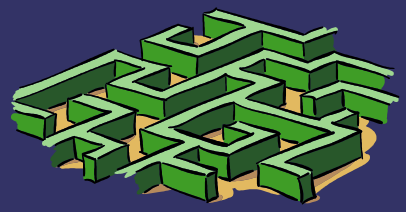
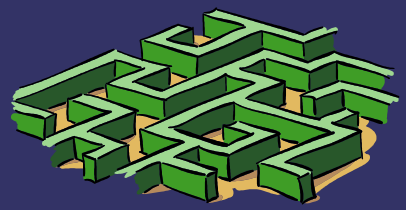
Software Engineering and the SWEBOK - An Introduction

Allen Byrne, CSDP 2003



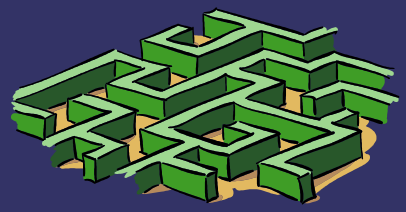
Overview

- Definitions
- Knowledge Areas
- Software Life Cycle



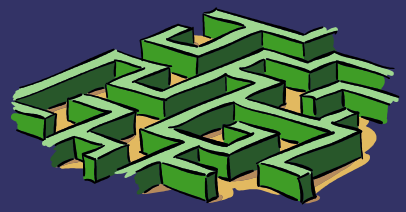
Definitions

- **Software Engineering**
- SWEBOK and the Guide



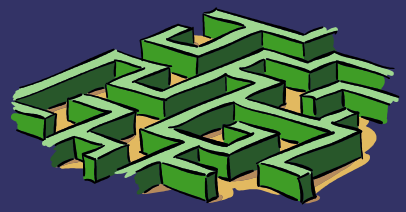
Definitions

- **Software Engineering**
  - (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
  - (2) The study of approaches as in (1).
- SWEBOK and the Guide



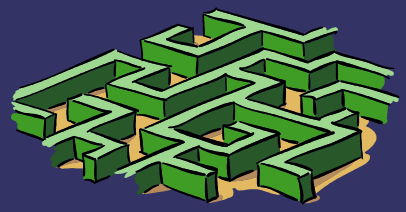
Definitions

- Software Engineering
- **SWEBOK and the Guide**



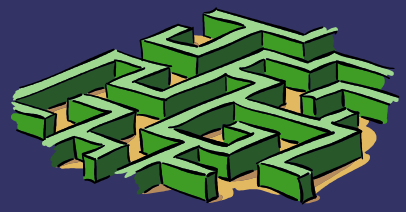
Improving the Software Lifecycle

- **Software should be a practice not an art**
- Creation should follow a well chosen process
- Maintenance will be planned
- Software should evolve



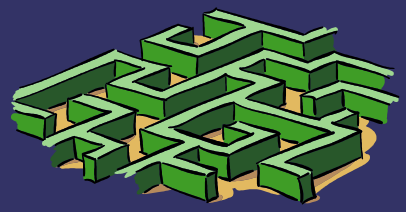
Improving the Software Lifecycle

- Software should be a practice not an art
- **Creation should follow a well chosen process**
- Maintenance will be planned
- Software should evolve



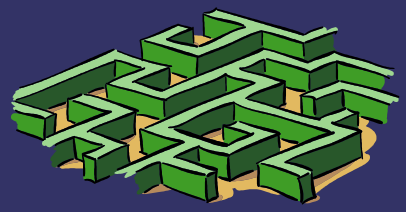
Improving the Software Lifecycle

- Software should be a practice not an art
- Creation should follow a well chosen process
- **Maintenance will be planned**
- Software should evolve



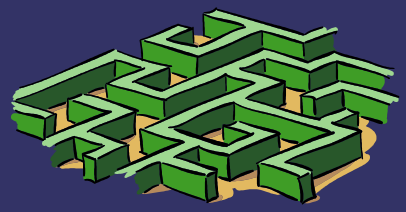
Improving the Software Lifecycle

- Software should be a practice not an art
- Creation should follow a well chosen process
- Maintenance will be planned
- **Software should evolve**



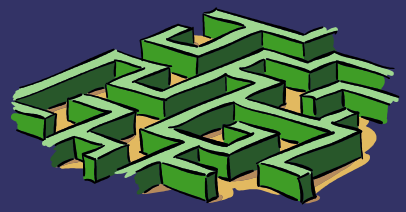
Software Engineering Goal

- Software that satisfies technical requirements
- ... satisfies user needs
- ... satisfies acquirement expectations
- ... is developed on time and within budget
- ... is easy to modify and maintain
- ... instills pride in the developers



Software Engineering Goal

Good project management and software engineering processes are required to successfully deliver a software system that finishes on time, within the allocated budget, and satisfies the needs of the system users.



