Professionalism in Programming

Alexander Stepanov

Abstract

Programming is about writing code. The code could be good or bad and it is not a matter of personal taste.

Programming is a profession. It requires constant professional education and professional ethics.

It is essential that organizational structures support writing of professional code and maintaining professional workforce.

```
Tracker& Tracker::GetTracker(void)
{
        // FIX_ME:  9/2/99 - Why is this here? It should be
        //explained with a
        // comment, or removed.
        if (!sTracker)
        {
                int foo = 44;
                foo++;
                Signal_("sTracker == NULL");
        }

        PPValidatePointer_(sTracker);

        return *sTracker;
}
```

```
bool PictureRadioButton::Track(Tracker& tracker)
{
	bool result = false;
	Action theAction = tracker.GetAction();
	switch (theAction)
	{
		case kButtonDownAction:
		{
			NRect localRect;
			NPoint point;
			bool needDraw = false;

			GetLocalRect(localRect);
			tracker.GetPoint(point);

			if (fButtonDown)
			{
				if (localRect.Contains(point))
				{
					if ((GetItemStyle() & kRadioButtonAllowNoneSetStyle) == 0)
						SetBooleanValue(true);
					else
					{
						SetBooleanValue(false);
						fButtonDown = false;
					}
				}
			}
			else
			{
				if (localRect.Contains(point))
				{
					if ((GetItemStyle() & kRadioButtonAllowNoneSetStyle) == 0)
						SetBooleanValue(true);
					else
					{
						SetBooleanValue(true);
						fButtonDown = true;
					}
				}
			}
			Invalidate();
			Update();

			result = true;
			break;
		}

	}
	return result;
}
```

```
bool PictureRadioButton::Track(Tracker& tracker)
{
	bool result = false;
	Action theAction = tracker.GetAction();
	switch (theAction)
	{
		case kButtonDownAction:
		{
			NRect localRect;
			NPoint point;
			bool needDraw = false;

			GetLocalRect(localRect);
			tracker.GetPoint(point);

			if (fButtonDown)
			{
				if (localRect.Contains(point))
				{
					if ((GetItemStyle() & kRadioButtonAllowNoneSetStyle) == 0)
						SetBooleanValue(true);
					else
					{
						SetBooleanValue(false);
						fButtonDown = false;
					}
				}
			}
			else
			{
				if (localRect.Contains(point))
				{
					if ((GetItemStyle() & kRadioButtonAllowNoneSetStyle) == 0)
						SetBooleanValue(true);
					else
					{
						SetBooleanValue(true);
						fButtonDown = true;
					}
				}
			}
			Invalidate();
			Update();

			result = true;
			break;
		}

	}
	return result;
}
```

```
bool PictureRadioButton::Track(Tracker& tracker)
{
	bool result = false;
	switch (tracker.GetAction())
	{
		case kButtonDownAction:
		{
			NRect localRect;
			NPoint point;
			bool needDraw = false;

			GetLocalRect(localRect);
			tracker.GetPoint(point);

			if (fButtonDown)
			{
				if (localRect.Contains(point))
				{
					if ((GetItemStyle() & kRadioButtonAllowNoneSetStyle) == 0)
						SetBooleanValue(true);
					else
					{
						SetBooleanValue(false);
						fButtonDown = false;
					}
				}
			}
			else
			{
				if (localRect.Contains(point))
				{
					if ((GetItemStyle() & kRadioButtonAllowNoneSetStyle) == 0)
						SetBooleanValue(true);
					else
					{
						SetBooleanValue(true);
						fButtonDown = true;
					}
				}
			}
			Invalidate();
			Update();

			result = true;
			break;
		}

	}
	return result;
}
```

```
bool PictureRadioButton::Track(Tracker& tracker)
{
	if (tracker.GetAction() != kButtonDownAction) return false;

	NRect localRect;
	NPoint point;
	bool needDraw = false;

	GetLocalRect(localRect);
	tracker.GetPoint(point);

	if (fButtonDown)
	{
		if (localRect.Contains(point))
		{
			if ((GetItemStyle() & kRadioButtonAllowNoneSetStyle) == 0)
				SetBooleanValue(true);
			else
			{
				SetBooleanValue(false);
				fButtonDown = false;
			}
		}
	}
	else
	{
		if (localRect.Contains(point))
		{
			if ((GetItemStyle() & kRadioButtonAllowNoneSetStyle) == 0)
				SetBooleanValue(true);
			else
			{
				SetBooleanValue(true);
				fButtonDown = true;
			}
		}
	}
	Invalidate();
	Update();
	return true;
}
```

```
bool PictureRadioButton::Track(Tracker& tracker)
{
	if (tracker.GetAction() != kButtonDownAction) return false;

	NRect localRect;
	NPoint point;
	bool needDraw = false;

	GetLocalRect(localRect);
	tracker.GetPoint(point);

	if (fButtonDown)
	{
		if (localRect.Contains(point))
		{
			if ((GetItemStyle() & kRadioButtonAllowNoneSetStyle) == 0)
				SetBooleanValue(true);
			else
			{
				SetBooleanValue(false);
				fButtonDown = false;
			}
		}
	}
	else
	{
		if (localRect.Contains(point))
		{
			if ((GetItemStyle() & kRadioButtonAllowNoneSetStyle) == 0)
				SetBooleanValue(true);
			else
			{
				SetBooleanValue(true);
				fButtonDown = true;
			}
		}
	}
	Invalidate();
	Update();
	return true;
}
```

```
bool PictureRadioButton::Track(Tracker& tracker)
{
     if (tracker.GetAction() != kButtonDownAction) return false;

     NRect localRect;
     NPoint point;

     GetLocalRect(localRect);
     tracker.GetPoint(point);

     if (localRect.Contains(point))
         if (GetItemStyle() & kRadioButtonAllowNoneSetStyle)
              SetBooleanValue(fButtonDown ^= true);
         else
             SetBooleanValue(true);
     Invalidate();
     Update();
     return true;
}
```

```
bool PictureRadioButton::Track(Tracker& tracker)
{
     if (tracker.GetAction() != kButtonDownAction) return false;

     NRect localRect;
     NPoint point;

     GetLocalRect(localRect);
     tracker.GetPoint(point);

     if (localRect.Contains(point))
          SetBooleanValue(!(GetItemStyle() & kRadioButtonAllowNoneSetStyle) ||
                              fButtonDown ^= true);
     Invalidate();
     Update();
     return true;
}
```

```
bool PictureRadioButton::Track(Tracker& tracker)
{
     if (tracker.GetAction() != kButtonDownAction) return false;

     NRect localRect;
     NPoint point;

     GetLocalRect(localRect);
     tracker.GetPoint(point);

      if (localRect.Contains(point))
         SetBooleanValue(!(GetItemStyle() & kRadioButtonAllowNoneSetStyle) ||
                         fButtonDown ^= true);
     Invalidate();
     Update();
     return true;
}
```

```
template <typename VisObj>
inline bool doesLocalRectContainPoint(VisObj& vob, Tracker& tracker)
{
    NRect localRect;
    NPoint point;

    vob.GetLocalRect(localRect);
    tracker.GetPoint(point);

    return localRect.Contains(point);
}
```

```
bool PictureRadioButton::Track(Tracker& tracker)
{
     if (tracker.GetAction() != kButtonDownAction) return false;

     if (doesLocalRectContainPoint(*this, tracker))
         SetBooleanValue(!(GetItemStyle() & kRadioButtonAllowNoneSetStyle) ||
                         fButtonDown ^= true);

     Invalidate();
     Update();
     return true;
}
```

