Reviewing Beginners' Code

PATRICE ROY

Cppcon
The C++ Conference

2022 | September 12th-16th

REVIEWING BEGINNERS' CODE

Patrice Roy

Patrice.Roy@USherbrooke.ca; Patrice.Roy@clg.qc.ca

CeFTI, Université de Sherbrooke; Collège Lionel-Groulx

Who am I?

- Father of five, ages 27 to 9
- Feeds and cleans up after a varying number of animals
  - Look for *Paws of Britannia* with your favorite search engine
- Used to write military flight simulator code, among other things
  - CAE Electronics Ltd, IREQ
- Full-time teacher since1998
  - Collège Lionel-Groulx, Université de Sherbrooke
  - Works a lot with game programmers
- Incidentally, WG21 and WG23 member (although I've been really busy recently)
  - Involved in SG14, among other study groups
  - Occasional WG21 secretary
- And so on…

Who am I?

- Father of five, ages 27 to 9
- Feeds and cleans up after a varying number of animals
  - Look for *Paws of Britannia* with your favorite search engine
- Used to write military flight simulator code, among other things
  - CAE Electronics Ltd, IREQ
- **Full-time teacher since1998**
  - **Collège Lionel-Groulx, Université de Sherbrooke**
  - Works a lot with game programmers
- Incidentally, WG21 and WG23 member (although I've been really busy recently)
  - Involved in SG14, among other study groups
  - Occasional WG21 secretary
- And so on…

WE HAVE A PROBLEM

The situation

```
const unsigned short MIN = 0,
                     MAX = 65535;
struct IdGenerator {
   virtual unsigned short Take() = 0;
   virtual unsigned short Give(Id) = 0;
};
class SequentialGenerator : IdGenerator {
   unsigned short currentValue = MIN;
   string prefix;
public:
   SequentialGenerator() { prefix = ""; }
   SequentialGenerator(string pref) { prefix = pref; }
   // ...
```

```
// ...
Id Take() {
      try {
         if (currentValue > MAX)
            throw new NoIdLeftException();
      } catch (NoIdLeftException*) {
         cout << "No id left" << endl;
         Id id = Id(currentValue, prefix);
         currentValue++;
         return id;
      }
   }
   public void Give(Id id) {    }
}
```

The situation

- This is an excerpt from an assignment that one student sent to me this week
  - There's much more in there; this is a small bit
  - From that specific group, I'll be reading ~20 such assignments
  - This is a representative sample
    - There are better ones
    - There are worse ones

The situation

```
const unsigned short MIN = 0,
                     MAX = 65535;
struct IdGenerator {
   virtual unsigned short Take() = 0;
   virtual unsigned short Give(Id) = 0;
};
class SequentialGenerator : IdGenerator {
   unsigned short currentValue = MIN;
   string prefix;
public:
   SequentialGenerator() { prefix = ""; }
   SequentialGenerator(string pref) { prefix = pref; }
   // ...
```

```
// ...
Id Take() {
      try {
         if (currentValue > MAX)
            throw new NoIdLeftException();
      } catch (NoIdLeftException*) {
         cout << "No id left" << endl;
         Id id = Id(currentValue, prefix);
         currentValue++;
         return id;
      }
   }
   public void Give(Id id) {    }
}
```

The situation

- You might be wincing, and you might want to criticize

The situation

- You might be wincing, and you might want to criticize
- Be careful: this is normal code at some stages of development
  - There’s information in that code
    - About the individual
    - About the individual’s background
    - About the individual’s understanding of things at this stage

The situation

- You might be wincing, and you might want to criticize
- Be careful: this is normal code at some stages of development
  - There's information in that code
    - About the individual
    - About the individual's background
    - About the individual's understanding of things at this stage
  - There are cultural references in that code
  - There's indications that will allow us to (try to) help that person get better at programming

The situation

```
const unsigned short MIN = 0,
                     MAX = 65535;
struct IdGenerator {
   virtual unsigned short Take() = 0;
   virtual unsigned short Give(Id) = 0;
};
class SequentialGenerator : IdGenerator {
   unsigned short currentValue = MIN;
   string prefix;
public:
   SequentialGenerator() { prefix = ""; }
   SequentialGenerator(string pref) { prefix = pref; }
   // ...
```

```
// ...
Id Take() {
      try {
         if (currentValue > MAX)
            throw new NoIdLeftException();
      } catch (NoIdLeftException*) {
         cout << "No id left" << endl;
         Id id = Id(currentValue, prefix);
         currentValue++;
         return id;
      }
   }
   public void Give(Id id) {    }
}
```

The situation

If one has to provide feedback on such code, context should influence the way that feedback is conveyed

```
const unsigned short MIN = 0,
                     MAX = 65535;
struct IdGenerator {
   virtual unsigned short Take() = 0;
   virtual unsigned short Give(Id) = 0;
};
class SequentialGenerator : IdGenerator {
   unsigned short currentValue = MIN;
   string prefix;
public:
   SequentialGenerator() { prefix = ""; }
   SequentialGenerator(string pref) { prefix = pref; }
   // ...
```

```
// ...
Id Take() {
      try {
         if (currentValue > MAX)
            throw new NoIdLeftException();
      } catch (NoIdLeftException*) {
         cout << "No id left" << endl;
         Id id = Id(currentValue, prefix);
         currentValue++;
         return id;
      }
   }
   public void Give(Id id) {    }
}
```

The situation

There are many kinds of issues here. If the code was produced by a co-worker, or if it was produced by a first year student, we might not insist on the same aspects

```
const unsigned short MIN = 0,
                     MAX = 65535;
struct IdGenerator {
   virtual unsigned short Take() = 0;
   virtual unsigned short Give(Id) = 0;
};
class SequentialGenerator : IdGenerator {
   unsigned short currentValue = MIN;
   string prefix;
public:
   SequentialGenerator() { prefix = ""; }
   SequentialGenerator(string pref) { prefix = pref; }
   // ...
```

```
// ...
Id Take() {
      try {
         if (currentValue > MAX)
            throw new NoIdLeftException();
      } catch (NoIdLeftException*) {
         cout << "No id left" << endl;
         Id id = Id(currentValue, prefix);
         currentValue++;
         return id;
      }
   }
   public void Give(Id id) {    }
}
```

