C++ ecosystem: For better, for worse

Anastasia Kazakova
JetBrains
@anastasiak2512
Agenda

1. The current state of C++ development
2. C++ in top areas. Needs and requests
3. What else is important? Unit testing & code analysis
4. Language evolution and tooling
The State of Developer Ecosystem

- Yearly: 2017, 2018, 2019
- ~15K respondents total
- 6 languages
- Enough data from all over the world
- Weighting
The State of Developer Ecosystem: C++

- C or C++ used in the last 12 months - **5427**
- C used in the last 12 months - **3410**
- C++ used in the last 12 months - **4148**
- Primary C++ - **1698**
C++ Developer Survey by CPP Foundation

- 2018
- C++ used at work - 2884
- Hobby/personal - 2380
- >50% have >5 years in C++
The State of Developer Ecosystem: C++

Platforms distribution

- Window: 67.7%
- UNIX / Linux: 69.9%
- macOS: 28.4%
- all-3: 9.9%
- any-2: 46.5%
The State of Developer Ecosystem: C++

Areas

- Embedded: 20.0%
- Games: 32.3%
- Libraries/Frameworks: 30.6%
- Mobile: 24.4%
- Desktop: 35.1%
- ML: 21.3%
- Web Back-end: 28.4%
- Web Front-end: 26.9%
The State of Developer Ecosystem: C++

Employment status

- Student: 32.5%
- Self-employed: 3.7%
- Retired: 3.2%
- Partially employed: 3.7%
- Fully employed: 54.5%
- Freelancer: 4.0%
- Other: 0.4%
Throwing a ball

C++ standards
C++ standards

C++ standards usage

- C++98: 13.5%
- C++03: 9.4%
- C++11: 59.8%
- C++14: 48.9%
- C++17: 35.8%
C++ standards

C++ standards 2019-2018

- C++98
  - 20,2 %
- C++03
  - 13,5 %
- C++11
  - 59,4 %
- C++14
  - 39,3 %
- C++17
  - 35,8 %

C++ standards 2018-2019

- 2018
  - C++98: 20,2 %
  - C++03: 12,1 %
  - C++11: 59,4 %
  - C++14: 48,9 %
  - C++17: 35,8 %

- 2019
  - C++98: 13,5 %
  - C++03: 9,4 %
  - C++11: 59,8 %
  - C++14: 39,3 %
  - C++17: 22,0 %
C++ standards

C++ versions

The most popular C++ version is currently C++11, with a share of 34%.
The State of Developer Ecosystem: C++

- Per platforms distribution
- Per compiler distribution
- Per area of development
- Per employment group
C++ standards

C++ standards by platform

Window
- C++98
- C++03
- C++11
- C++14
- C++17

UNIX / Linux
- C++98
- C++03
- C++11
- C++14
- C++17

macOS
- C++98
- C++03
- C++11
- C++14
- C++17
C++ standards

Standards distribution inside each employment group

- Student
- Self-employed
- Partially employed
- Fully employed
- Freelancer

% at C++98 % at C++03 % at C++11 % at C++14 % at C++17
C++ standards

Standards usage for two biggest employment groups

- C++98: 56.3%
  - Students: 27.9%
  - Fully employed: 71.1%
- C++03: 71.1%
  - Students: 13.2%
  - Fully employed: 57.2%
- C++11: 57.2%
  - Students: 30.1%
  - Fully employed: 34.4%
- C++14: 53.3%
  - Students: 34.4%
  - Fully employed: 30.1%
- C++17: 46.7%
  - Students: 40.6%
  - Fully employed: 13.2%
Throwing a ball

Upgrading
C++ standards: upgrade

Plans to upgrade

- 53.4% to C++17
- 53.4% to C++14
- 1.7% to C++11
- 0.1% to C++03
- 38.8% no upgrade
C++ standards: upgrade

