IMPLEMENTATION ISSUES

- Software engineering includes all of the activities involved in software development from the initial requirements of the system through to maintenance and management of the deployed system.
- A critical stage of this process is system implementation, where you create an executable version of the software.
- Implementation may involve developing programs in high- or low-level programming languages or tailoring and adapting generic, off-the-shelf systems to meet the specific requirements of an organization.

- 3 aspects of implementation that are particularly important to software engineering:
  1. **Reuse** → Most modern software is constructed by reusing existing components or systems. When you are developing software, you should make as much use as possible of existing code.
  2. **Configuration management** → During the development process, many different versions of each software component are created. If you don't keep track of these versions in a configuration management system, you are liable to include the wrong versions of these components in your system.
  3. **Host-target development** → Production software does not usually execute on the same computer as the software development environment. Rather, you develop it on one computer (the host system) and execute it on a separate computer (the target system). The host and target systems are sometimes of the same type, but often they are completely different.

Reuse:

- Software reuse is possible at a number of different levels:
  1. **The abstraction level** → At this level, you don't reuse software directly but rather use knowledge of successful abstractions in the design of your software. Design patterns and architectural patterns are ways of representing abstract knowledge for reuse.
  2. **The object level** → At this level, you directly reuse objects from a library rather than writing the code yourself. To implement this type of reuse, you have to find appropriate libraries and discover if the objects and methods offer the functionality that you need.

3. **The component level** → Components are collections of objects and object classes that operate together to provide related functions and services. You often have to adapt and extend the component by adding some code of your own. An example of component-level reuse is where you build your user interface using a framework.
4. **The system level** → At this level, you reuse entire application systems. This function usually involves some kind of configuration of these systems. This may be done by adding and modifying code (if you are reusing a software product line) or by using the system's own configuration interface.