- C, C++ and STL are tools built by professional programmers for professional programmers

- Their effective use presupposes knowledge of the core areas of Computer Science

Core of Computer Science

- Data Structures and algorithms
- Theory of computation
- Programming Languages and Compilers
- Operating systems
- Computer architecture

Common machine architecture

- Reasons
  - Ability to build diverse applications
  - Ease to understand, analyze and extend
  - Portability
- Features
  - Byte-addressable memory
  - Pointers
  - Stack-based function call

C machine

- C abstracts from instructions
- C++ abstracts from data types
- STL abstracts from data structures

They share the same fundamental machine model!

In order to understand C++, in order to understand STL, one needs to understand C machine

The way C handles pointers was a brilliant innovation; it solved a lot of problems that we had before in data structuring and made the programs look good afterwards.

Donald Knuth

Value semantics

- C has value semantics
  - If you need pointer semantics – use pointers
- C++ extends value semantics with copy constructors, assignment and destructors
- STL extends value semantics on data structures and generalizes pointer semantics to iterators

Regular types requirements

- T a = b; assert(a == b);
- a = b; assert(a == b);
- T a = b; T c = b; mutate(a); assert(b == c);
  - No sharing

Regular types advantages

- Pass to functions
- Return from functions
- Create temporaries on the stack
- Store in data structures
- Understandable to the compiler
  - Copy propagation
  - Common sub-expression elimination
- Understandable to a human
- EXTENSIBILTY

Sacred Cows

- Top-down design
- Object Orientation
- Design Patterns
- Template Metaprogramming

Learning from the greats

- Ken Thompson
  - Simple, abstract
    - *Lions' Commentary on UNIX 6th Edition*
    - *Linux is the best modern imitation*
- Donald Knuth
  - Methodical, meticulous
    - *TEX + Web*
- Bjarne Stroustrup
  - Persistent, evolutionary, pragmatic
    - *Design and Evolution of C++*
- Seymour Cray
  - Efficient, minimal
    - (Blaauw and Brooks, *Computer Architecture)*

Great Books

- Knuth, *The Art of Computer Programming*

  If you think that you are a good programmer … read Art of Computer Programming…

  Bill Gates

- Dijkstra, *Discipline of Programming*
- Abelson and Sussman, *Structure and Interpretation of Computer Programs*
- Hennessy & Patterson, *Computer Architecture*

Source code is the product

- Much more time reading than writing
- Code is the main communication channel
- Code is documentation
- Code is the asset
- Aesthetics of code

Software engineering

- Programs == Algorithms + Data Structures
- Good programmers
  - Know many
  - Use them properly
    - 80% - 20% rule
  - Occasionally (very seldom) invent new ones
- Professional standards
  - Educational
  - Quality
  - Professional responsibility

Group engineering

- Design
  - Ownership
    - Clear
    - Transferable
  - Reviewed
  - Responsible
- Code
  - Ownership
    - Clear
    - Transferable
  - Reviewed
  - Responsible

Software economics

- Code as liability
  - Depreciation
  - Maintenance
- Organizational tax on code
  - Lines
  - Changes across releases
  - Bugs
- Benefits
  - Reuse
  - Investing into design
  - Continuous improvement of code base

We are heirs to a glorious tradition:

Let us be proud of what we are