The situation

There are issues of hygiene, things that should be routinely done

```
const unsigned short MIN = 0,
                     MAX = 65535;
struct IdGenerator {
   virtual unsigned short Take() = 0;
   virtual unsigned short Give(Id) = 0;
};
class SequentialGenerator : IdGenerator {
   unsigned short currentValue = MIN;
   string prefix;
public:
   SequentialGenerator() { prefix = ""; }
   SequentialGenerator(string pref) { prefix = pref; }
   // ...
```

```
// ...
Id Take() {
      try {
         if (currentValue > MAX)
            throw new NoIdLeftException();
      } catch (NoIdLeftException*) {
         cout << "No id left" << endl;
         Id id = Id(currentValue, prefix);
         currentValue++;
         return id;
      }
   }
   public void Give(Id id) {     }
}
```

The situation

There are issues of hygiene, things that should be routinely done

```
const unsigned short MIN = 0,
                     MAX = 65535;
struct IdGenerator {
   virtual unsigned short Take() = 0;
   virtual unsigned short Give(Id) = 0;
};
class SequentialGenerator : IdGenerator {
   unsigned short currentValue = MIN;
   string prefix;
public:
   SequentialGenerator() { prefix = ""; }
   SequentialGenerator(string pref) { prefix = pref; }
   // ...
```

```
// ...
Id Take() {
      try {
         if (currentValue > MAX)
            throw new NoIdLeftException();
      } catch (NoIdLeftException*) {
         cout << "No id left" << endl;
         Id id = Id(currentValue, prefix);
         currentValue++;
         return id;
      }
   }
   public void Give(Id id) {     }
}
```

The situation

There are issues related to understanding the role of objects and the role of client code

```
const unsigned short MIN = 0,
                     MAX = 65535;
struct IdGenerator {
   virtual unsigned short Take() = 0;
   virtual unsigned short Give(Id) = 0;
};
class SequentialGenerator : IdGenerator {
   unsigned short currentValue = MIN;
   string prefix;
public:
   SequentialGenerator() { prefix = ""; }
   SequentialGenerator(string pref) { prefix = pref; }
   // ...
```

```
// ...
Id Take() {
      try {
         if (currentValue > MAX)
            throw new NoIdLeftException();
      } catch (NoIdLeftException*) {
         cout << "No id left" << endl;
         Id id = Id(currentValue, prefix);
         currentValue++;
         return id;
      }
   }
   public void Give(Id id) {    }
}
```

The situation

There are issues related to understanding the role of objects and the role of client code

```
const unsigned short MIN = 0,
                     MAX = 65535;
struct IdGenerator {
   virtual unsigned short Take() = 0;
   virtual unsigned short Give(Id) = 0;
};
class SequentialGenerator : IdGenerator {
   unsigned short currentValue = MIN;
   string prefix;
public:
   SequentialGenerator() { prefix = ""; }
   SequentialGenerator(string pref) { prefix = pref; }
   // ...
```

```
// ...
Id Take() {
      try {
         if (currentValue > MAX)
            throw new NoIdLeftException();
      } catch (NoIdLeftException*) {
          cout << "No id left" << endl;
         Id id = Id(currentValue, prefix);
         currentValue++;
         return id;
      }
   }
   public void Give(Id id) {    }
}
```

The situation

There are issues with understanding object lifecycle and memory management

```
const unsigned short MIN = 0,
                     MAX = 65535;
struct IdGenerator {
   virtual unsigned short Take() = 0;
   virtual unsigned short Give(Id) = 0;
};
class SequentialGenerator : IdGenerator {
   unsigned short currentValue = MIN;
   string prefix;
public:
   SequentialGenerator() { prefix = ""; }
   SequentialGenerator(string pref) { prefix = pref; }
   // ...
```

```
// ...
Id Take() {
      try {
         if (currentValue > MAX)
            throw new NoIdLeftException();
      } catch (NoIdLeftException*) {
         cout << "No id left" << endl;
         Id id = Id(currentValue, prefix);
         currentValue++;
         return id;
      }
   }
   public void Give(Id id) {     }
}
```

The situation

There are issues with understanding object lifecycle and memory management

```
const unsigned short MIN = 0,
                     MAX = 65535;
struct IdGenerator {
   virtual unsigned short Take() = 0;
   virtual unsigned short Give(Id) = 0;
};
class SequentialGenerator : IdGenerator {
   unsigned short currentValue = MIN;
   string prefix;
public:
   SequentialGenerator() { prefix = ""; }
   SequentialGenerator(string pref) { prefix = pref; }
   // ...
```

```
// ...
Id Take() {
      try {
         if (currentValue > MAX)
              throw new NoIdLeftException();
      } catch (NoIdLeftException*) {
         cout << "No id left" << endl;
         Id id = Id(currentValue, prefix);
         currentValue++;
         return id;
      }
   }
   public void Give(Id id) {    }
}
```

The situation

There are issues with understanding how/when to use control structures

```
const unsigned short MIN = 0,
                     MAX = 65535;
struct IdGenerator {
   virtual unsigned short Take() = 0;
   virtual unsigned short Give(Id) = 0;
};
class SequentialGenerator : IdGenerator {
   unsigned short currentValue = MIN;
   string prefix;
public:
   SequentialGenerator() { prefix = ""; }
   SequentialGenerator(string pref) { prefix = pref; }
   // ...
```

```
// ...
Id Take() {
      try {
         if (currentValue > MAX)
            throw new NoIdLeftException();
      } catch (NoIdLeftException*) {
         cout << "No id left" << endl;
         Id id = Id(currentValue, prefix);
         currentValue++;
         return id;
      }
   }
   public void Give(Id id) {    }
}
```