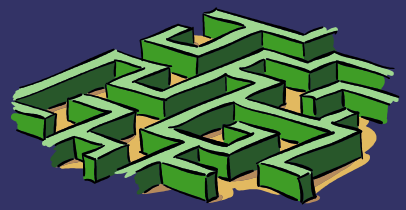
Computer Program vs Programming Product

Cost

3X

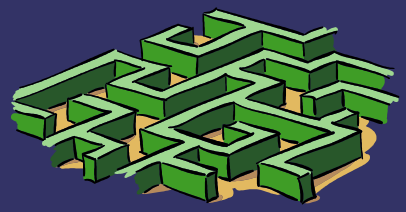
3X

A Program	A Programming System
A Programming Product	A Programming System Product



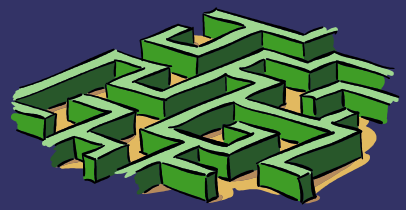
The Tragedy of software engineering is not that we don't know how to plan and conduct software projects, but that we know how and just don't do it

... Richard E. Fairley



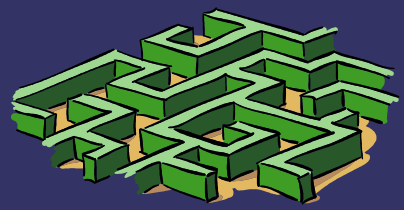
Knowledge Areas

- Software Requirements
- Software Design
- Software Construction
- Software Testing
- Software Maintenance
- Software Configuration Management
- Software Engineering Management
- Software Engineering Process
- Software Tools and Methods
- Software Quality



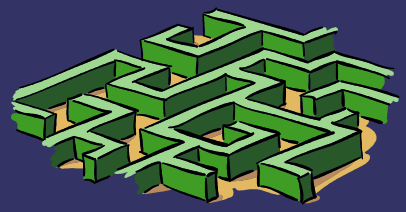
Knowledge Areas

- **Software Requirements**
- Software Design
- Software Construction
- Software Testing
- Software Maintenance
- Software Configuration Management
- Software Engineering Management
- Software Engineering Process
- Software Tools and Methods
- Software Quality



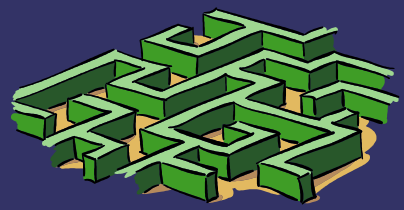
What is a Software Requirement

- A software capability needed by a user to solve a problem or achieve an objective
- A software capability that must be met or possessed by a system or system component to satisfy a contract, specification, standard, or other formally imposed document



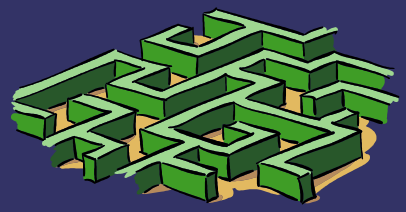
Knowledge Areas

- Software Requirements
- **Software Design**
- Software Construction
- Software Testing
- Software Maintenance
- Software Configuration Management
- Software Engineering Management
- Software Engineering Process
- Software Tools and Methods
- Software Quality



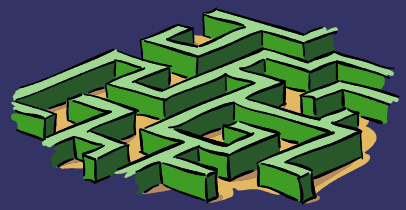
Software Design Process

- Design is a creative process
- Two phases: Architectural and Detailed



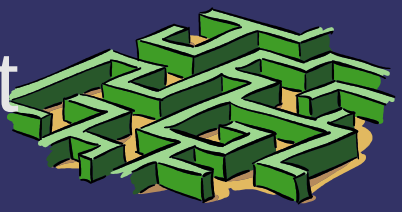
Knowledge Areas

- Software Requirements
- Software Design
- **Software Construction**
- Software Testing
- Software Maintenance
- Software Configuration Management
- Software Engineering Management
- Software Engineering Process
- Software Tools and Methods
- Software Quality



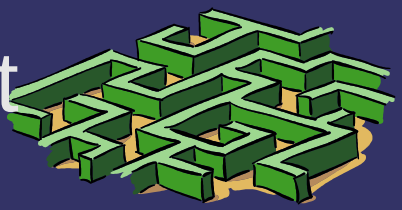
Software Construction

- **Coding and Unit Testing**
- Planning
- Quality Assurance
- Tools
- Code Design and Documentation
- Code Tuning
- Source Code Organization
- Data Design
- Error Processing
- System Integration and Deployment



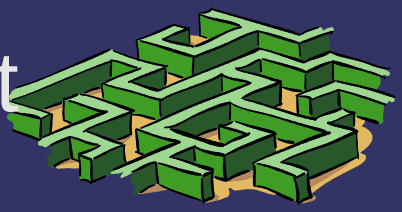
Software Construction

- Coding and Unit Testing
- **Planning**
- Quality Assurance
- Tools
- Code Design and Documentation
- Code Tuning
- Source Code Organization
- Data Design
- Error Processing
- System Integration and Deployment



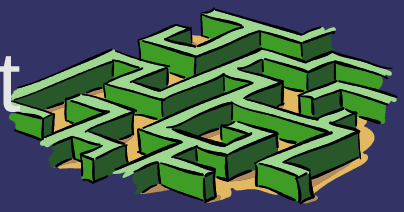
Software Construction

- Coding and Unit Testing
- Planning
- **Quality Assurance**
- Tools
- Code Design and Documentation
- Code Tuning
- Source Code Organization
- Data Design
- Error Processing
- System Integration and Deployment