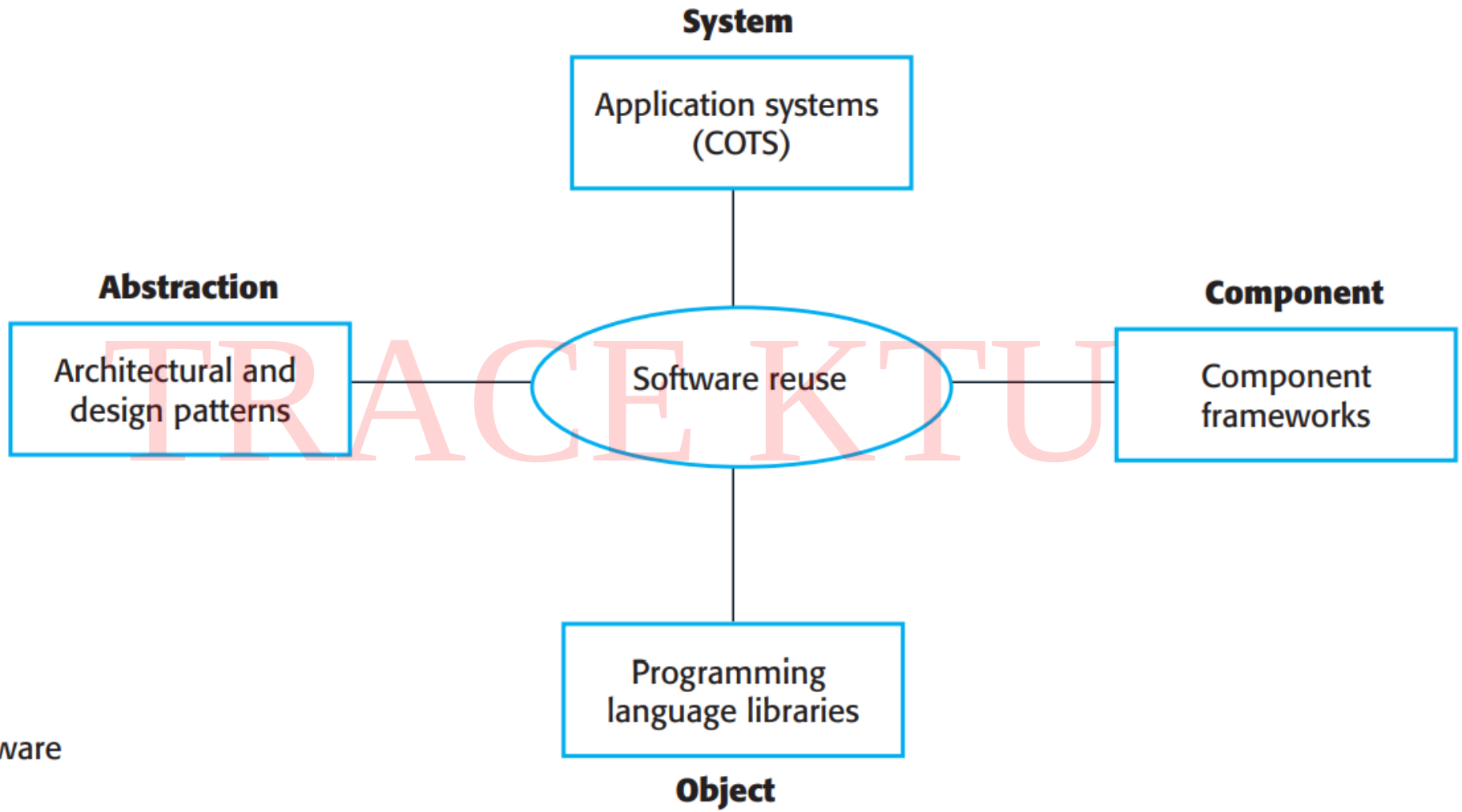

Figure 7.13 Software reuse

- By reusing existing software, you can develop new systems more quickly, with fewer development risks and at lower cost.
- The costs that are associated with reuse:
    1. The costs of the time spent in looking for software to reuse and assessing whether or not it meets your needs.
    2. Where applicable, the costs of buying the reusable software.
    3. The costs of adapting and configuring the reusable software components or systems to reflect the requirements of the system that you are developing.
    4. The costs of integrating reusable software elements with each other and with the new code that you have developed.

Configuration Management:

- Configuration management is the name given to the general process of managing a changing software system.
- The aim of configuration management is to support the system integration process so that all developers can access the project code and documents in a controlled way, find out what changes have been made, and compile and link components to create a system.

- 4 fundamental configuration management activities:
  1. **Version management** → where support is provided to keep track of the different versions of software components. Version management systems include facilities to coordinate development by several programmers. They stop one developer from overwriting code that has been submitted to the system by someone else
  2. **System integration** → where support is provided to help developers define what versions of components are used to create each version of a system. This description is then used to build a system automatically by compiling and linking the required components.
  3. **Problem tracking** → where support is provided to allow users to report bugs and other problems, and to allow all developers to see who is working on these problems and when they are fixed.
  4. **Release management** → where new versions of a software system are released to customers. Release management is concerned with planning the functionality of new releases and organizing the software for distribution.