The situation

There are issues with understanding how/when to use control structures

```
const unsigned short MIN = 0,
                     MAX = 65535;
struct IdGenerator {
   virtual unsigned short Take() = 0;
   virtual unsigned short Give(Id) = 0;
};
class SequentialGenerator : IdGenerator {
   unsigned short currentValue = MIN;
   string prefix;
public:
   SequentialGenerator() { prefix = ""; }
   SequentialGenerator(string pref) { prefix = pref; }
   // ...
```

```
// ...
Id Take() {
      try {
          if (currentValue > MAX)
             throw new NoIdLeftException();
       } catch (NoIdLeftException*) {
          cout << "No id left" << endl;
          Id id = Id(currentValue, prefix);
          currentValue++;
          return id;
       }
   }
   public void Give(Id id) {    }
}
```

The situation

There are minor issues one looks at when one wants to make working code better

```
const unsigned short MIN = 0,
                     MAX = 65535;
struct IdGenerator {
   virtual unsigned short Take() = 0;
   virtual unsigned short Give(Id) = 0;
};
class SequentialGenerator : IdGenerator {
   unsigned short currentValue = MIN;
   string prefix;
public:
   SequentialGenerator() { prefix = ""; }
   SequentialGenerator(string pref) { prefix = pref; }
   // ...
```

```
// ...
Id Take() {
      try {
         if (currentValue > MAX)
            throw new NoIdLeftException();
      } catch (NoIdLeftException*) {
         cout << "No id left" << endl;
         Id id = Id(currentValue, prefix);
         currentValue++;
         return id;
      }
   }
   public void Give(Id id) {     }
}
```

The situation

There are minor issues one looks at when one wants to make working code better

```
const unsigned short MIN = 0,
                     MAX = 65535;
struct IdGenerator {
   virtual unsigned short Take() = 0;
   virtual unsigned short Give(Id) = 0;
};
class SequentialGenerator : IdGenerator {
   unsigned short currentValue = MIN;
    string prefix;
public:
    SequentialGenerator() { prefix = ""; }
   SequentialGenerator(string pref) { prefix=pref; }
   // ...
```

```
// ...
Id Take() {
      try {
         if (currentValue > MAX)
            throw new NoIdLeftException();
      } catch (NoIdLeftException*) {
         cout << "No id left" << endl;
         Id id = Id(currentValue, prefix);
         currentValue++;
         return id;
      }
   }
   public void Give(Id id) {    }
}
```

The situation

There are some things that are just dangerous

```
const unsigned short MIN = 0,
                     MAX = 65535;
struct IdGenerator {
   virtual unsigned short Take() = 0;
   virtual unsigned short Give(Id) = 0;
};
class SequentialGenerator : IdGenerator {
   unsigned short currentValue = MIN;
   string prefix;
public:
   SequentialGenerator() { prefix = ""; }
   SequentialGenerator(string pref) { prefix = pref; }
   // ...
```

```
// ...
Id Take() {
      try {
         if (currentValue > MAX)
            throw new NoIdLeftException();
      } catch (NoIdLeftException*) {
         cout << "No id left" << endl;
         Id id = Id(currentValue, prefix);
         currentValue++;
         return id;
      }
   }
   public void Give(Id id) {    }
}
```

The situation

There are some things that are just dangerous

```
const unsigned short MIN = 0,
                         MAX = 65535;
struct IdGenerator {
   virtual unsigned short Take() = 0;
   virtual unsigned short Give(Id) = 0;
}; // no virtual dtor
class SequentialGenerator : IdGenerator {
   unsigned short currentValue = MIN;
   string prefix;
public:
   SequentialGenerator() { prefix = ""; }
   SequentialGenerator(string pref) { prefix = pref; }
   // ...
```

```
// ...
Id Take() {
      try {
         if (currentValue > MAX)
            throw new NoIdLeftException();
      } catch (NoIdLeftException*) {
         cout << "No id left" << endl;
         Id id = Id(currentValue, prefix);
           currentValue++;
         return id;
      }
   }
   public void Give(Id id) {    }
}
```

The situation

- It’s difficult for a beginner to absorb all of that information at once
  - Even if you show and explain things, chances are you will have to do it again soon
    - Stress…
    - Information overload…

The situation

- It’s difficult for a beginner to absorb all of that information at once
  - Even if you show and explain things, chances are you will have to do it again soon
    - Stress…
    - Information overload…
  - Providing feedback requires some measure of patience and kindness

The situation

- Programmers require feedback

The situation

- Programmers require feedback
- In particular, beginner programmers require feedback
  - This includes both junior programmers and students who program, be they students in computer science, software engineering or other fields involving programming

The situation

- **Providing** this feedback can be tedious
  - Requires time and attention
  - Difficult to do
  - … but essential, and too often neglected

The situation

- **Receiving** this feedback can be difficult
  - We are more emotionally involved in our code than many suspect
  - Criticism of one’s work can make one sensitive

ON A PERSONAL NOTE

We all need feedback

- True story… From a few days ago…

We all need feedback

```
// we are going to be storing at most N
// instances of T in buf, but we don't
// want to create the T instances
template <int N, class T, class F>
auto store_then_use(F f, bool(*read_func)(char*,int)) {
   // ...
}
```

We all need feedback

```
// we are going to be storing at most N instances of T in buf, but we don't
// want to create the T instances
template <int N, class T, class F>
auto store_then_use(F f, bool(*read_func)(char*,int)) {
   alignas(T) char buf[N * sizeof(T)];
   if (!read_func(buf+0, sizeof buf))
      throw read_error{};
   else {
      // technically, UB, and incomplete (T objects not finalized). Shhh...
      return f(*reinterpret_cast<T*>(buf+0), N);
   }
}
```

