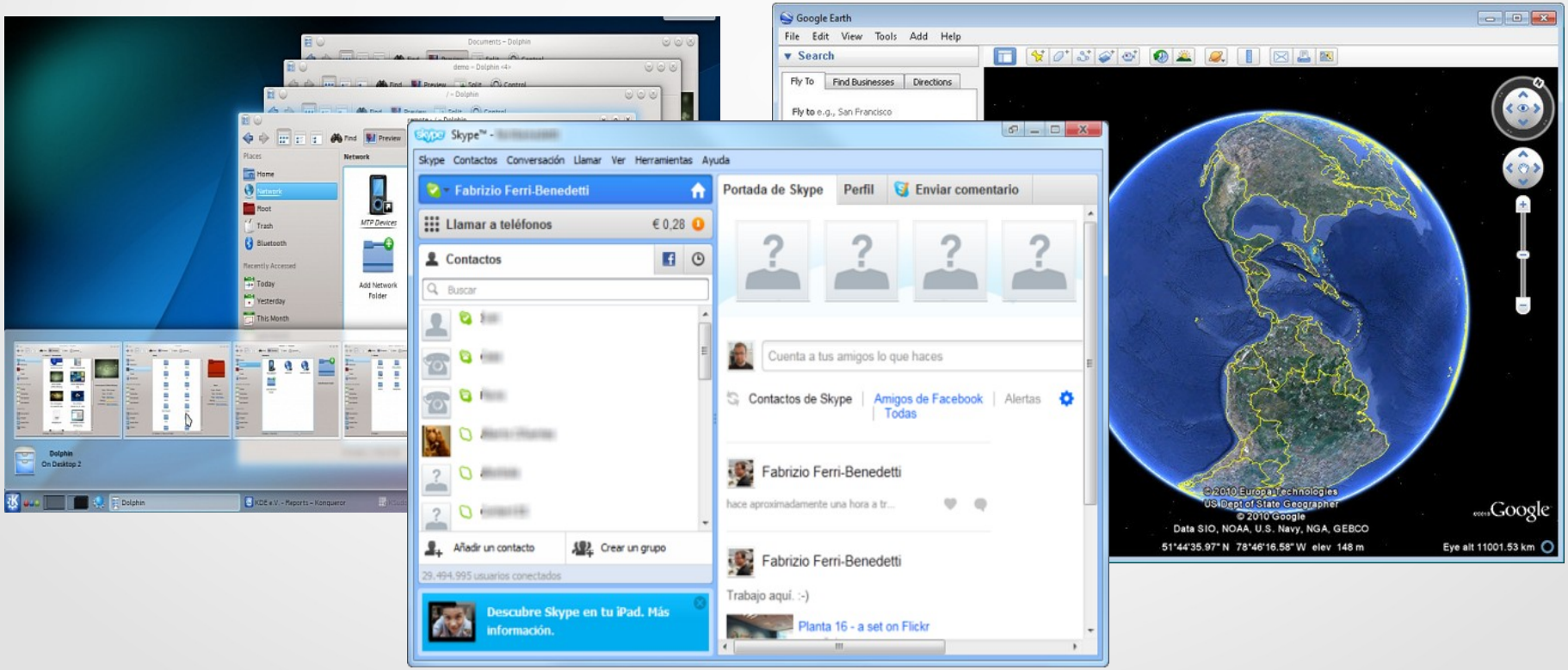
Code less.
Create more.
Deploy everywhere.

Qt

- Qt es un Framework para hacer interfaces multiplataformas con c++
- Qt es desarrollada como un software libre y de código abierto a través de Qt Project, donde participa tanto la comunidad, como desarrolladores de Nokia, Digia y otras empresas
- Qt es utilizada en KDE, entorno de escritorio para sistemas como GNU/Linux o FreeBSD, entre otros. Qt utiliza el lenguaje de programación C++ de forma nativa, adicionalmente puede ser utilizado en varios otros lenguajes de programación a través de bindings

Qt

- Qt es utilizado por muchas empresas para hacer productos que utilizamos a diario, entre los más importantes podemos nombrar el entorno de escritorio KDE, Skype y Google Earth.