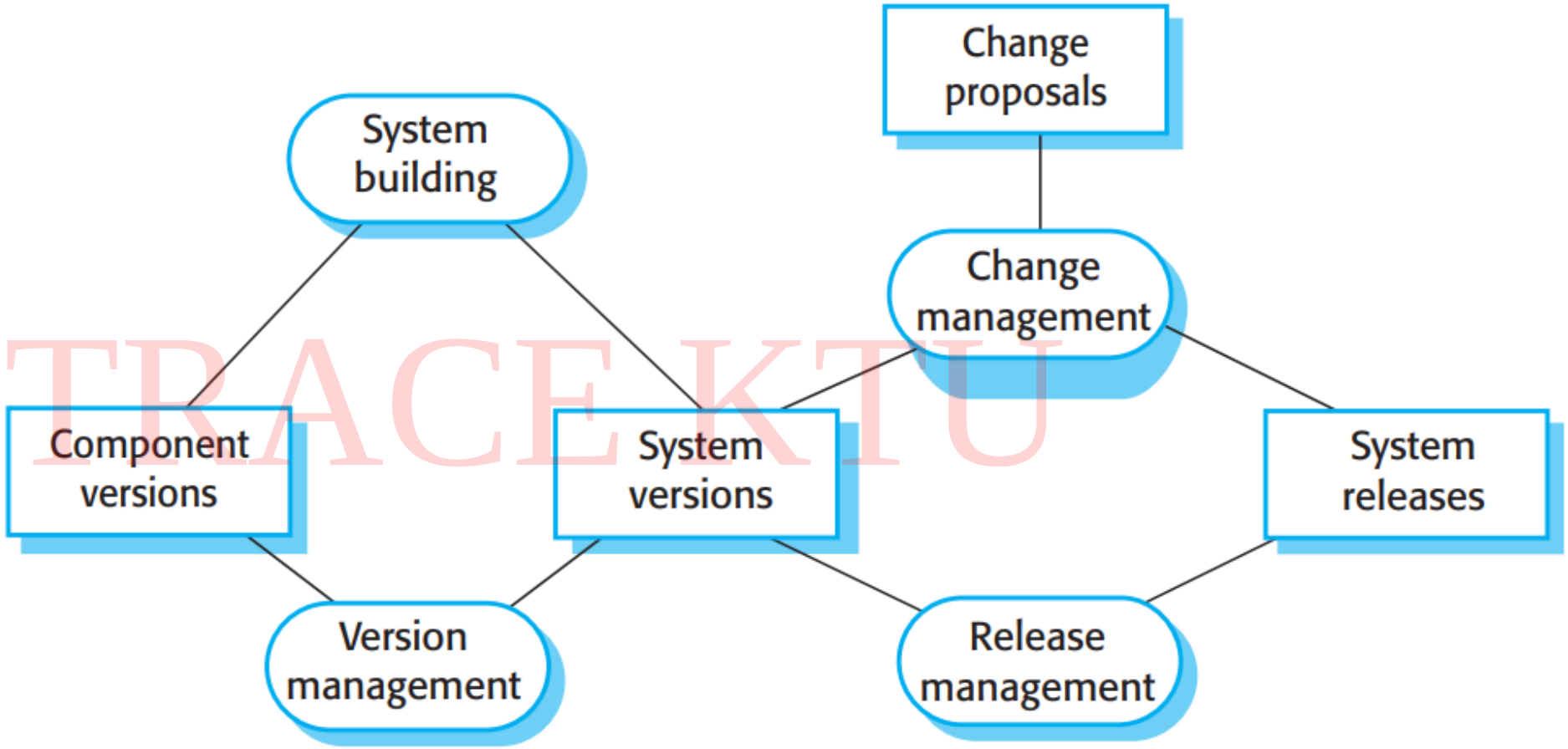

Figure 7.14 Configuration management

Host-target Development:

- Most professional software development is based on a host-target model.
- Software is developed on one computer (the host) but runs on a separate machine (the target).
- A platform includes the installed operating system plus other supporting software such as a database management system or, for development platforms, an interactive development environment.
- Simulators are often used when developing embedded systems.

- Simulators speed up the development process for embedded systems as each developer can have his or her own execution platform with no need to download the software to the target hardware.
- A software development platform should provide a range of tools to support software engineering processes. These may include:
    1. An integrated compiler and syntax-directed editing system that allows you to create, edit, and compile code.
    2. A language debugging system
    3. Graphical editing tools, such as tools to edit UML models.
    4. Testing tools, such as JUnit, that can automatically run a set of tests on a new version of a program.
    5. Tools to support refactoring and program visualization.
    6. Configuration management tools to manage source code versions and to integrate and build systems.

- In addition to these standard tools, your development system may include more specialized tools such as static analyzers.
- Software development tools are now usually installed within an integrated development environment (IDE).
- An IDE is a set of software tools that supports different aspects of software development within some common framework and user interface.
- A general-purpose IDE is a framework for hosting software tools that provides data management facilities for the software being developed and integration mechanisms that allow tools to work together. The best-known general-purpose IDE is the Eclipse environment.

