La **construcción de software** es un proceso fundamental en el ciclo de vida del desarrollo de software. Se define como el conjunto de actividades dedicadas a la **creación de software funcional mediante la codificación y la integración**. Este proceso también incluye las pruebas unitarias y de integración, aunque no abarca las pruebas de nivel superior, como las pruebas del sistema.

A continuación, se detallan las características y actividades clave de la construcción de software:

- **Integración en el Ciclo de Vida del Desarrollo de Software (SDLC)**:
    - La construcción es un aspecto inherente del ciclo de vida del desarrollo de software.
    - En modelos secuenciales como el **modelo en cascada**, el esfuerzo de construcción comienza después de que se han completado las fases de análisis de requisitos, diseño y planificación.
    - En modelos iterativos, como **Scrum**, el **prototipado evolutivo** o la **programación extrema**, la actividad de construcción puede ocurrir de forma **concurrente o solapada** con otras actividades.
    - La planificación de la construcción puede incluir la definición del orden en que se crean e integran los componentes, los procesos de gestión de la calidad del software y la asignación de tareas a equipos y desarrolladores.
- **Actividades Principales de la Construcción de Software**: La construcción de software involucra diversas actividades esenciales:
    - **Codificación**: Consiste en la escritura del código. Aspectos clave incluyen la elección de nombres para identificadores, la organización de la lógica en sentencias y rutinas (buscando alta cohesión y bajo acoplamiento), la modularidad (estructuración del código en clases y paquetes), el manejo de errores, la gestión de recursos computacionales, la seguridad (prevención de vulnerabilidades como desbordamientos de búfer), la optimización del rendimiento y la documentación (comentarios en el código y documentos externos).
    - **Integración**: Se refiere a la combinación de las partes del software construidas por separado. Implica planificar la secuencia de integración, crear un "andamio" para soportar versiones intermedias, determinar el nivel de pruebas y trabajo de calidad en los componentes antes de su integración, e identificar los puntos del proyecto donde se probarán las versiones intermedias.
    - **Pruebas**: Esta fase busca detectar fallos cometidos en etapas anteriores para corregirlos, idealmente antes de que los encuentre el usuario final. Durante la construcción, se realizan al menos **pruebas unitarias** y **pruebas de integración**, a menudo por el mismo desarrollador que escribió el código. Las pruebas pueden realizarse después de la codificación, o incluso antes, como en la programación *test-first*.
    - **Reutilización**: Implica formalizar la práctica de reutilizar componentes de software existentes. Las tareas relacionadas durante la codificación y las pruebas pueden incluir la selección de código reutilizable, la evaluación de su reusabilidad y el reporte de métricas de reutilización.
    - **Aseguramiento de la Calidad**: Se utilizan diversas técnicas para garantizar la calidad del software durante la construcción, como pruebas

(con tasas de detección de defectos que varían según el tipo de prueba), inspección de software (las inspecciones formales de código tienen una tasa de detección de defectos promedio del 60% y pueden ser más costo-efectivas para encontrar defectos de diseño que las pruebas), revisión técnica y análisis estático.
    - **Rediseño**: Durante la construcción, pueden ser necesarias modificaciones de diseño para corregir deficiencias no anticipadas en el diseño original del software.
- **Lenguajes y Mejores Prácticas**:
    - Para la construcción, se utilizan diversos tipos de lenguajes, incluyendo lenguajes de programación de propósito general, lenguajes de configuración y lenguajes de kit de herramientas. Se ha observado que los lenguajes de alto nivel (como C++, Java) ofrecen una productividad, fiabilidad, simplicidad y comprensibilidad significativamente mejores que los lenguajes de bajo nivel.
    - Las mejores prácticas buscan **minimizar la complejidad**, **anticipar cambios** (para construir software extensible que se pueda mejorar sin afectar el diseño intrínseco), **construir para la verificación** (facilitando la detección de fallos), aplicar el **ocultamiento de información** (mejorando la modificabilidad), promover la **reutilización** de activos de software existentes, adherirse a **estándares** (de documentación, modelado como UML y codificación), utilizar la **abstracción de datos**, aplicar la **programación defensiva** y gestionar adecuadamente el **manejo de errores**.
- **Relación con el Desarrollo e Ingeniería de Software**:
    - La **ingeniería de software** se ocupa del ciclo de vida completo del desarrollo de sistemas complejos, aplicando principios de ingeniería para **diseñar, desarrollar y mantener software**. Se considera un enfoque más amplio y estratégico, donde la programación es una herramienta fundamental para implementar los diseños y soluciones planificados.
    - El **desarrollo de software** es un subconjunto más específico, centrado en la **creación de aplicaciones o programas concretos**, con los desarrolladores siendo responsables principalmente de la escritura, prueba y mantenimiento del código. En este contexto, la programación es el núcleo de la actividad diaria del desarrollador, enfocada en la creación y **construcción de aplicaciones** específicas y la resolución de problemas técnicos.
    - Aunque a menudo se usan indistintamente los términos "ingeniería de software" y "desarrollo de software", la ingeniería de software implica la aplicación práctica del conocimiento científico al diseño y **construcción de programas**, buscando la **creación de software confiable y de calidad**.