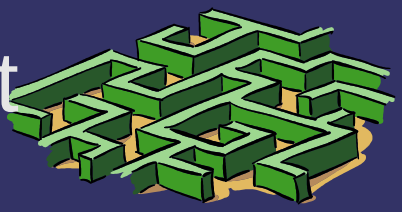
Software Construction

- Coding and Unit Testing
- Planning
- Quality Assurance
- **Tools**
- Code Design and Documentation
- Code Tuning
- Source Code Organization
- Data Design
- Error Processing
- System Integration and Deployment



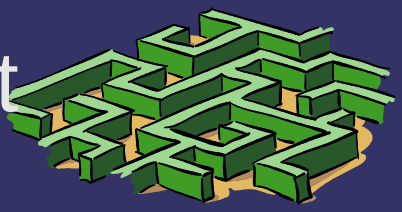
Software Construction

- Coding and Unit Testing
- Planning
- Quality Assurance
- Tools
- **Code Design and Documentation**
- Code Tuning
- Source Code Organization
- Data Design
- Error Processing
- System Integration and Deployment



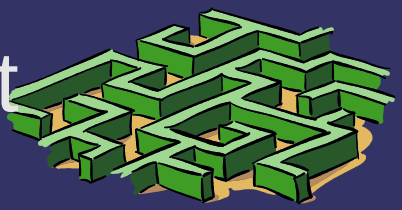
Software Construction

- Coding and Unit Testing
- Planning
- Quality Assurance
- Tools
- Code Design and Documentation
- **Code Tuning**
- Source Code Organization
- Data Design
- Error Processing
- System Integration and Deployment



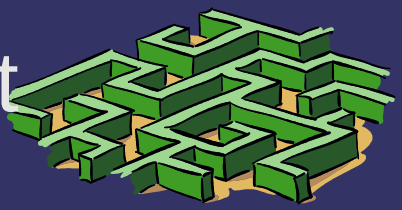
Software Construction

- Coding and Unit Testing
- Planning
- Quality Assurance
- Tools
- Code Design and Documentation
- Code Tuning
- **Source Code Organization**
- Data Design
- Error Processing
- System Integration and Deployment



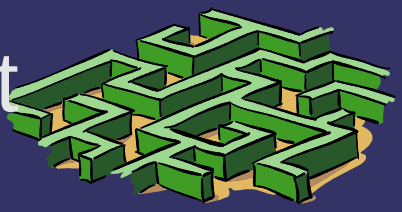
Software Construction

- Coding and Unit Testing
- Planning
- Quality Assurance
- Tools
- Code Design and Documentation
- Code Tuning
- Source Code Organization
- **Data Design**
- Error Processing
- System Integration and Deployment



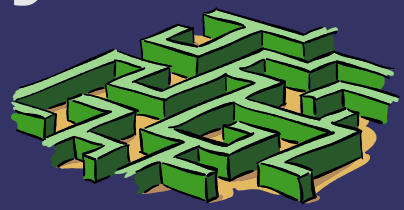
Software Construction

- Coding and Unit Testing
- Planning
- Quality Assurance
- Tools
- Code Design and Documentation
- Code Tuning
- Source Code Organization
- Data Design
- **Error Processing**
- System Integration and Deployment



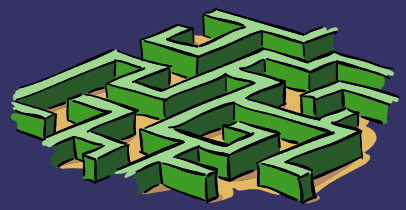
Software Construction

- Coding and Unit Testing
- Planning
- Quality Assurance
- Tools
- Code Design and Documentation
- Code Tuning
- Source Code Organization
- Data Design
- Error Processing
- **System Integration and Deployment**



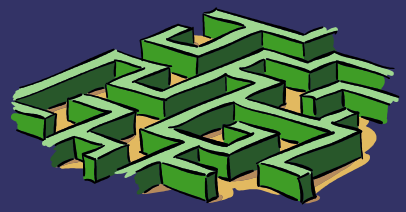
Knowledge Areas

- Software Requirements
- Software Design
- Software Construction
- **Software Testing**
- Software Maintenance
- Software Configuration Management
- Software Engineering Management
- Software Engineering Process
- Software Tools and Methods
- Software Quality



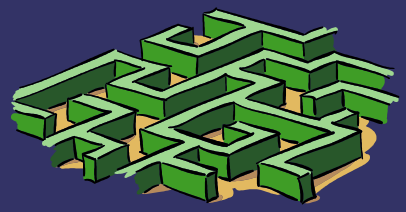
Software Testing

- **Issues and Principles**
- Test Design
- Types of Tests
- Documentation and Management



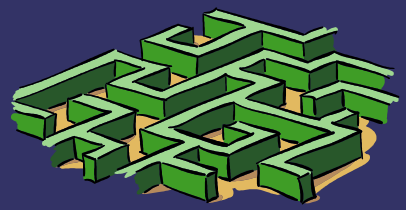
Software Testing

- Issues and Principles
- **Test Design**
- Types of Tests
- Documentation and Management