Herramientas de desarrollo

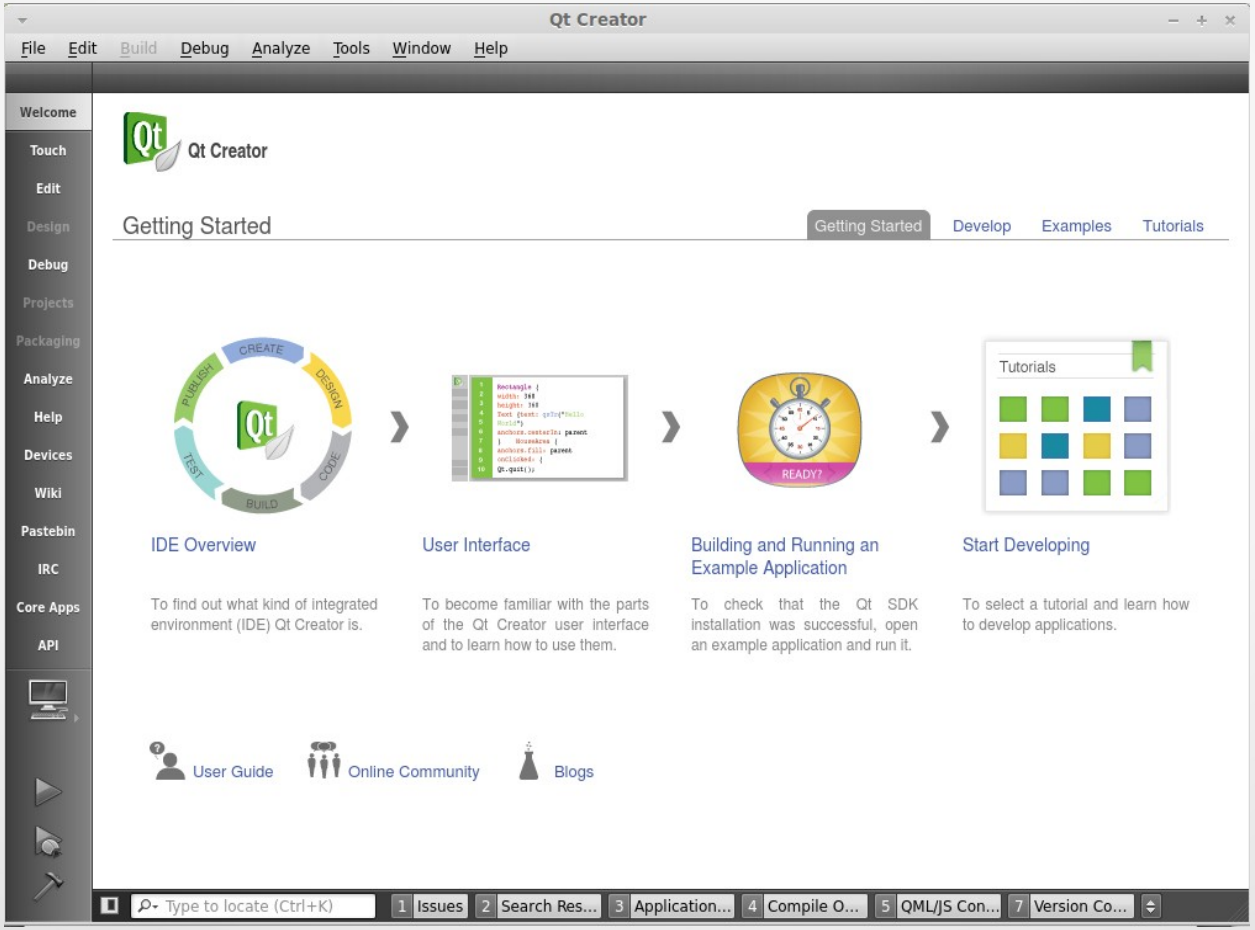
- Qt permite desarrollar con cualquier entorno de desarrollo, pero a la vez nos provee un potente entorno de desarrollo llamado Qt creator.
- A al vez podemos utilizar Qt designer para el diseño de nuestras interfaces.
- Qt decidió separar el diseño y el desarrollo para que el programador pueda compartir su trabajo con un diseñador.
- De igual forma Qt creator trae un diseñador de interfaces similar a Qt designer.

Qt creator


Projects
• qmake
• CMake
Debuggers
• gdb
• CDB
Editors
• Code editor
• UI designer
Qt
Tools
• Version control
• Qt Simulator
Languages
• C++
• QML
Targets
• Desktop
• Mobile

Qt creator

- Qt creator es un entorno de desarrollo potente que permite desarrollar interfaces qt de forma interactiva



Macros

- Qt utiliza macros para extender el lenguaje c++ y poder facilitarnos el desarrollo
- Por medio de palabras claves que debemos utilizar. El compilador en la etapa de preprocesamiento utiliza estas palabras claves para introducir código y transforma nuestro código en C++ estándar, aumentando nuestra productividad y ahorrándonos escribir código repetitivo.

