ISE TN01 v1.2
Setting-up the INGENIA Development Environment

Ricardo Sanz [*] [†] **and Javier Antn** [‡]

[*]UPM Autonomous Systems Laboratory, [†]UPM-SIC Centro de Automtica y Robtica, and [‡]UPM ETSII Industrial Engineering Master

Technical note TN01 v1.2 — INGENIA Systems Engineering Course

This technical note succinctly describes the process to set-up a development environment for C++ development. It describes what is needed to use a basic toolchain for C++ development and also what is needed for developing using an integrated development environment.

Keywords: programming | C++ | toolchain | CygWin | IDE | Eclipse

INGENIA Systems Engineering (ISE) focuses on systems engineering of software-intensive autonomous systems. This requires from the student teams the capability of programming complex applications.

This technical note (TN01) gives instructions on how to set up a simple, personal *development environment* to write C++ programs.

The initial sections provide some general knowledge about the issues at hand: the needed tools, the alternatives, the chosen tools, *etc.* The final section describes step by step the process to set-up the programming environment using i) the bareback GCC toolchain and ii) the Eclipse CDT IDE.

Contents

Introduction

In ISE we develop an intelligent system where intelligence comes from the execution of sophisticated software for perception, decision-making and control of a physical system. The ISE software systems depart from common desktop applications in several aspects:

- The software incorporates artificial intelligence technologies.
- The software is distributed, *i.e.* is a collection of coupled programs running on a set of connected computers.
- The software is embedded, *i.e.* runs on (usually small) computers that are parts of larger systems.
- The software is real-time, *i.e.* it has requirements concerning the time taken to do some specific computations.

These last issues —esp. the embedded real-time character— pose specific requirements for the programming languages to be used. The most natural election —following common trends in industry— is to use C/C++ as development languages for these systems.

C++ is a modern, sophisticated programming language that provides resources for imperative programming, object-oriented programming and generic programming. It is a language widely used and supported as inheritor of the C language tradition in systems programming.

This note describes what is needed to use a basic toolchain for C++ development and also what is needed for developing using an integrated development environment (IDE).

While there are many C/C++ toolchains and many IDEs available, we will specifically focus on:

- The GNU C/C++ compiler system.
- The Eclipse CDT development environment.

These two are the most used and most widely available tools, hence the effort dedicated to learn them will not be wasted because it could be leveraged pervasively.

The C and C++ Languages

C (Kernighan and Ritchie, 1988; ISO/IEC, 2011) and C++ (Stroustrup, 2014; ISO/IEC, 2014) are the programming languages most used in the development of embedded systems. While there are some concerns about the use of C++ for real-time systems, the capability that it provides for both the small-scale of hardware-related issues and the large-scale of complex modular applications make it the language of election for this class of systems.

While C and C++ are two different languages, C++ can be considered a *de facto* superset of C. Differences between C

About ISE

INGENIA Systems Engineering is a transversal competences course focused on the application of systems engineering to the construction of an embedded intelligent system with augmented autonomy. Autonomy is a technical property of systems that make them capable of sustained provision of a particular service even in the presence of major uncertainties in the service demand, the context of service execution and the disturbances that the system may be suffering from outside or inside.

and the equivalent subset of C++ are minimal (See for example section 44.3 of the Stroustrup book (Stroustrup, 2014)). This implies that a programmer using a C++ environment can in fact produce ISO-C compliant code. Standard-compliant code is code that strictly follows a standard. This is of enormous importance to guarantee stability, quality and, especially, portability.

So said, in what follows we will refer to C++ and not to C (unless specifically noted). Updated information on the evolution of the C++ standard can be found at `https://isocpp.org/std/the-standard`.

C++ Toolchains

To write software in C++ we need a C++ toolchain. A toolchain is a set of programming tools that are used to create a software product (typically an executable program). The tools in the tool set will depend on the type of programming languages and execution environments for the final programs. C++ is a compiled language, hence the minimal toolchain is a C++ compiler capable of generating executable programs.