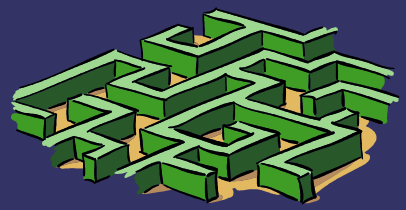
Test Design

- **Testing Strategies**
- Test Coverage Specifications
- Test Case Development



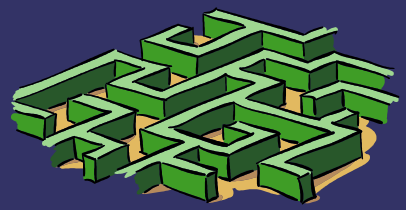
Test Design

- Testing Strategies
- **Test Coverage Specifications**
- Test Case Development



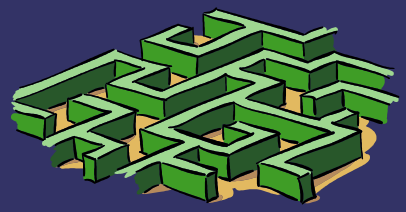
Test Design

- Testing Strategies
- Test Coverage Specifications
- **Test Case Development**



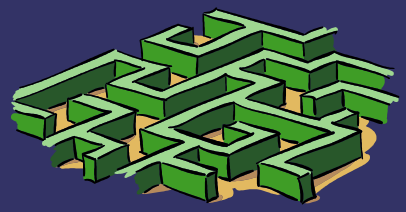
Software Testing

- Issues and Principles
- Test Design
- **Types of Tests**
- Documentation and Management



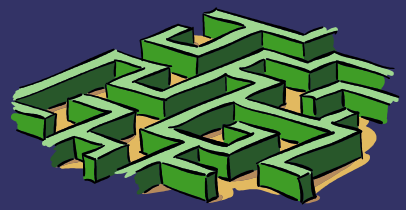
Software Testing

- Issues and Principles
- Test Design
- Types of Tests
- **Documentation and Management**



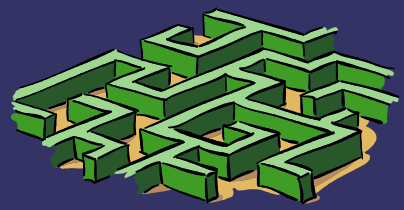
Knowledge Areas

- Software Requirements
- Software Design
- Software Construction
- Software Testing
- **Software Maintenance**
- Software Configuration Management
- Software Engineering Management
- Software Engineering Process
- Software Tools and Methods
- Software Quality



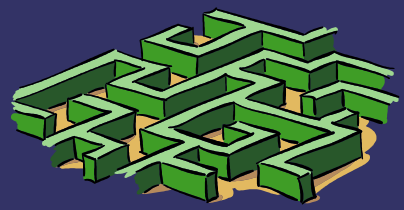
Knowledge Areas

- Software Requirements
- Software Design
- Software Construction
- Software Testing
- Software Maintenance
- **Software Configuration Management**
- Software Engineering Management
- Software Engineering Process
- Software Tools and Methods
- Software Quality



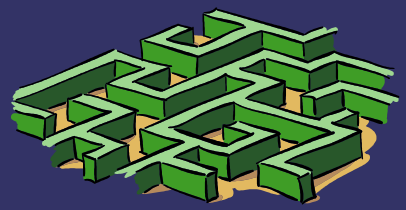
Knowledge Areas

- Software Requirements
- Software Design
- Software Construction
- Software Testing
- Software Maintenance
- Software Configuration Management
- **Software Engineering Management**
- Software Engineering Process
- Software Tools and Methods
- Software Quality



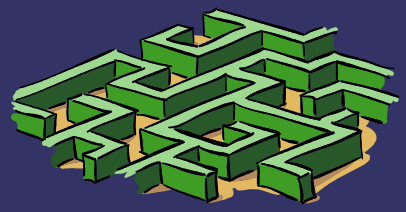
Knowledge Areas

- Software Requirements
- Software Design
- Software Construction
- Software Testing
- Software Maintenance
- Software Configuration Management
- Software Engineering Management
- **Software Engineering Process**
- Software Tools and Methods
- Software Quality



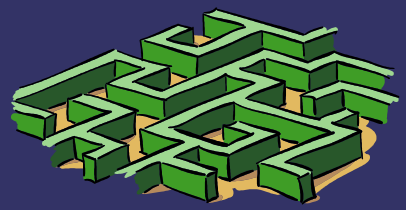
Software Engineering Process

- Establish Process Infrastructure
- Planning of Process Implementation and Change
- Process Implementation and Change
- Process Evaluation



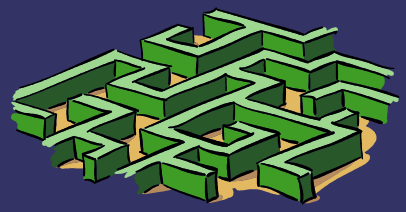
Knowledge Areas

- Software Requirements
- Software Design
- Software Construction
- Software Testing
- Software Maintenance
- Software Configuration Management
- Software Engineering Management
- Software Engineering Process
- **Software Tools and Methods**
- Software Quality



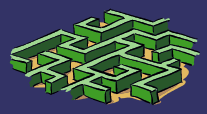
Knowledge Areas

- Software Requirements
- Software Design
- Software Construction
- Software Testing
- Software Maintenance
- Software Configuration Management
- Software Engineering Management
- Software Engineering Process
- Software Tools and Methods
- **Software Quality**