We all need feedback

```
// we are going to be storing at most N instances of T in buf, but we don't
// want to create the T instances
template <int N, class T, class F>
auto store_then_use(F f, bool(*read_func)(char*,int)) {
   alignas(T) char buf[N * sizeof(T)];
   if (!read_func(buf+0, sizeof buf))
      throw read_error{};
   else {
      // technically, UB, and incomplete (T objects not finalized). Shhh...
      return f(*reinterpret_cast<T*>(buf+0), N);
   }
}
```

We all need feedback

```
// we are going to be storing at most N instances of T in buf, but we don't
// want to create the T instances
template <int N, class T, class F>
auto store_then_use(F f, int(*read_func)(char*,int)) {
   alignas(T) char buf[N * sizeof(T)];
   if (int n = read_func(buf+0, sizeof buf); n == 0)
      throw read_error{};
   else {
      // technically, UB, and incomplete (T objects not finalized). Shhh...
      return f(*reinterpret_cast<T*>(buf+0), n);
   }
}
```

We all need feedback

- I was the culprit, explaining something unrelated in one of my classes

We all need feedback

- I was the culprit, explaining something unrelated in one of my classes
- It’s the kind of thing I would have caught right away in someone else’s code
  - When I reviewed my material, I skimmed over it, as it was from a class I had already given in the past
  - … when in class, this time, it just jumped to me, live, as I was presenting
  - One of the (bright!) participants in my class saw it too

We all need feedback

- I was the culprit, explaining something unrelated in one of my classes
- It’s the kind of thing I would have caught right away in someone else’s code
  - When I reviewed my material, I skimmed over it, as it was from a class I had already given in the past
  - … when in class, this time, it just jumped to me, live, as I was presenting
  - One of the (bright!) participants in my class saw it too
- We turned it into a design discussion
  - I fixed it during lunch time (whew!)

We All Need Feedback

- I write code and course material every day
- My colleagues read and criticize my work
  - I'm grateful for this!
  - I do the same for them
- I have done this for almost a quarter of a century
- I still find stuff that's defective in material used dozens of time
  - … it makes me happy!
  - … it also makes me happy when people find bugs in my code
- *We all need feedback*

A PRIORI

A priori

- This presentation supposes a few things
  - A willingness on the part of the providers to help the receivers
  - A willingness on the part of the receivers to learn and get better at what they do
  - A welcoming environment for those who work or study there
- This presentation supposes goodwill, really

NAMES AND DEFINITIONS

Names and definitions

- In order to make this talk more objective and palatable, a few definitions
- From the perspective of a code review:
  - The **provider** or **providers** are those reviewing the code
  - The **receiver** or **receivers** are those who produced the code under review
  - I will use **company** for the place one works in
    - Companies have needs, requirements, preferences, culture, etc.
  - I will use **training profile** for academic contexts
    - Some considerations are important to all programmers, but others depend on what a class / a training / a learning program are trying to convey / prepare individuals for
    - This may be representative of some internship situations

WHAT?

What?

- In a way, providing feedback is akin to a debugging experience
    - It can be done *ad hoc*, but it's generally more efficient if one is prepared
- It's hard (impossible?) to make a complete list of aspects to prepare for
    - ...but we can open some avenues for reflection

What?

- What is important to me as a provider?
- What do I want to look for?
- What would benefit the receiver the most?
- What would benefit the company / the training profile the most?
- What is essential for the code under review to be acceptable

What?

- **What is important to me as a provider?**
  - My experience counts. It’s why I’m asked to provide feedback
  - I know others who will read / use the code
  - I know the context to which the code is destined and can guide the receiver towards it
  - I have aesthetical / style preferences I might want to convey

What?

- **What do I want to look for?**
  - Maybe I know the receiver's strengths and blind spots
  - Maybe I know the system's constraints
  - Maybe I'm training someone and the code is an answer to a problem prepared to bring about some enlightenment
  - Maybe the respect of some style guide is important to my company and that needs to be conveyed

What?

- **What would benefit the receiver the most?**
  - Given what I know of the receiver, what would be the feedback I should insist on?
  - What are the things I could mention but that are less essential to the receiver’s personal or professional development?
  - What skills do I think my feedback could help the receiver improve?

What?

- **What would benefit the company / the training profile the most?**
  - Does my company do real-time or low-latency software? If so, I might focus on deterministic behavior and predictable performance
  - Does my company expose an API to external clients? If so, I might focus on usability and security

What?

- **What would benefit the company / the training profile the most?**
  - Am I training beginners? If so, I will want to ensure that what has been discussed in class is well understood, and I might « abstract away » problems the receiver would not be aware of or know how to solve
  - Am I training intermediate-level students? I might consider some things known to them (e.g.: indentation) and be a bit more severe if the code seems negligent in those regards, and I might be pay more attention to design issues (*but grade them less agressively*)
  - In my class, I might want to insist on the idioms and good practices of the language(s) we use to solve problems

What?

- **What is essential for the code under review to be acceptable**
  - There are requirements that need to be met
    - Things without which the system will not run properly
    - Aspects without which the classes following mine will be harder to teach
  - There might be aspects that will need to be addressed at some point but are not urgent

What?

- **What is important to me as a provider?**
- **What do I want to look for?**
- **What would benefit the receiver the most?**
- **What would benefit the company / the training profile the most?**
- **What is essential for the code under review to be acceptable**

As a provider, I need to prioritize and make choices on each of these aspects

For this, I need to know what I am trying to achieve

HOW?

How?

- How to review code is a question with many aspects
- Some aspects concern the provider
  - How to be understood?
  - How to provide feedback in a human and constructive manner?
  - How to be efficient?
- Some aspects concern the receiver
  - How to use the feedback efficiently?
  - How to ask questions in order to get the most from the experience
  - How to use feedback for personal and professional growth?

How?

