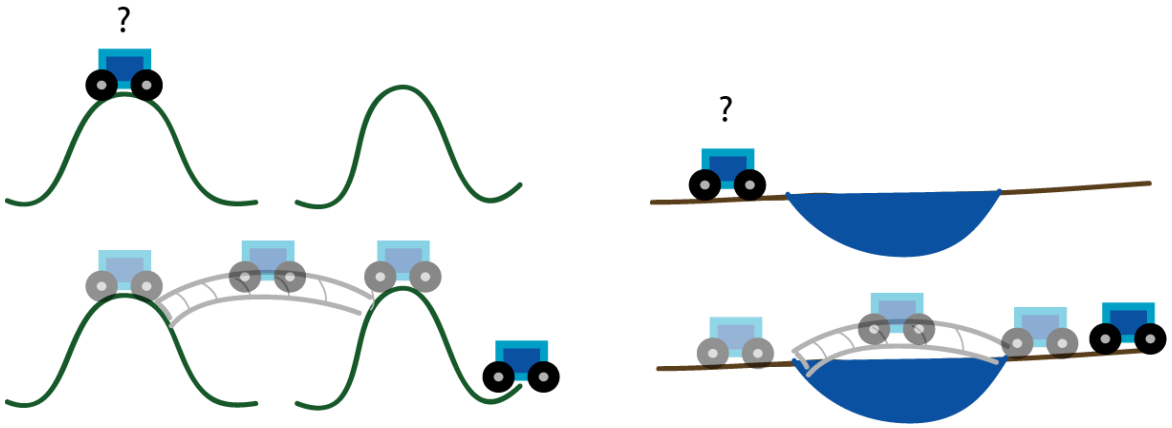


UML class diagram showing some ACRU2000 Component classes and their relationships.

Further reading: https://developer.ibm.com/technologies/web-development/articles/an-introduction-to-uml/

What is a design pattern?

- describes **problem** that occurs over and over
- describes the core of the **solution** to the problem
- **consequences** of applying the pattern
- no two solutions will ever be the same



Classification

Purpose (what the pattern does)

Creational – creating objects
Structural – how objects are composed
Behavioral – interaction and responsibility distribution

Scope (application: classes or objects?)

classes relationships: static / compile time
object relationships*: dynamic / execution time

* most patterns are in the object scope

Examples of Design Patterns

Singleton: Creational Object
Bridge: Structural Object
Factory Method: Creational Class
Iterator: Behavioral Object
Visitor: Behavioral Object
Template: Behavioral Class

Further reading: https://refactoring.guru/design-patterns/cpp