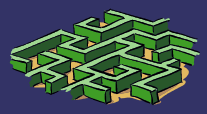
Software Engineering and the SWEBOK - An Introduction

Allen Byrne, CSDP 2003



Overview

- Definitions
- Knowledge Areas
- Software Life Cycle

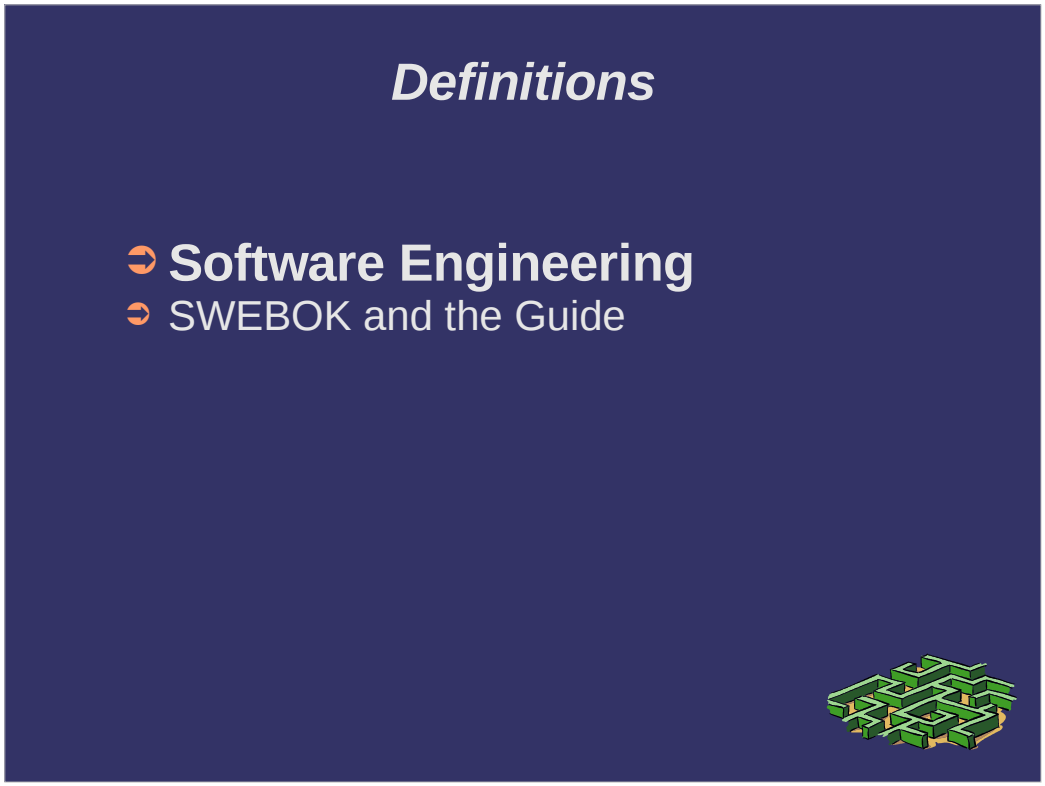


What is SE and the SWEBOK? We will define these concepts, and explore the associated knowledge areas covered by the SWEBOK.

What is the life cycle of software and how is it being managed.

Finally we will explore what can be done to improve software.

Software engineering continues to be infused with new technology and new practices. Acceptance of new techniques grows and older techniques are discarded.



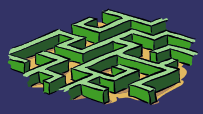
WHAT IS SOFTWARE ENGINEERING?
The IEEE Computer Society defines software engineering as:

(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

(2) The study of approaches as in (1).

Definitions

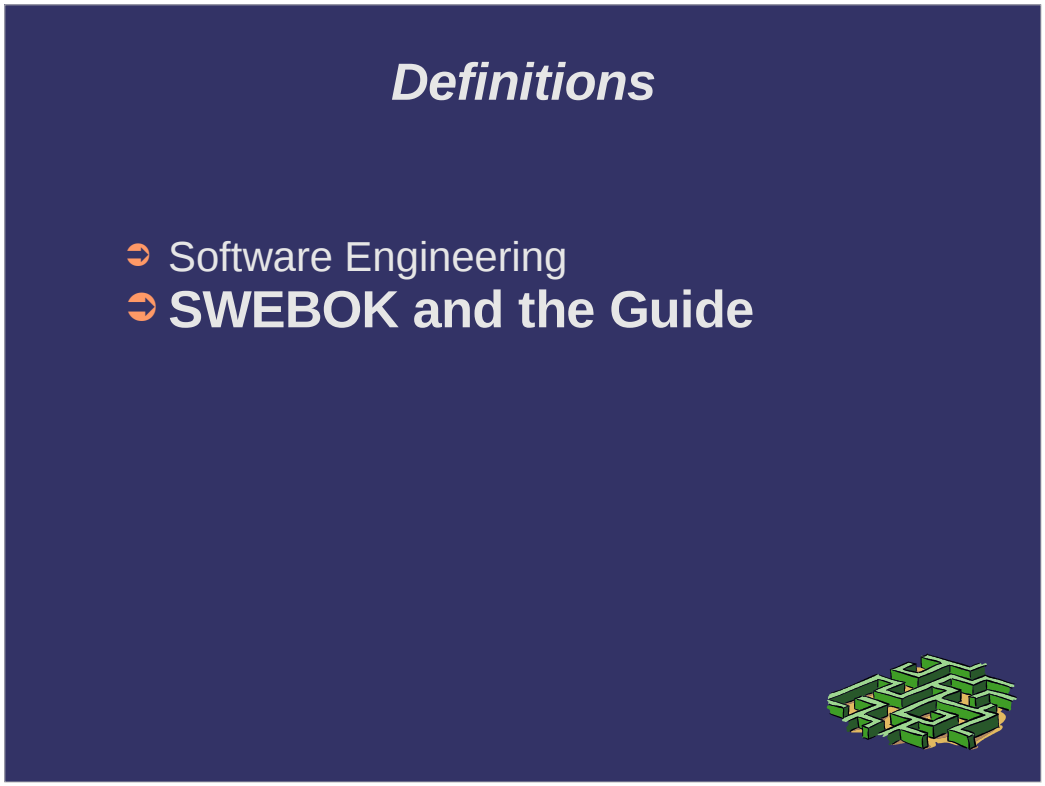
- **Software Engineering**
  - (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
  - (2) The study of approaches as in (1).
- SWEBOK and the Guide