Primera aplicación

- Vamos a empezar con un programa de Qt muy simple y lo estudiaremos línea por línea:

```
#include <QApplication>
#include <QLabel>


int main(int argc, char *argv[])
{
QApplication app(argc, argv);
QLabel *label = new QLabel("Hello Qt!");
label->show();
return app.exec();
}
```

Primera aplicación

```
#include <QApplication>

#include <QLabel>
```

- En la linea 1 y 2 incluimos las clases QAplication y Qlabel. En QT todas las clases comienzan con la letra Q en mayúscula.

```
QApplication app(argc, argv);
```

- En la linea 5 creamos la aplicación qt y le pasamos como parámetro, los parámetros de main dado que la aplicación se puede ejecutar desde la linea de comandos con algún parámetro.

Primera aplicación

QLabel *label = new QLabel("Hello Qt!");

- En la linea 6 se crea el widget QLabel el cual muestra “Hello Qt!”, los **widget** en la terminología Qt y unix son elementos visuales. Un botón, menú, scroll son ejemplos de widget. Además un widget puede contener a otro widget.

label->show();

- En la linea 7 se hace visible el componente, dado que todos los componentes nacen invisibles.

Primera aplicación

return app.exec();

- Línea 8 pasa el control de la aplicación a Qt. En este punto, el programa entra en el bucle de eventos. Esta es una especie de modo de espera, donde el programa espera de las acciones del usuario tales como clics del ratón y las pulsaciones de teclas. Las acciones del usuario generan eventos (también llamados "mensajes") a los que el programa puede responder, por lo general se ejecutan una o más funciones.

Un poco más avanzado...

- Vamos a agregar un botón para salir de la aplicación:

```
#include <QApplication>
#include <QPushButton>

int main(int argc, char *argv[])
{
QApplication app(argc, argv);
QPushButton *button = new QPushButton("Quit");
QObject::connect(button, SIGNAL(clicked()), &app, SLOT(quit()));
button->show();
return app.exec();
}
```

Signals y Slots

QObject::connect(button, SIGNAL(clicked()), &app, SLOT(quit()));

- En este ejemplo agregamos un botón el cual emite la señal clicked cuando el usuario hace click en el componente.
- Los widgets de Qt emiten señales para indicar que se ha producido una acción del usuario o de un cambio de estado. Por ejemplo, QPushButton emite una señal clicked() cuando el usuario hace clic en el botón. Una señal se puede conectar a una función (utilizando la función SLOT), de modo que cuando se emite la señal, la función se ejecuta automáticamente.
- Tanto la función SIGNAL como SLOT son macros que nos ahorran de escribir código repetitivo.

Segunda aplicación

- Vamos a hacer una segunda aplicación que contenga 2 componentes y que nos permita utilizar en más profundidad las señales.
- El objetivo de la aplicación es que el Slider y la caja de texto, muestren siempre el mismo valor.

Segunda aplicación

- La aplicación consiste en 3 widget un QSpinBox, un QSlider y un QWidget. QWidget va a ser la ventana de la aplicación. Como es de esperar QSpinBox y QSlider se van a dibujar dentro de QWidget. Por lo tanto podemos afirmar que QWidget es contenedor de QSpinBox y QSlider. Veamos el código:

```
#include <QApplication>
#include <QHBoxLayout>
#include <QSlider>
#include <QSpinBox>


int main(int argc, char *argv[])
{
  QApplication app(argc, argv);
  QWidget *window = new QWidget;
  window->setWindowTitle("Enter Your Age");
  QSpinBox *spinBox = new QSpinBox;
  QSlider *slider = new QSlider(Qt::Horizontal);
  spinBox->setRange(0, 130);
  slider->setRange(0, 130);
```

Segunda aplicación

```
  QObject::connect(spinBox, SIGNAL(valueChanged(int)), slider, SLOT(setValue(int)));
  QObject::connect(slider, SIGNAL(valueChanged(int)), spinBox, SLOT(setValue(int)));

  spinBox->setValue(35);

  QHBoxLayout *layout = new QHBoxLayout;
  layout->addWidget(spinBox);
  layout->addWidget(slider);
  window->setLayout(layout);

  window->show();
  return app.exec();
}
```