Willing to upgrade to newer standard per current standard in use

- All respondents
- C++98
- C++03
- C++11
- C++14
- C++17

% to C++17 | % to C++14 | % to C++11 | % to C++03 | % no upgrade
Throwing a ball

C++ per areas
C++ per areas

- Finances / Banking / Trading
- Embedded
- Games
C++ in Banking and Trading

- Language choices:
  - **Java** for the big enterprise systems, back end trading platforms etc.
  - **C++** for the low latency / high performance stuff
  - **C#** for front-end / desktop apps
  - **Python** for various scripting
- C++ is a primary choice
- Especially low latency trading and quantitative analytics
- Performance
Performance:
• Low latency, not quick throughput
• And safety
• Requires understanding of the compiler output

Carl Cook “When a Microsecond Is an Eternity: High Performance Trading Systems in C++” (CppCon 2017)
C++ in Banking and Trading

C++ usage:
• Allocations are important
• Exceptions are fine, if they don’t throw and not in the control flow
• Templates over virtual functions and branches
• Usage of low-level CPU instructions

Related ecosystem:
• Huge infrastructure, learning materials, wide expertise
• Lots of SDKs (CUDA, QuantLib)
• High cost of moving to the new technologies
• Affects clients
C++ in Embedded
C++ in Embedded

- Controlled by MCUs vendors
- Testing / Standards compliance / Certification tools
- Language choices:
  - C and C++, often more C than C++
  - Python, Lua, etc. for scripting, configurations, etc.
- Vendor’s compilers / debuggers / etc.
C++ in Embedded

C++ usage:
- Classes are C structs with function pointers
- Macros are everywhere
- Direct memory/registers access
- Data structures in memory are specifically packed
C++ in Games
C++ in Games

- Language choices:
  - Unity/C# takes the biggest part of the market
  - AAA is mostly C++, Unreal Engine, Lumberyard, CryEngine and custom in-house engines
  - Rendering is mostly in C
- Console SDKs in binaries
- Performance (latency)
C++ in Games

C++ usage
- C++03 and C++11
- In-house reflection implementations
- No Boost or STL because of the allocations
- Minimal template usage
- No exceptions because of their cost
C++ in Games

Reflection
  • For serialization
  • For GC
  • For network replication
  • For various characteristics
C++ in Games

Reflection in Unreal Engine:
- Serves for interaction between C++/Blueprint
- Implemented with macros
- RPC methods

```cpp
#include "MyObject.generated.h"

UCLASS(Blueprintable)
class UMyObject : public UObject {
  GENERATED_BODY()

public:
  MyUObject();

  UPROPERTY( BlueprintReadOnly, EditAnywhere )
  float ExampleProperty;

  UFUNCTION( BlueprintCallable )
  void ExampleFunction();

  /** [server] remove all weapons from inventory and destroy them */
  void DestroyInventory();

  /** equip weapon */
  UFUNCTION( reliable, server, WithValidation )
  void ServerEquipWeapon(class AShooterWeapon* NewWeapon);

  void ServerSetTargeting( bool bNewTargeting );

  /** update targeting state */
  UFUNCTION( reliable, server, WithValidation )
  void ServerSetRunning( bool bNewRunning, bool bToggle );
```
Custom STL & Allocations
- No STL, custom structures, plain arrays
- Non-default memory alignment requirements
- Newly constructed or reset container allocates no memory
- Avoiding heap
- Temporal allocators with the life-time of the frame

Sample: InplaceArray<ubi32, 8>

Nicolas Fleury "C++ in Huge AAA Games" (CppCon 2014)
Scott Wardle "Memory and C++ debugging at Electronic Arts" (CppCon 2015)
EASTL – Electronic Arts Standard Template Library

"Among game developers the most fundamental weakness [of the STL] is the std allocator design, and it is this weakness that was the largest contributing factor to the creation of EASTL."
Throwing a ball

Unit testing
Unit testing

Regularly used unit testing framework

- None: 33.4% (2018), 25.2% (2019)
- Boost.Test: 7.8% (2018), 8.5% (2019)
- Google Test: 28.3% (2018), 32.6% (2019)
- CppUnit: 12.1% (2018), 12.8% (2019)
- Catch: 8.5% (2018), 12.0% (2019)
- Custom: 6.8% (2018), 6.0% (2019)
Unit testing