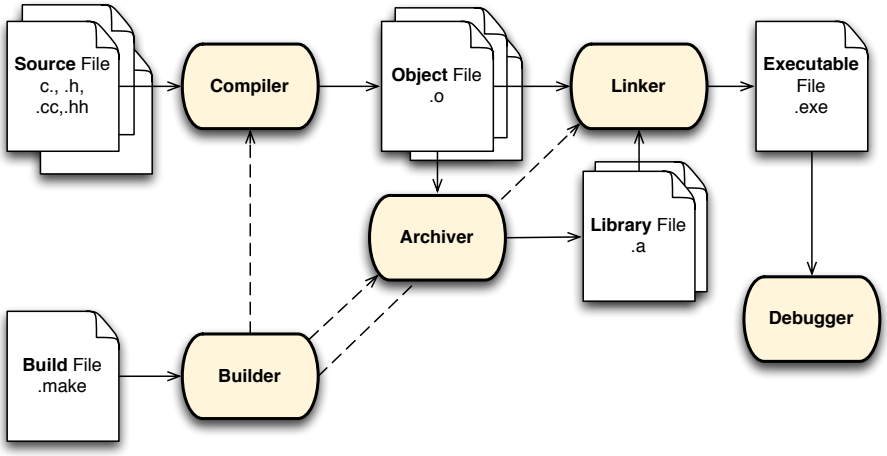


Figure 1 – A common workflow in the world of C++ programming. The collection of chained tools —the curved boxes— is usually called *toolchain*.

There are many C++ compilers that are steadily maintained to keep track of recent developments in the C++ language. Some of them are the following: GCC, Clang, MSVC, EDG eccp, Intel C++, IBM XLC++, Sun/Oracle C++, Embarcadero C++ Builder, Cray, Portland Group (PGI) HP aCC, and Digital Mars C++.

The most known and used are GCC[1] (the GNU C/C++ Compiler), Clang[2] (the C++ frontend to the LLVM compiler infrastructure) and MSVC[3] (Microsoft Visual C++ Compiler), . MSVC is specifically targeted towards MS Windows applications, while GCC and Clang are cross-platform compilers. GCC is the historically most relevant compiler, while Clang is a recent incorporation intended to fully replace GCC. Both GCC and Clang would be adequate for the ISE purposes. We will use GCC in this technical note as is it the most used toolchain in the embedded systems domain.

The GNU Compiler Collection (GCC) is a compiler system produced by the GNU Project supporting various programming languages. GCC has been ported to produce executable code for a wide variety of computer architectures, being used in the development of both free and proprietary software. The most important aspect for us is that GCC is available for most embedded systems both as a native or cross-compiler.

Some of the tools that are included in the GNU toolchain are:

- GNU Compiler Collection (GCC[4]): a suite of compilers for several programming languages (we are only interested in C/C++ compilers `gcc` and `g++`).
- GNU `make`: an automation tool for compilation and build.
- GNU Binutils: a suite of tools including the `ld` linker, the `as` assembler and other tools related to binary files[5].
- GNU `bison`: a parser generator, often used with the `flex` lexical analyser.
- GNU `m4`: a macro pre-processor.
- GNU `gdb`: a code debugging tool.
- GNU auto-configurable build system (autotools): `autoconf`, `automake` and `libtool`.

[1] `https://gcc.gnu.org/`

[2] `https://clang.llvm.org/`

[3] `https://docs.microsoft.com/en-us/cpp/`

[4] The original meaning of the acronym "GCC" was "GNU C Compiler" because the native compiler in original UNIX systems was simply called 'cc' — *i.e.* C Compiler.

[5] The term "binary file" is used to refer to files not readable by humans and intended to be used by the computer or by programs. Paradigmatic examples are images, object files generated by compilers and executable files generated by linkers.

```
sanz@UBallard:~/code$ gcc -version
gcc: error: unrecognized command line option '-version'
gcc: fatal error: no input files
compilation terminated.
sanz@UBallard:~/code$ gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.5) 5.4.0 20160609
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

sanz@UBallard:~/code$
```

Figure 2 – Using the gcc compiler in an Ubuntu system.