- For the provider
  - Choosing what to focus on
    - Make a plan!
    - Define objectives
  - Doing it with reasonable effort
    - Tricks and techniques
  - Being effective
    - Being understood
    - Being kind

How? Choosing what to focus on

- For the provider
  - Choosing what to focus on
    - Make a plan!
    - Define objectives

See the « What » section, on previous slides

How? Choosing what to focus on

- For the provider
  - Choosing what to focus on
    - Make a plan!
    - Define objectives

See the « What » section, on previous slides

If the code review is a **live experience**, one thing that can help is reading the code prior to the meeting and having knowledge of the code's prior incarnation (if any)

How? Choosing what to focus on

- For the provider
  - Choosing what to focus on
    - Make a plan!
    - Define objectives

See the « What » section, on previous slides

The best way to get satisfying code is to be clear a priori on what the expectations are

If the code review is a **live experience**, one thing that can help is reading the code prior to the meeting and having knowledge of the code's prior incarnation (if any)

How? Choosing what to focus on

- For the provider
  - Choosing what to focus on
    - Make a plan!
    - Define objectives

See the « What » section, on previous slides

If the code review is a **written experience** (e.g.: grading assignments), then ensuring one only grades a combination of what's supposed to be known a priori and what has been taught / announced beforehand is important. Otherwise, one can add remarks, comments, but going further would be problematic…

How? Choosing what to focus on

- For the provider
  - Choosing what to focus on
    - Make a plan!
    - Define objectives

See the « What » section, on previous slides

If the code review is a **written experience** (e.g. grading assignments), then ensuring one only grades a combination of what's supposed to be known a priori and what has been taught / announced beforehand is important. Otherwise, one can add remarks, comments, but going further would be problematic…

Students should know how they will be evaluated before they begin working on an assignment

How? Doing it with reasonable effort

- For the provider
  - Doing it with reasonable effort
    - Tricks and techniques

If the code review is a **group effort**, one can assemble a small group of people with a common set of expectations, but different skills

How? Doing it with reasonable effort

- For the provider
  - Doing it with reasonable effort
    - Tricks and techniques

If the code review is a **group effort**, one can assemble a small group of people with a common set of expectations, but different skills

Rare is the polymath programmer, good at everything. Most of us have blind spots, and it's just normal

How? Doing it with reasonable effort

- For the provider
  - Doing it with reasonable effort
    - Tricks and techniques

For a « live » code review, applying some basic rules of meeting management definitely helps

- Be there on time
- Be ready on time
- Set a time limit
- Make sure the expectations (if any) are clear for everyone at the end of the meeting
- If possible, put them in writing!

How? Doing it with reasonable effort

- For the provider
  - Doing it with reasonable effort
    - Tricks and techniques

Written code reviews can be tedious and delicate

- **Tedious** because to be understood, one has to be clear

How? Doing it with reasonable effort

- For the provider
  - Doing it with reasonable effort
    - Tricks and techniques

Written code reviews can be tedious and delicate

- Tedious because to be understood, one has to be clear
- This can mean writing long phrases, providing examples of what is expected, rewriting excerpts, etc.

How? Doing it with reasonable effort

- For the provider
  - Doing it with reasonable effort
    - Tricks and techniques

Written code reviews can be tedious and delicate

- Tedious because to be understood, one has to be clear
- This can mean writing long phrases, providing examples of what is expected, rewriting excerpts, etc.
- **Delicate** because the provider does not necessarily see the receiver when the feedback is being received

How? Doing it with reasonable effort

- For the provider
  - Doing it with reasonable effort
    - Tricks and techniques

Written code reviews can be tedious and delicate

- Tedious because to be understood, one has to be clear
- This can mean writing long phrases, providing examples of what is expected, rewriting excerpts, etc.
- Delicate because the provider does not necessarily see the receiver when the feedback is being received
- Written feedback can seem dry (no facial expressions, less empathy)

How? Doing it with reasonable effort

- For the provider
  - Doing it with reasonable effort
    - Tricks and techniques

Written code reviews can be tedious and delicate

- Tedious because to be understood, one has to be clear
- This c[illegible]ean writing long phrases, providing [illegible]ples of what is expected, rewriting excerpts, etc.
- [illegible]use the provider does not necessarily see [illegible]hen the feedback is being received
- [illegible]ack can seem dry (no facial expressions,

In a situation where the same remarks tend to come up regularly (e.g.: with students), a good trick is to have a set of **short mnemonics** that convey the essence of the message, and that receivers can understand quickly

How? Doing it with reasonable effort

- Examples of mnemonics I use
  - CMT
  - CNS
  - DANGER
  - FR
  - IND
  - MOD
  - NFP
  - NRC
  - NS
  - TYPE

How? Doing it with reasonable effort

- Examples of mnemonics I use
  - CMT
  - CNS
  - DANGER
  - FR
  - IND
  - MOD
  - NFP
  - NRC
  - NS
  - TYPE

Your list may, of course, differ from mine!

How? Doing it with reasonable effort

- Examples of mnemonics I use
    - **CMT – problem with comments, e.g.: one would have been useful, one is misleading**
    - CNS
    - DANGER
    - FR
    - IND
    - MOD
    - NFP
    - NRC
    - NS
    - TYPE

How? Doing it with reasonable effort

- Examples of mnemonics I use
  - CMT
  - **CNS – problem with constants, e.g.: a « magic number » where this does not seem justified**
  - DANGER
  - FR
  - IND
  - MOD
  - NFP
  - NRC
  - NS
  - TYPE

How? Doing it with reasonable effort

- Examples of mnemonics I use
  - CMT
  - CNS
  - **DANGER – this is just dangerous**
  - FR
  - IND
  - MOD
  - NFP
  - NRC
  - NS
  - TYPE

How? Doing it with reasonable effort

- Examples of mnemonics I use
  - CMT
  - CNS
  - DANGER
  - **FR – I have a hard time reading your prose (FR is for « français » as I teach mostly in French)**
  - IND
  - MOD
  - NFP
  - NRC
  - NS
  - TYPE

How? Doing it with reasonable effort

- Examples of mnemonics I use
  - CMT
  - CNS
  - DANGER
  - FR
  - **IND – misleading indentation**
  - MOD
  - NFP
  - NRC
  - NS
  - TYPE