WHAT IS SOFTWARE ENGINEERING?

Just as electrical engineering is based upon the science of physics, software engineering should be based, among others, upon computer science. In both cases, though, the emphasis is necessarily different. Scientists extend our knowledge of the laws of nature while engineers apply those laws of nature to build useful artifacts, under a number of constraints.

A consequence of an engineering-based approach is that many important aspects of information technology, that may constitute important software engineering knowledge, are not specified. In all fields—not only computing—the designers of engineering curricula have realized that specific technologies are replaced much more rapidly than the engineering work force. An engineer must be equipped with the essential knowledge that supports the selection of the appropriate technology at the appropriate time in the appropriate circumstance.

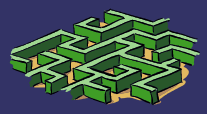


The Guide to the Software Engineering Body of Knowledge (SWEBOK) was established with the following five objectives:

1. To promote a consistent view of software engineering worldwide
2. To clarify the place–and set the boundary–of software engineering with respect to other disciplines such as computer science, project management, computer engineering, and mathematics
3. To characterize the contents of the software engineering discipline
4. To provide a topical access to the Software Engineering Body of Knowledge
5. To provide a foundation for curriculum development and for individual certification and licensing material

Improving the Software Lifecycle

- **Software should be a practice not an art**
- Creation should follow a well chosen process
- Maintenance will be planned
- Software should evolve

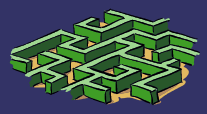


Placing clear, complete thought before action almost always produces better results. You also gain knowledge about how to do it right again. If you do think about something and still do it wrong, it becomes valuable experience. A side effect of thinking is learning to recognize when you don t know something, at which point you can research the answer. When clear thought has gone into a system, value comes out.

Software design is not a haphazard process. There are many factors to consider in any design effort. Simplicity facilitates having a more easily understood, and easily maintained system. The more elegant designs are usually the more simple ones. Simple does not mean "quick and dirty." In fact, it often takes a lot of thought and work over multiple iterations to simplify. The payoff is software that is more maintainable and less error-prone.

Improving the Software Lifecycle

- Software should be a practice not an art
- **Creation should follow a well chosen process**
- Maintenance will be planned
- Software should evolve

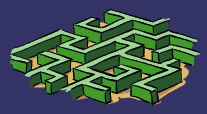


A clear vision is essential to the success of a software project. Without one, a project almost unfailingly ends up being "of two [or more] minds" about itself. Without conceptual integrity, a system threatens to become a patchwork of incompatible designs, held together by the wrong kind of screws.
Grady Booch summarizes:
It is only through having a clear sense of a system s architecture that it becomes possible to discover common abstractions and mechanisms. Exploiting this commonality ultimately leads to systems that are simpler, and therefore smaller and more reliable.

Seldom is an industrial-strength software system constructed and used in a vacuum. In some way or other, someone else will use, maintain, document, or otherwise depend on being able to understand your system. So, always specify, design, and implement knowing someone else will have to understand what you are doing. The audience for any product of software development is potentially large. Specify with an eye to the users. Design, keeping the implementers in mind. Code with concern for those that must maintain and extend the system. Someone may have to debug the code you write, and that makes them a user of your code. Making their job easier adds value to the system.

Improving the Software Lifecycle

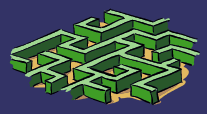
- Software should be a practice not an art
- Creation should follow a well chosen process
- Maintenance will be planned
- **Software should evolve**



A system with a long lifetime has more value. In today's computing environments, where specifications change on a moment's notice and hardware platforms are obsolete when just a few months old, software lifetimes are typically measured in months instead of years. However, true "industrial-strength" software systems must endure far longer. To do this successfully, these systems must be ready to adapt to these and other changes. Systems that do this successfully are those that have been designed this way from the start. Never design yourself into a corner. Always ask "what if ", and prepare for all possible answers by creating systems that solve the general problem, not just the specific one. This could very possibly lead to the reuse of an entire system.