From all these tools, we will only explicitly use the C++ compilation driver (`g++`) the debugger (`gdb`) and the build tool (`make`). Other tools will be implicitly used (for example the m4 preprocessor or the linker).

Installing a C++ toolchain

In Linux systems. This is a natural environment for the GCC toolchain; just use the software installation commands of your Linux distribution. For example, in Ubuntu just write in a console the command '`sudo apt-get install g++`".

In OSX systems. Apple OSX is the natural environment of Clang/LLVM; so in these systems it is natural to use Clang more than GCC. Compilers are installed as part of the large XTools install but can also be installed alone as a separate package (look for 'Command Line Tools', available as a separate download from the Apple developers website.)

In Windows systems. There are several ways of having a GCC compiler in a Windows box. The most used alternatives are CygWin and MinGW. MinGW, a contraction of "Minimalist GNU for Windows", is simpler and provides a minimalist development environment for native Microsoft Windows applications. However, I recommend CygWin that provides a complete UNIX-like environment that supports the complete GNU toolchain.

In all these cases, the program is written with any editor and the compilation is done giving compilation commands in a console (see Figure 2).

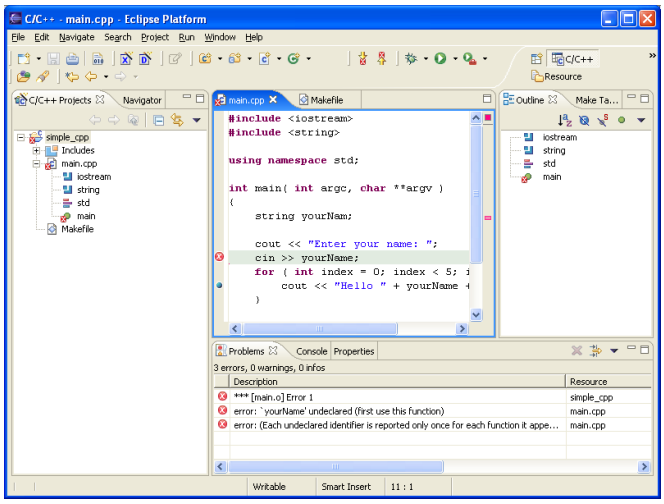


Figure 3 – The Eclipse CDT workbench.

In many cases the host and the target of a specific development project coincide (*e.g.* programming in a Linux computer to execute in the same or another similar Linux computer). In other cases host and target do not coincide and it is necessary to install a cross-compilation toolchain.

The Eclipse C/C++ Development Toolkit

The use of command line tools is extremely effective but cumbersome for large scale code development that needs multiple tools. This is especially important concerning code management and visualization. The common strategy is to integrate multiple tools in a graphical environment that provides easy access to the many tools and visual rendering of many code features. These variants of bareback programming toolchains are known as Integrated Development Environments (IDE).

There are many IDEs out there. The best known ones are Microsoft Visual Studio in Windows systems, Apple XCode in OSX boxes and Eclipse in all classes of environments (Windows, OSX and Linux). This wider availability makes Eclipse the best IDE for systems engineers (that are never tied to particular technologies and systems).

Eclipse is in fact not an IDE but an empty platform for IDE construction. It can be extended to support many classes of software development projects. In the C and C++ domains, Eclipse is enlarged into an IDE by means of the installation of the C/C++ Development Toolkit (CDT). The CDT is a collection of Eclipse-based features/plugins that provides the capability to create, edit, navigate, build, and debug projects that use C and/or C++ as a programming language (See Figure 3).