How? Doing it with reasonable effort

- Examples of mnemonics I use
  - CMT
  - CNS
  - DANGER
  - FR
  - IND
  - **MOD – short for « modularization », I use this for code structure issues (badly written loops, missing functions, functions or classes that do too much, etc.)**
  - NFP
  - NRC
  - NS
  - TYPE

How? Doing it with reasonable effort

- Examples of mnemonics I use
  - CMT
  - CNS
  - DANGER
  - FR
  - IND
  - MOD
  - **NFP – that just does not work (French: « ne fonctionne pas », hence the mnemonic)**
  - NRC
  - NS
  - TYPE

How? Doing it with reasonable effort

- Examples of mnemonics I use
  - CMT
  - CNS
  - DANGER
  - FR
  - IND
  - MOD
  - NFP
  - **NRC – does not do what has been asked (stands for « non-respect des consignes », not respectful of the stated requirements)**
  - NS
  - TYPE

How? Doing it with reasonable effort

- Examples of mnemonics I use
  - CMT
  - CNS
  - DANGER
  - FR
  - IND
  - MOD
  - NFP
  - NRC
  - **NS – names that are misleading or just not significant enough**
  - TYPE

How? Doing it with reasonable effort

- Examples of mnemonics I use
  - CMT
  - CNS
  - DANGER
  - FR
  - IND
  - MOD
  - NFP
  - NRC
  - NS
  - **TYPE – badly picked type, e.g.: an int where a bool would be reasonable, a bool where an enum would be cleared, etc.**

How? Doing it with reasonable effort

- Examples of mnemonics I use
  - CMT
  - CNS
  - DANGER
  - FR
  - IND
  - MOD
  - NFP
  - NRC
  - NS
  - TYPE

I also use ☺ when they make me smile, ☹ when they make me sad, and **?** when they make me confused

How? Doing it with reasonable effort

- Examples of mnemonics I use
  - CMT
  - CNS
  - DANGER
  - FR
  - IND
  - MOD
  - NFP
  - NRC
  - NS
  - TYPE

These cover ≈95%+ of the feedback on code I need to give in practice (for the rest, I take the time to write more detailed explanations). I take a red pen, trace an ellipsis or put arrows on the appropriate location(s) and write 2-5 capital letters

How? Doing it with reasonable effort

- Examples of mnemonics I use
  - CMT
  - CNS
  - DANGER
  - FR
  - IND
  - MOD
  - NFP
  - NRC
  - NS
  - TYPE

These cover ≈95%+ of the feedback on code I need to give in practice (for the rest, I take the time to write more detailed explanations). I ta trace an ellipsis or

When I grade code, my process is:

- I suppose the code is perfect (if it compiles and runs)

How? Doing it with reasonable effort

- Examples of mnemonics I use
  - CMT
  - CNS
  - DANGER
  - FR
  - IND
  - MOD
  - NFP
  - NRC
  - NS
  - TYPE

These cover ≈95%+ of the feedback on code I need to give in practice (for the rest, I take the time to write more detailed explanations). I ta trace an ellipsis or

When I grade code, my process is:

- I suppose the code is perfect (if it compiles and runs)
- Then, I read it

How? Doing it with reasonable effort

- Examples of mnemonics I use
  - CMT
  - CNS
  - DANGER
  - FR
  - IND
  - MOD
  - NFP
  - NRC
  - NS
  - TYPE

These cover ≈95%+ of the feedback on code I need to give in practice (for the rest, I take the time to write more detailed explanations). I ta[illegible] trace an ellipsis or

When I grade code, my process is:

- I suppose the code is perfect (if it compiles and runs)
- Then, I read it
- Then, I count the number of each such mnemonic I found to see how much I was mistaken (I sometimes stop counting if one appears too often; the idea is to help the student grow, not crush that person)

How? Doing it with reasonable effort

- Examples of mnemonics I use
  - CMT
  - CNS
  - DANGER
  - FR
  - IND
  - MOD
  - NFP
  - NRC
  - NS
  - TYPE

These cover ≈95%+ of the feedback on code I need to give in practice (for the rest, I take the time to write more detailed explanations). I ta trace an ellipsis or

When I grade code, my process is:

- I suppose the code is perfect (if it compiles and runs)
- Then, I read it
- Then, I count the number of each such mnemonic I found to see how much I was mistaken (I sometimes stop counting if one appears too often; the idea is to help the student grow, not crush that person)
- I assign a weight to each mnemonic (e.g.: dangerous code costs more than a bad choice of name), and substract from the initial, perfect score

How? Doing it with reasonable effort

- Examples of mnemonics I use
  - **CMT**
  - **CNS**
  - DANGER
  - **FR**
  - **IND**
  - MOD
  - **NFP**
  - **NRC**
  - **NS**
  - TYPE

These tend to come up even on first year students; the others appear typically when code becomes more elaborate

How? Doing it with reasonable effort

- Examples of mnemonics I use
  - CMT
  - CNS
  - DANGER
  - FR
  - IND
  - **MOD**
  - NFP
  - NRC
  - NS
  - TYPE

I call the **MOD** cases « erreurs de grande personne » (grown person’s mistakes) since they demand some thinking with respect to design, code structure, code organization, etc. and hopefully are opportunities to grow as a programmer

How? Doing it with reasonable effort

- Examples of mnemonics I use
  - CMT
  - CNS
  - DANGER
  - FR
  - IND
  - MOD
  - **NFP**
  - **NRC**
  - NS
  - TYPE

These two are probably the most annoying:

- **NFP** often comes up when they sent me something that does not compile or something they did not test
- **NRC** typically means they did not read the assignment seriously…