Software Engineering Goal

- Software that satisfies technical requirements
- ... satisfies user needs
- ... satisfies acquirement expectations
- ... is developed on time and within budget
- ... is easy to modify and maintain
- ... instills pride in the developers

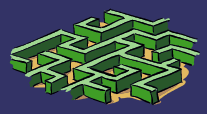


The relative success or failure of an IT project may depend on the exact meaning of the word success.

A 2005 research from consultancy Cutter Consortium, which found that IT projects premised on quality improvements—such as increased customer satisfaction, product quality or staff productivity—were much less likely to fail than projects evaluated in terms of other objectives—such as whether they came in on time and under budget.

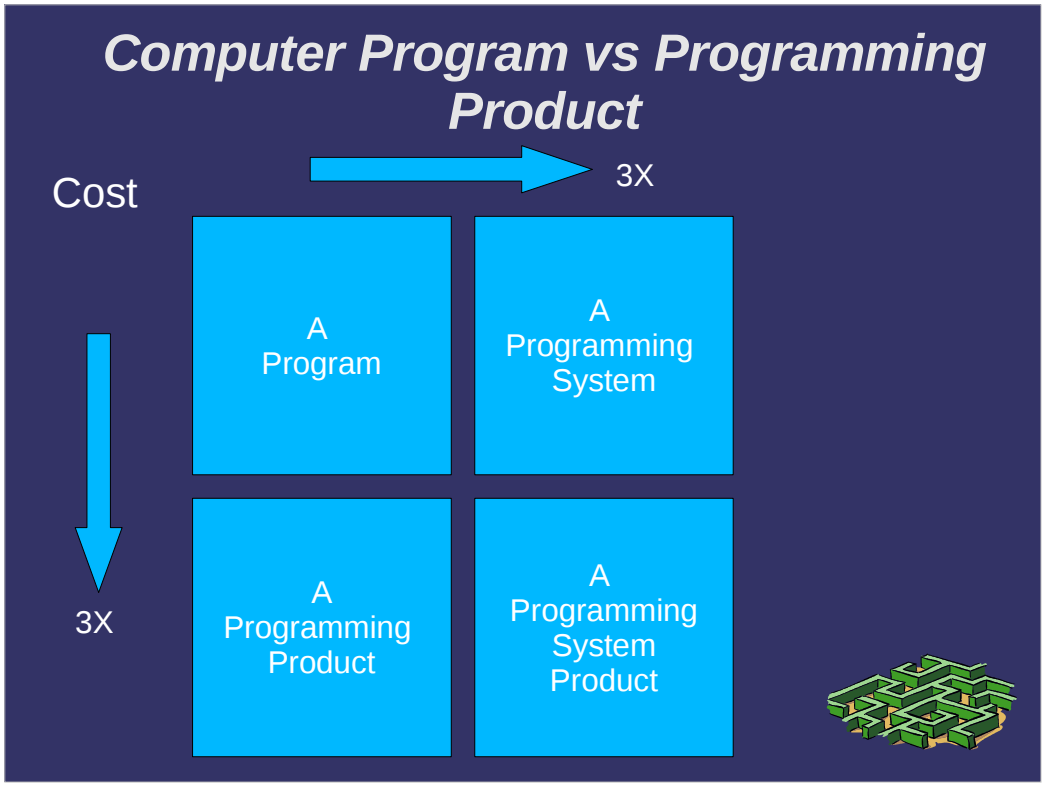
Software Engineering Goal

Good project management and software engineering processes are required to successfully deliver a software system that finishes on time, within the allocated budget, and satisfies the needs of the system users.



Most development organizations experience severe overruns, and many (nearly half) have canceled or abandoned some of their projects.

Why is it still a significant problem? For starters, whenever organizations are confronted with severe project delays or budget overruns, they typically opt to stay the course—without substantially revisiting or revising their initial project assumptions, the governance model they're using to manage and track project progress, or the capabilities of their project development team.



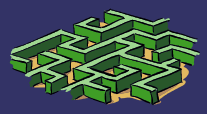
A Programming System includes:
Interfaces and, System Integration

A Programming Product includes:
Generalization, Testing, Documentation, and Maintenance

From:
The Mythical Man-Month: Essays on Software Engineering, by F.P. Brooks Jr, 1975

The Tragedy of software engineering is not that we don't know how to plan and conduct software projects, but that we know how and just don't do it

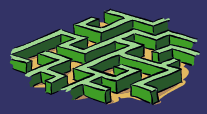
... Richard E. Fairley



Software engineering was developed out of the necessity to be able to handle large software projects that could no longer be handled by a few individuals using ad hoc methods.

Knowledge Areas

- Software Requirements
- Software Design
- Software Construction
- Software Testing
- Software Maintenance
- Software Configuration Management
- Software Engineering Management
- Software Engineering Process
- Software Tools and Methods
- Software Quality



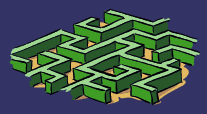
The material that is recognized as being within this discipline are organized into ten Knowledge Areas.

In establishing a boundary, it is also important to identify what disciplines share that boundary, and often a common intersection, with software engineering. Also, software engineers should have knowledge of material from eight related disciplines(and the Knowledge Areas descriptions may make reference to them).