However, the CDT just extends the Eclipse user frontend environment and it does does not include the necessary backend compilers. Compilers —and associated tools like linkers, profilers or debuggers— are used to convert C/C++ code into binary executable programs and to inspect, assess or debug those programs. The CDT provides the frameworks that allow such backend toolchain elements to be integrated in a consistent fashion hence providing an effective C/C++ IDE.

The Eclipse versatility is always there to allow you to mix and match such or other tools depending on your project requirements.

Sometimes, specific distributions of the Eclipse/CDT —e.g. some commercial distributions— do include the necessary toolchain and the frontend integrations. If yours does not —as is the case of plain vanilla Eclipse—, the base CDT does provide support for integration with some common toolchains —e.g. the GNU tools— for code build and debug. All this means that if you want to use Eclipse/CDT you must previously install a C++ toolchain like GCC.

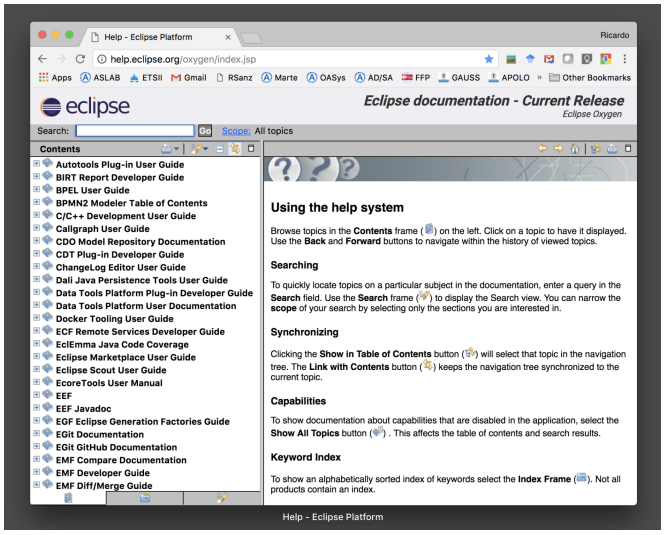


Figure 4 – The Eclipse online help system contains help for the available plugins of a specific release. It can be found at `http://help.eclipse.org`. The figure shows the help content for the Eclipse Oxygen release. The C/C++ Development User Guide item on the left provides help on CDT use.

Step by Step Installation of a C++ Environment

This section describes the process to install a programming environment for C++ in Windows. This environment will be based on the GCC toolchain. The real toolchain will be provided by CygWin and we will add the Eclipse CDT to have a graphical frontend to it (*i.e.* an IDE).

1. Install the C++ toolchain:
   (a) Download **CygWin Setup** tool[6] (`setup-x86.exe` or `setup-x86_64.exe` for 32 and 64 bit Windows).
   (b) Execute CygWin setup tool and install the latest GNU C++ package (`gcc-g++`). Other tools in the toolchain requiere the installation of other packages (*e.g.* to use the `make` tool we must install the `make` package.
2. Install Eclipse CDT:
   (a) Download and install a **Java runtime**. Note that Eclipse is a Java application. Having a Java runtime environment[7] in your Windows system is necessary for running Eclipse.
   (b) Download Eclipse with the CDT plugin pre-installed from the `http://eclipse.org` website[8]. Oxygen is the codename for the last Eclipse version. This package is called **Eclipse IDE for C/C++ Developers**. There are download links for Windows 32-bit, Windows 64-bit, Mac OS X, Linux 32-bit and Linux 64-bit.
   (c) Uncompress the downloaded Eclipse package and execute Eclipse.exe to configure it to use the CygWin GCC toolchain. As with most Java applications, you can decompress and place the Eclipse folder with the executable program wherever you like.