How? Doing it with reasonable effort

```
const unsigned short MIN = 0,
                     MAX = 65535;
struct IdGenerator {
   virtual unsigned short Take() = 0;
   virtual unsigned short Give(Id) = 0;
};
class SequentialGenerator : IdGenerator {
   unsigned short currentValue = MIN;
   string prefix;
public:
   SequentialGenerator() { prefix = ""; }
   SequentialGenerator(string pref) { prefix = pref; }
   // ...
```

```
// ...
Id Take() {
      try {
         if (currentValue > MAX)
            throw new NoIdLeftException();
      } catch (NoIdLeftException*) {
         cout << "No id left" << endl;
         Id id = Id(currentValue, prefix);
         currentValue++;
         return id;
      }
   }
   public void Give(Id id) {    }
}
```

How? Doing it with reasonable effort

DANGER

```
const unsigned short MIN = 0,
                     MAX = 65535;
struct IdGenerator {
   virtual unsigned short Take() = 0;
   virtual unsigned short Give(Id) = 0;
};
class SequentialGenerator : IdGenerator {
   unsigned short currentValue = MIN;
   string prefix;
public:
   SequentialGenerator() { prefix = ""; }
   SequentialGenerator(string pref) { prefix = pref; }
   // ...
```

We can do better, let's talk

NFP

TYPE

MOD

```
// ...
Id Take() {
    try {
      if (currentValue > MAX)
        throw new NoIdLeftException();
    } catch (NoIdLeftException*) {
      cout << "No id left" << endl;
      Id id = Id(currentValue, prefix);
      currentValue++;
      return id;
    }
  }
  public void Give(Id id) {  }
}
```

Not the right place

I have a trick for you ☺

How? Being effective

- For the provider
  - Being effective
    - Being understood
    - Being kind

How? Being effective

- For the provider
  - Being effective
    - **Being understood**
    - Being kind

I could not stress this enough: if the receiver does not understand the intent behind the feedback, it’s wasted time for everyone involved

How? Being effective

- For the provider
  - Being effective
    - **Being understood**
    - Being kind

I could not stress this enough: if the receiver does not understand the intent behind the feedback, it's wasted time for everyone involved

It's tempting to blame the receiver for misunderstanding feedback, but clarity of intent is typically a shared burden, and for the code review to bear fruits, being understood is essential

How? Being effective

- For the provider
  - Being effective
    - **Being understood**
    - Being kind

I could not stress this enough: if the receiver does not understand the intent behind the feedback, it's wasted time for everyone involved

It's tempting to blame the receiver for misunderstanding feedback, but clarity of intent is typically a shared burden, and for the code review to bear fruits, being underst[illegible] essential

When in doubt, don't just ask if the receiver understood; verify, ask specific questions

How? Being effective

- For the provider
  - Being effective
    - Being understood
    - **Being kind**

We must never forget that we're talking to a human being, and that we do not know what's going on in the receiver's life

How? Being effective

- For the provider
  - Being effective
    - Being understood
    - **Being kind**

We must never forget that we're talking to a human being, and that we do not know what's going on in the receiver's life

The ultimate objective is for the receiver to perfect his or her skills, to get better, to contribute… Not to go hire someone else

How? Being effective

- For the provider
  - Being effective
    - Being understood
    - **Being kind**

We must never forget that we're talking to a human being, and that we do not know what's go[illegible] on in the receiver's life

The ultimate objective is for the receiver to perfect his or her skills, to get better, to contribute... Not t[illegible] hire someone else

If the receiver does not seem to be in a good mental state for a code review, or **if the provider is not in a good mental state**, it might be better to postpone the effort, or to do the review through some other means

How?

- For the receiver
  - Doing one’s homework
    - Try to understand what you’ve done
    - Try to explain it clearly
      - Ideally, the code should not need much explaining, but still…
    - Re-read your own code before the review
  - Simplify the process
    - Respect your environment’s standards
    - Respect the directives you’ve been given
  - Being effective
    - Be open to feedback
    - Take it as a learning and growing opportunity

How? Doing one’s homework

- For the receiver
  - Doing one’s homework
    - Try to understand what you’ve don
    - Try to explain it clearly
      - Ideally, the code should not need much explai still…
    - Re-read your own code before the review

We cannot expect a beginner to be an expert, but we can ask the receiver to be able to re-read the code to be reviewed before the review effort occurs (or before submitting an assignment)

How? Doing one’s homework

- For the receiver
  - Doing one’s homework
    - Try to understand what you’ve don
    - Try to explain it clearly
      - Ideally, the code should not need much explai …still…
    - Re-read your own code before the review

We cannot expect a beginner to be an expert, but we can ask the receiver to be able to re-read the code to be reviewed before the review effort occurs (or before submitting an assignment)

Alternatively, as a friend to read it and criticize if that’s reasonable

How? Doing one’s homework

- For the receiver
  - Doing one’s homework
    - Try to understand what you’ve don
    - Try to explain it clearly
      - Ideally, the code should not need much explai still…
    - Re-read your own code before the review

We cannot expect a beginner to be an expert, but we can ask the receiver to be able to re-read the code to be reviewed before the review effort occurs (or before submitting an assignment)

Alternatively, as a friend to read it and criticize if that’s reasonable

We would hope that code would be understandable on its own, without additional explanations, but making the effort of explaining things may bring to light incoherences, confusing names, functions or classes with unclear roles, etc. which can save time and effort overall

How? Simplify the process

- For the receiver
  - Simplify the process
    - Respect your environment’s standards
    - Respect the directives you’ve been given

How? Simplify the process

- For the receiver
  - Simplify the process
    - Respect your environment's standards
    - Respect the directives you've been given

One can help the process by doing some work beforehand: identify the standards in your environment and strive to respect them

How? Simplify the process

- For the receiver
  - Simplify the process
    - Respect your environment’s standards
    - Respect the directives you’ve been given

One can help the process by doing some work beforehand: identify the standards in your environment and strive to respect them

This will save everyone time by reducing the « mechanical » part of the review, e.g.:respect the local casing protocol (snake, pascal, other), comment format, naming practices, etc.