- ~70 in the list: https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks#C++
- Reddit discussions:
  - Most Popular C++ Unit Testing Frameworks
    https://www.reddit.com/r/cpp/comments/4e9afx/most_popular_c_unit_testing_frameworks/
  - Best way to do unit testing in c++?
    https://www.reddit.com/r/cpp/comments/36pru0/best_way_to_do_unit_testing_in_c/
  - Is there a de-facto standard "framework" for unit testing in C++?
    https://www.reddit.com/r/cpp/comments/1zh0p1/is_there_a_defacto_standard_framework_for_unit/
- Recommendations: Google Test (with Google Mock), Catch
## Unit testing

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature rich</td>
<td>Google Test, Boost.Test</td>
</tr>
<tr>
<td>Easy-to-start</td>
<td>Catch</td>
</tr>
<tr>
<td>Integrations</td>
<td>Google Test</td>
</tr>
</tbody>
</table>
Embedded market:
- tests running on hardware
- tests are required for certifications according to the standards
- no home-made products because of the certification
- no integration into IDEs (Eclipse)
- pricy

<table>
<thead>
<tr>
<th>values</th>
<th>External channels N: 227</th>
<th>Internal channels N: 276</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>shares</td>
<td>lower CI</td>
</tr>
<tr>
<td>No, I don’t use any</td>
<td>89%</td>
<td>84%</td>
</tr>
<tr>
<td>Other - Write In</td>
<td>7%</td>
<td>4%</td>
</tr>
<tr>
<td>VectorCAST</td>
<td>1%</td>
<td>0%</td>
</tr>
<tr>
<td>TestPlant</td>
<td>1%</td>
<td>0%</td>
</tr>
<tr>
<td>Parasoft DTP</td>
<td>1%</td>
<td>0%</td>
</tr>
<tr>
<td>RogueWave KlockWork</td>
<td>1%</td>
<td>0%</td>
</tr>
<tr>
<td>QA Systems CANTATA</td>
<td>1%</td>
<td>0%</td>
</tr>
<tr>
<td>Elvior TTCN-3</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>hitex TESSY</td>
<td>0%</td>
<td>0%</td>
</tr>
</tbody>
</table>
Throwing a ball

Code analysis / guidelines enforcement
Code analysis

Code analysis / guideline enforcement tools

None: 30.7%
Clang-analyzer: 21.9%
Clang-Tidy: 23.1%
CppCheck: 16.5%
Coverity: 3.8%
Cpplint: 8.2%
PVS-Studio: 3.2%
IDE: 42.0%
Not throwing a ball

How C++ committee and tooling can help?
Language evolution & tooling

Compatibility and reduced cost of the integration
• C++ mostly never breaks the compatibility
• Redesigning modules
• New exceptions
Support in tooling

- Compilers adopting new features quickly
- IDEs providing support for features
- Features are toolable
Language evolution & tooling

Example:
Templates intellisense

Visual Studio
Language evolution & tooling

Example:
Templates intellisense

ReSharper C++
References

- C++ Foundation Developer Survey
- The State of Developer Ecosystem Survey
- C/C++ Infographics
- Nicolas Fleury "C++ in Huge AAA Games"
  - [CppCon 2014] https://www.youtube.com/watch?v=qYN6eduU06s
- Scott Wardle “Memory and C++ debugging at Electronic Arts”
  - [CppCon 2015] https://www.youtube.com/watch?v=8KlvWJUYbDA
- EASTL - Electronic Arts Standard Template Library
  - [GitHub] https://github.com/electronicarts/EASTL
- Carl Cook “When a Microsecond Is an Eternity: High Performance Trading Systems in C++”
  - [CppCon 2017] https://www.youtube.com/watch?v=NH1Tta7purM
Thank you for your attention.

Questions?