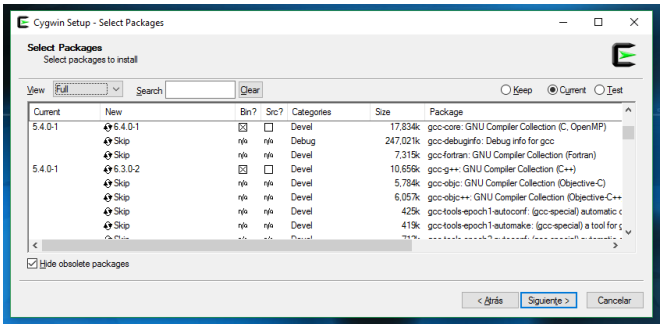


Figure 5 – The CygWin setup package selection tool.

To experiment with C++ programming only Step 1 is necessary (Install C++ toolchain). This will install just the CygWin environment that is used in the same way as a Linux terminal (*i.e.* using GCC command line tools for compilation). In this case you must find an editor to write the programs (because you are not installing the IDE that provides it). In CygWin you may install UNIX-like editors (*e.g.* `vim`) but you can also use a native Windows application like `Notepad` or `Wordpad` or else install in Windows a programmer-friendly editor like `Notepad++`, `Sublime Text` or `Visual Studio Code` (this last is a good election).

Summary of needed software. In summary, for a full development environment it is necessary to install the following software:

- CygWin with the gcc-g++ package.
- Java runtime for Eclipse execution.
- Eclipse with the CDT plugin preinstalled.

Details on Installing CygWin

As was said before, to experiment with C++ programming only a C++ toolchain -and an editor- is needed. We will use the CygWin GCC toolchain. To install CygWin you must i) download and execute the installer program, ii) give some installation parameters, iii) choose packages to install and iv) do the installation.

The most important decisions are i) select the right installer (32 or 64 bit), ii) choose a good download site (I always use ftp://mirror.switch.ch because I'm quite old and it has been there since I was a student), and iii) choose the packages to install. The installer can be executed again when needed to install, update or remove packages).

The installer package selection list has several columns (see Figure 5):

Current: The currently installed version (this makes sense when updating CygWin).

[6] `https://cygwin.com/install.html`
[7] `https://java.com/en/download/`
[8] `http://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/oxygen1`

New: What the installer will do (*e.g.* skip, keep, reinstall, uninstall, or install a concrete version of the package).

Bin? Install the binary packages (*i.e.* the executable programs). This is what we need.

Src? Install the source packages (*i.e.* the original source programs written by the package programmers). This is needed by people that do special things with the software. We don't need it.

Categories: The packages are organised in categories to help manage them. Ours are in the "Devel" category.

Size: The amount of disk storage that the installation will use. This is important for small systems-

Package: The name of the package to be installed. For console use we only need the C++ compiler that is installed with the "gcc-g++" package. To quickly find this package write "g++" in the CygWin setup program search box.

In most packages it says "skip" (that means they will not be installed). Clicking on the icon of the double circular arrow -in the "New" column- the action will change (install a version, update to another version, uninstall *etc.*). Sometimes choosing a package or updating one will automatically select other packages (*e.g.* selecting gcc-g++ will automatically select gcc-core).

After selecting the packages you can proceed to install/upgrade.