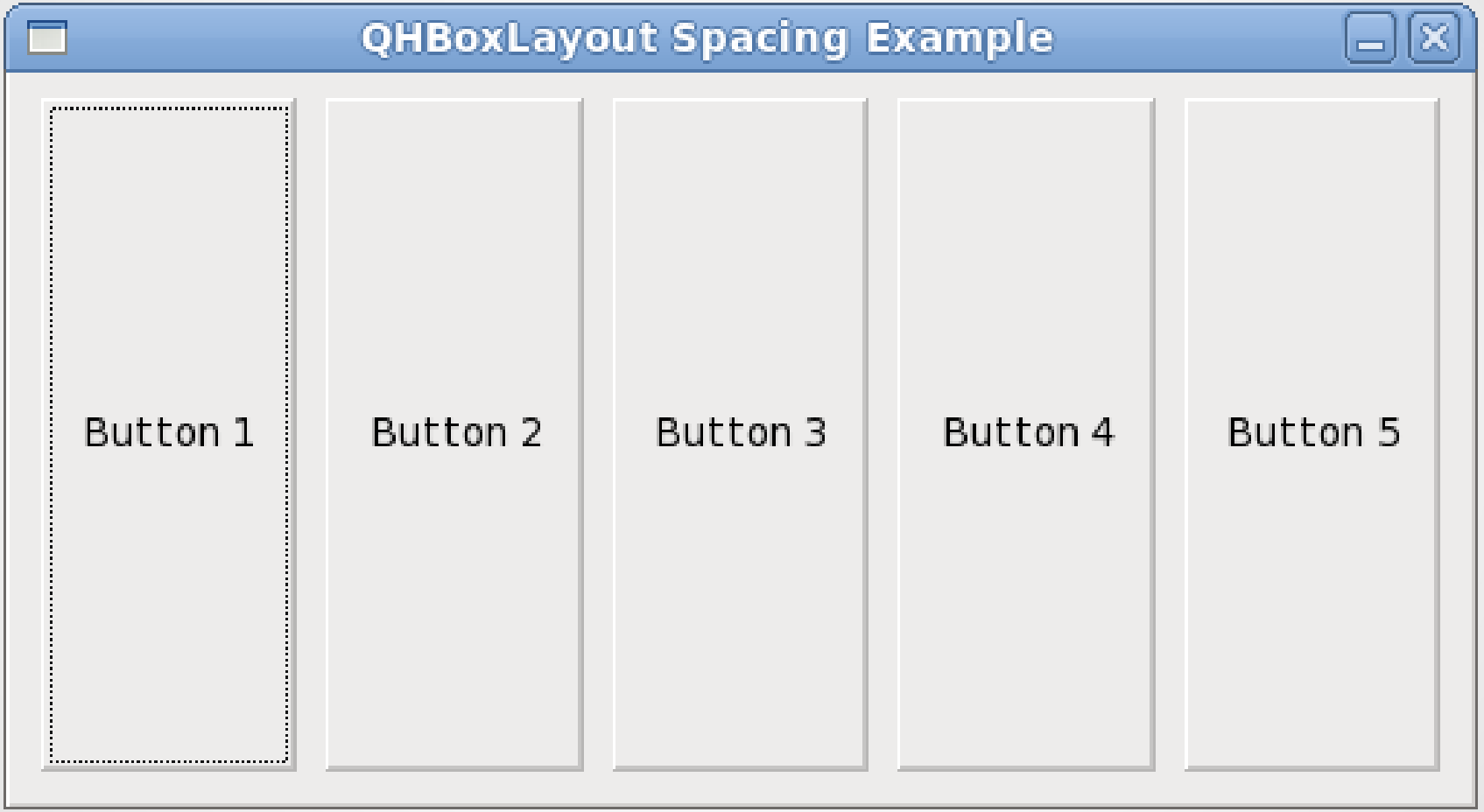
Segunda aplicación

- Luego de crear la ventana se establecer el texto de la barra de titulo.
- Después creamos los dos componentes QSpinBox y QSpinBox y configuramos el rango de 0 a 130. Además indicamos a Qt que si spinBox lanza la señal valueChanged (que indica que el valor del componente ha cambiado) ejecute la función setValue de slider.
- QHBoxLayout es un layout, los layouts nos permiten configurar fácilmente la disposición de los componentes en una ventana.
- QHBoxLayout es un layout que organiza los componentes de forma horizontal. Agregamos QSpinBox y QspinBox al layout y luego agregamos el layout a la ventana para que se muestre.

Layouts

- Los layouts, son las dispociciones de los componentes en las ventanas
- Qt nos brinda 3 grandes tipos de layouts:
  - QHBoxLayout que permite organizar los componentes de forma horizontal y de izquierda a derecha.
  - QVBoxLayout que permite organizar los componentes de forma vertical y de arriba a abajo.
  - QGridLayout organiza los componentes como una grilla.
- Con el método setLayout indicamos el layout a la ventana. Es posible indicar un árbol de componentes y de esta forma una ventana puede tener diferentes layouts.

QHBoxLayout



QVBoxLayout y QGridLayout