- As part of the development process, you need to make decisions about how the developed software will be deployed on the target platform.
- Issues that you have to consider in making this decision are:
  1. The hardware and software requirements of a component.
     - If a component is designed for a specific hardware architecture, or relies on some other software system, it must obviously be deployed on a platform that provides the required hardware and software support.
  2. The availability requirements of the system.
     - High-availability systems may require components to be deployed on more than one platform. This means that, in the event of platform failure, an alternative implementation of the component is available.
  3. Component communications.
     - If there is a lot of intercomponent communication, it is usually best to deploy them on the same platform or on platforms that are physically close to one another. This reduces communications latency—the delay between the time that a message is sent by one component and received by another.

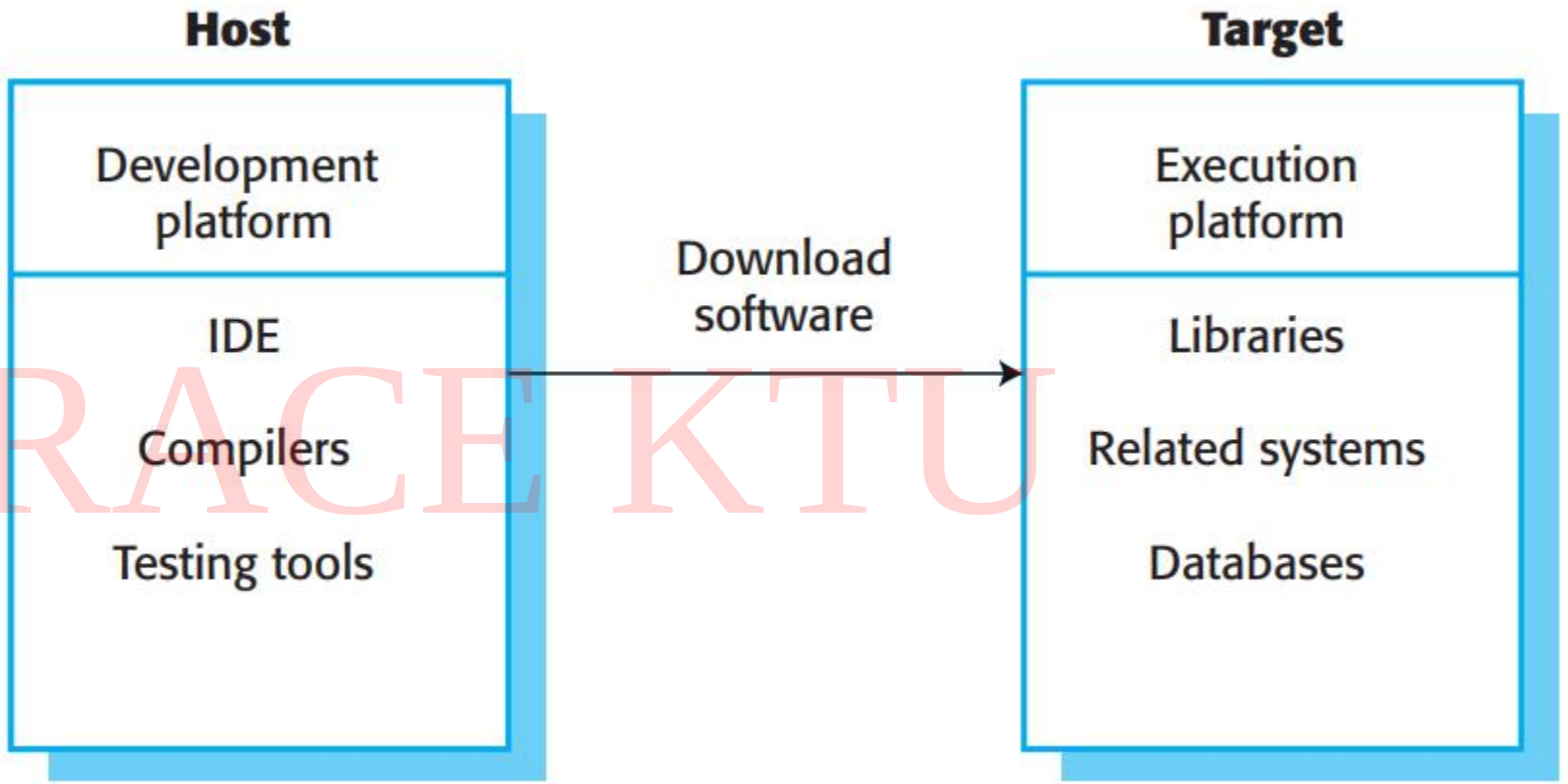

Figure 7.15 Host-target development

OPEN-SOURCE DEVELOPMENT

- Open-source development is an approach to software development in which the source code of a software system is published and volunteers are invited to participate in the development process.
- Open-source software is the backbone of the Internet and software engineering.
- The Linux operating system is the most widely used server system, as is the open-source Apache web server. Other important and universally used open-source products are Java, the Eclipse IDE, and the mySQL database management system.

- It is usually cheap or even free to acquire open-source software.
- The other key benefit of using open-source products is that widely used open-source systems are very reliable.
- They have a large population of users who are willing to fix problems themselves rather than report these problems to the developer and wait for a new release of the system.
- Bugs are discovered and repaired more quickly than is usually possible with proprietary software.

- For a company involved in software development, there are two open-source issues that have to be considered:
    1. Should the product that is being developed make use of open-source components?
    2. Should an open-source approach be used for its own software development?
- The answers to these questions depend on the type of software that is being developed and the background and experience of the development team.
- It is quicker and cheaper to modify the open-source system rather than redevelop the functionality that you need.

- Many software product companies are now using an open-source approach to development, especially for specialized systems.
- They believe that involving the open-source community will allow software to be developed more cheaply and more quickly and will create a community of users for the software.
- Some companies believe that adopting an open-source approach will reveal confidential business knowledge to their competitors and so are reluctant to adopt this development model.
- There are thousands of open-source projects on Sourceforge and GitHub.