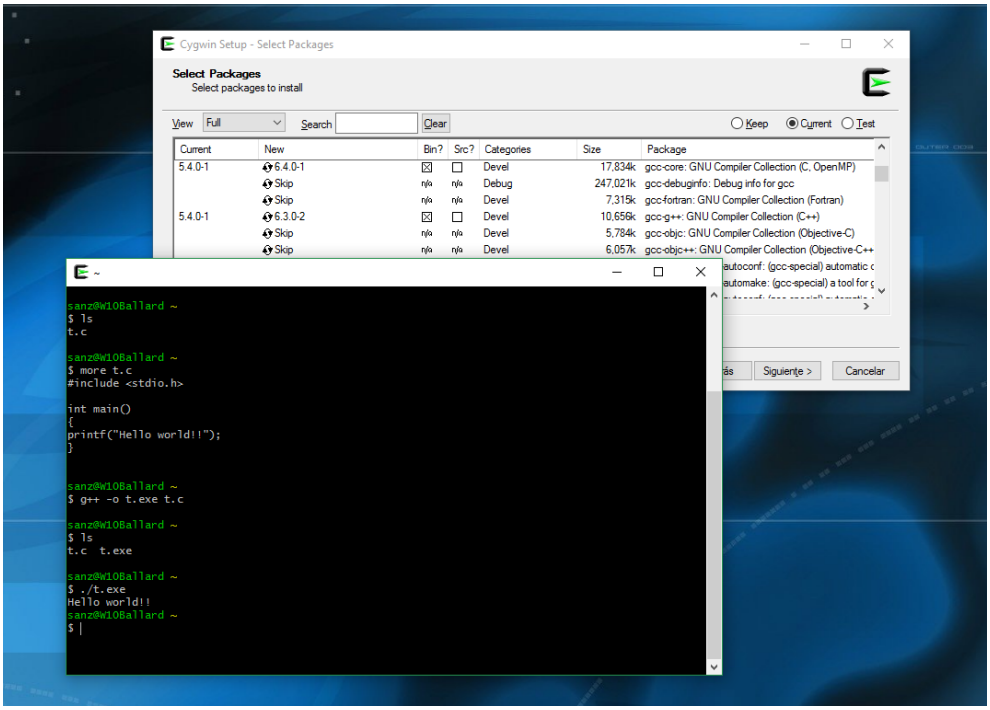


Figure 6 – Compiling and executing a program in the CygWin environment. The CygWin setup package selection tool is also seen.

After this you will have a CygWin installation that you can use runningn the `cygwin` program —a console where you can work like in a UNIX system (see Figure 6) .

Setup Cygwin toolchain in Eclipse CDT. If you want to use the CygWin toolchain as a backend for the Eclipse CDT IDE, it is necessary to do some extra configuration steps to link Eclipse with the CygWin toolchain.

First, to easy Eclipse configuration, you must install first CygWin and after this is installed and working, install Eclipse+CDT.

For a quite complete Cygwin toolchain installation that can support most Eclipse CDT features these are the suggested CygWin packages to be installed:

- binutils
- gcc
- gcc-core
- gcc-g++
- gcc-mingw-core
- gcc-mingw-g++
- make
- gdb

Add %cygwin%\bin to your Windows PATH, so when you open eclipse the Cygwin programs will be available to it. The Cygwin toolchain will be shown when you create a new C/C++ project using the wizard.

To use the `gdb` debugger it may be necessary to establish some "drive mappings" to give access to source code for debug. This is done in the Eclipse Preferences configuration panel.

In Eclipse, go to Windows → Preferences → C/C++→ Debug → Common Source Lookup. Add the following 'Path mappings':
\cygdrive\c → c:\
\cygdrive\d → d:\
plus any other drives if any ...

Eclipse modeling tools

SysML modeling

eclipse.org/modeling/mdt

Setting the engineering environment for SysML modeling

Extending the tool support along the lifecycle requieres other tools for concrete activities.