How? Simplify the process

- For the receiver
  - Simplify the process
    - Respect your environment's standards
    - Respect the directives you've been given

One can help the process by doing some work beforehand: identify the standards in your environment and strive to respect them

This will save everyone time by reducing the « mechanical » part of the review, e.g.:respect the local casing protocol (snake, pascal, other), format, naming practices, etc.

Ideally, if you disagree with local practices, the code review is not the best time to address this

How? Simplify the process

- For the receiver
  - Simplify the process
    - Respect your environment’s standards
    - Respect the directives you’ve been given

Small, personal note: pay attention to your written prose. It saves *so much* time when one can be understood unambiguously…

How? Being effective

- For the receiver
  - Being effective
    - Be open to feedback
    - Take it as a learning and growing opportunity

How? Being effective

- For the receiver
  - Being effective
    - Be open to feedback
    - Take it as a learning and growing opportunity

It might seem intimidating to receive feedback on one's code, but it should not be

How? Being effective

- For the receiver
  - Being effective
    - Be open to feedback
    - Take it as a learning and growing opportu

It might seem intimidating to receive feedback on one's code, but it should not be

Evaluation, grading and feedback on one's efforts in general should be a positive thing, an opportunity to grow as a person and as a professional

How? Being effective

- For the receiver
  - Being effective
    - Be open to feedback
    - Take it as a learning and growing opportu

It might seem intimidating to receive feedback on one's code, but it should not be

Evaluation, grading and feedback on one's efforts in general should be a positive thing, an opportunity to grow as a person and as a professional

Everyone has a role in establishing a constructive ambience for feedback. The receiver's part is to be open, take notes, and use the opportunity to get better

SOME IDEAS

Some ideas…

- Here are a few ideas you might want to try or adapt to your own situation
  - Only if you feel like it and if you feel it might work for you
  - The list is just a start; feel free to write to me if you experiment and try things you think are interesting!

Some ideas…

- Expose the receiver to good code

Some ideas…

- Expose the receiver to good code
  - This might be done informally
    - Asking them to them make a small addition to a codebase you find particularly well done
    - Suggesting they watch a good conference (with you?)
    - Sharing interesting articles or tricks over lunch or coffee
    - Lending them a cool book where the programming practices seem recommendable

Some ideas…

- Expose the receiver to good code
  - This might be done informally
    - Asking them to make a small addition to a codebase you find particularly well done
    - Suggesting they watch a good conference (with you?)
    - Sharing interesting articles or tricks over lunch or coffee
    - Lending them a cool book where the programming practices seem recommendable
  - Learning by example can work
    - Make sure you don’t make the receiver feel diminished
    - Seek to create a form of empowerment, a sentiment of growth and caring

Some ideas…

- Exchange roles on occasion

Some ideas…

- Exchange roles on occasion
    - Ask the receivers for their opinion

Some ideas…

- Exchange roles on occasion
  - Ask the receivers for their opinion
  - Apply the socratic technique
    - Encourage them to express their opinion through questions
    - You might learn interesting things about them and their thought process

Some ideas…

- Exchange roles on occasion
  - Ask the receivers for their opinion
  - Apply the socratic technique
    - Encourage them to express their opinion through questions
    - You might learn interesting things about them and their thought process
  - The receiver will eventually have to review code
    - Being « on the other side of the fence » can help people understand each other better

Some ideas…

Crazy idea: you can make them review **your** code, from when you began working there (if there are still traces of it). *You don't even need to tell them it's yours!*

- Exchange roles on occasion
  - Ask the receivers for their opinion
  - Apply the socratic technique
    - Encourage them to express their opinion through questions
    - You might learn interesting things about them and their thought process
  - The receiver will eventually have to review code
    - Being « on the other side of the fence » can help people understand each other better

Some ideas…

Crazy idea: you can make them review **your** code, from when you began working there (if there are still traces of it). *You don't even need to tell them it's yours!*

- Exchange roles on occasion
  - Ask the receivers for their opinion
  - Apply the socratic technique
    - Encourage them to express their opinion through quest
    - You might learn interesting things about them and thei
  - The receiver will eventually have to review code
    - Being « on the other side of the fence » can help people understand each other better

Be humble, it's probably not all that good, so accept (encourage!) criticism

Some ideas…

- If you get the feeling the receiver is not progressing as you hoped, consider assigning that person a mentor

Some ideas…

- If you get the feeling the receiver is not progressing as you hoped, consider assigning that person a mentor
  - It does not have to be formal
  - Sometimes, just having someone else one can ask questions to makes a difference
  - If some coding ideas are discussed with a more experienced colleague beforehand, the code reviews might get more productive

Some ideas…

- If you get the feeling the receiver is not progressing as you hoped, consider assigning that person a mentor
  - It does not have to be formal
  - Sometimes, just having someone else one can ask questions to makes a difference
  - If some coding ideas are discussed with a more experienced colleague beforehand, the code reviews might get more productive
- This can also be useful if the receiver shows promise but seems isolated
  - Some people are uncomfortable with larger groups or with people in positions of authority
  - Informal mentoring is often more relaxed, like having a friend to talk to

Some ideas…

- If reviewing code with which you're not entirely satisfied, try suggesting alternatives

Some ideas…

- If reviewing code with which you’re not entirely satisfied, try suggesting alternatives
  - This requires more preparation

Some ideas…

- If reviewing code with which you’re not entirely satisfied, try suggesting alternatives
  - This requires more preparation
  - Might involve some design effort
    - Involve the receiver!
    - It’s a learning opportunity

Some ideas…

- If reviewing code with which you’re not entirely satisfied, try suggesting alternatives
  - This requires more preparation
  - Might involve some design effort
    - Involve the receiver!
    - It’s a learning opportunity
  - Make sure you ask of the receiver tasks that this person can accomplish
    - Giving someone tasks one cannot complete serves no good purpose

Some ideas…

- Stay healthy
- Stay kind
- Be human

THANK YOU