- If users of your software have concerns about its availability in future, making the software open source means that they can take their own copy and so be reassured that they will not lose access to it.

Open-source Licensing:

- Although a fundamental principle of open-source development is that source code should be freely available, this does not mean that anyone can do as they wish with that code.
- Legally, the developer of the code (either a company or an individual) owns the code.
- They can place restrictions on how it is used by including legally binding conditions in an open-source software license.

- Most open-source licenses are variants of one of 3 general models:
  1. The **GNU General Public License (GPL)** → This is a so-called reciprocal license that simplistically means that if you use open-source software that is licensed under the GPL license, then you must make that software open source.
  2. The **GNU Lesser General Public License (LGPL)** → This is a variant of the GPL license where you can write components that link to open-source code without having to publish the source of these components. However, if you change the licensed component, then you must publish this as open source.
  3. The **Berkley Standard Distribution (BSD) License** → This is a nonreciprocal license, which means you are not obliged to re-publish any changes or modifications made to open-source code. You can include the code in proprietary systems that are sold. If you use open-source components, you must acknowledge the original creator of the code. The MIT license is a variant of the BSD license with similar conditions.

- Licensing issues are important because if you use open-source software as part of a software product, then you may be obliged by the terms of the license to make your own product open source.
- If you are trying to sell your software, you may wish to keep it secret. This means that you may wish to avoid using GPL-licensed opensource software in its development.
- If you are building software that runs on an open-source platform but that does not reuse open-source components, then licenses are not a problem.
- If you embed open-source software in your software, you need processes and databases to keep track of what's been used and their license conditions.

- Bayersdorfer suggests that companies managing projects that use open source should:
  1. Establish a system for maintaining information about open-source components that are downloaded and used.
  2. Be aware of the different types of licenses and understand how a component is licensed before it is used.
  3. Be aware of evolution pathways for components.
  4. Educate people about open source.
  5. Have auditing systems in place.
  6. Participate in the open-source community.

- The open-source approach is one of several business models for software.
- In this model, companies release the source of their software and sell add-on services and advice in association with this. They may also sell cloud-based software services -- an attractive option for users who do not have the expertise to manage their own open-source system and also specialized versions of their system for particular clients.
- Open-source is therefore likely to increase in importance as a way of developing and distributing software.