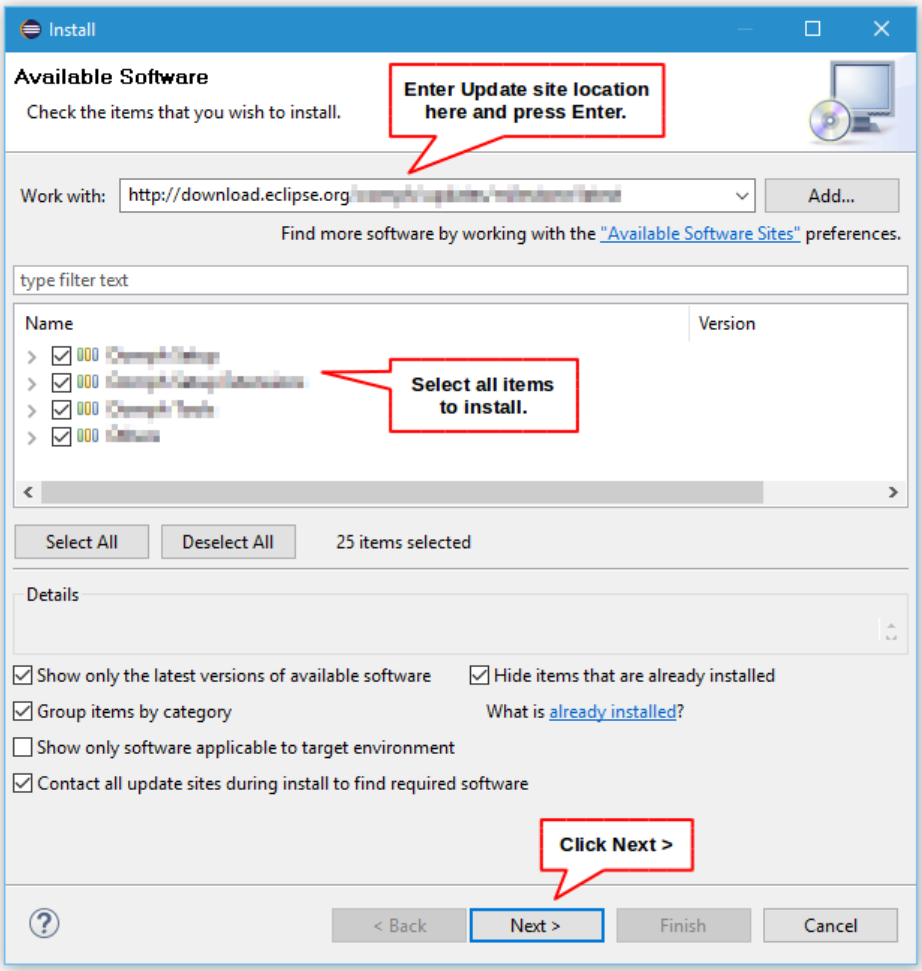


Figure 7 – Installing new plugins in Eclipse. Go to **Help** → **Install New Software**.

http://download.eclipse.org/modeling/mdt/papyrus/components/sysml12/oxygen/

Glossary

C is a programming language defined in (Kernighan and Ritchie, 1988). 1

C++ is a programming language defined in (Stroustrup, 2014). 1, 2

features is a form of packaging pluggable extension of the Eclipse platform. 3

generic programming is a form of programming where generic program snippets can be adapted to manipulate custom data types. 1

host is the computer system where a particular software is developed. See target. 3

IDE is an Integrated Development Environment, *i.e.* a set of integrated tools usually employed in the production of software (*e.g.* an editor, a debugger, a compiler, *etc.*). 1, 3

imperative programming is a form of programming where program instructions are orders given to perform a specific action. 1

object-oriented programming is a form of programming where code and that data it manipulates are conjointly encapsulated in program modules usually called objects. 1

systems programming is the development of programs that are part or deeply linked to operating systems, in opposition to application programming, where the programs serve some purpose external to the computer system. 1

target is the computer system where a particular software will be executed. See host. 3

toolchain is the collection of elementary tools used to transform source code into target executables. 1, 2, 4

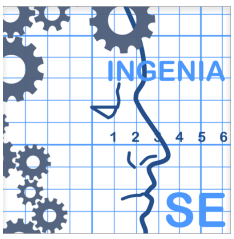
References

ISO/IEC (2011). ISO/IEC 9899:2011 - information technology - programming languages - C. International Standards.

ISO/IEC (2014). ISO/IEC 14882:2014 – programming languages - C++. International Standards.

Kernighan, B. and Ritchie, D. M. (1988). *The C Programming Language*. Addison-Wesley, Reading, MA, second edition.

Stroustrup, B. (2014). *The C++ Programming Language - 4th Edition*. Addison-Wesley, Reading, MA, fourth edition.



The ISE logo highlights the abstract nature of systems thinking. Abstraction is the key to good programming.