Computer engineering
Computer science
Management
Mathematics
Project management
Quality management
Software ergonomics
Systems engineering

Knowledge Areas

- **Software Requirements**
- Software Design
- Software Construction
- Software Testing
- Software Maintenance
- Software Configuration Management
- Software Engineering Management
- Software Engineering Process
- Software Tools and Methods
- Software Quality



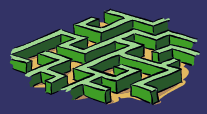
Related terms

Software Requirements Engineering
Requirements Engineering

Requirements engineering helps software engineers to better understand the problem they will work to solve. It encompasses the set of tasks that lead to an understanding of what the business impact of the software will be, what the customer wants and how end-users will interact with the software.

What is a Software Requirement

- A software capability needed by a user to solve a problem or achieve an objective
- A software capability that must be met or possessed by a system or system component to satisfy a contract, specification, standard, or other formally imposed document



Two Types

Operational: User needs, customer expectations, design goals

Technical: For the software designer and tester, mapping of operational reqs. into feasible, quantified, testable specs.

Categories

Primary: Contractual or from outside the development org.

Derived: from the Primary

Product: Apply to the product of the services; Qualitative-Not directly measurable; or Quantitative-Directly measurable

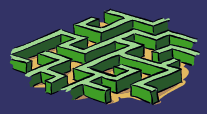
*Process:*Apply to the activities of creation of product or service; Task-Perform and analyze, develop a product operate a system; Compliance Evaluation- Measure compliance; Regulatory/Standards-Compliance with laws, standards, regulations, and rules

Compliance Level: Mandatory/Essential; Desirable/Guidance; Optional; Information

Priority: Numeric;Relative

Knowledge Areas

- Software Requirements
- **Software Design**
- Software Construction
- Software Testing
- Software Maintenance
- Software Configuration Management
- Software Engineering Management
- Software Engineering Process
- Software Tools and Methods
- Software Quality



Related terms

Design methodologies
Software architecture
Software reuse
Software portability

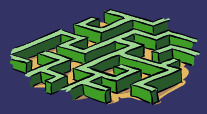
Software Design is a problem solving process in which the designer applies knowledge and experience to produce a conceptualization that defines and describes a solution to a problem.

Software Design bridges the gap between requirements and code.

Software Design produces a Software Design Description, which is a representation of software, created to facilitate analysis, planning, implementation, and decision making.

Software Design Process

- Design is a creative process
- Two phases: Architectural and Detailed



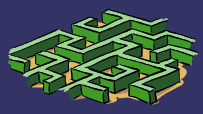
Problem solving process:
Analyze the problem, Propose solutions, Select the best solution

Architectural design:
Determines the general structure of a system and the description and interfaces between modules.

Detailed design:
Determines the contents of the modules.

Knowledge Areas

- Software Requirements
- Software Design
- **Software Construction**
- Software Testing
- Software Maintenance
- Software Configuration Management
- Software Engineering Management
- Software Engineering Process
- Software Tools and Methods
- Software Quality



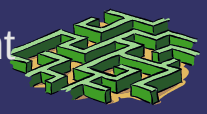
Related terms

Component-based software development
Software implementation
Software programming

Software Construction is the building of something, usually involving planning, building, and checking the final product.
The role of Software Construction is:
Verifying that the archectural design is acceptable
Designing and writing routines and modules
Creating data types and naming variables
Selecting control structures and organizing blocks of statements
Finding and fixing errors
Conducting peer reviews
Policing code through careful formatting and commenting
Integrating the completed modules
Tuning the code to make it shorter and faster

Software Construction

- **Coding and Unit Testing**
- Planning
- Quality Assurance
- Tools
- Code Design and Documentation
- Code Tuning
- Source Code Organization
- Data Design
- Error Processing
- System Integration and Deployment



Coding translates a detailed design representation of software into a programming language.
Unit testing exercises the code to establish confidence in each component.

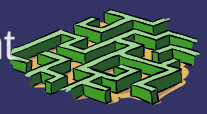
ISSUES
Coding is sometimes viewed as the only important and necessary activity.
Coding alone cannot scale up into larger systems.
Coding alone is difficult to modify, update, and maintain.
Coding is easy to learn compared to software engineering.

Coding and unit testing are labor intensive. Therefore it is a large component of the software lifecycle. Frequently, it is the only accurate description of the software and the only activity, that is guaranteed to be completed.

Good Practices
At least two people assigned to every part of the program.
Peer reviews conducted and signed-off by senior developers.
Code listings are public property and not owned by one person.
Promote good coding standards.
Keep comments up to date and appropriate.

Software Construction

- Coding and Unit Testing
- **Planning**
- Quality Assurance
- Tools
- Code Design and Documentation
- Code Tuning
- Source Code Organization
- Data Design
- Error Processing
- System Integration and Deployment



Construction Planning lays out the work plan or schedule to design, implement, debug, and unit test the software.

ISSUES

Coders are typically not planners.
Time will be difficult to manage unless a good architectural design is in place to estimate the size of the project.
Historical project data is needed on which to base future estimates. Also, individual historical data is needed on which to base rate of production estimates.
Many people consider planning to be a waste of time.
Many project plans are limited to only the construction phase.

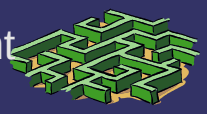
Good Practices

Allow the time for the estimate and plan accordingly.
Keep historical data on individuals and projects, and only use for estimating.
Use several different approaches, and estimate the project in pieces.
Periodically re-estimate the tasks to be done, and review those completed.

Make informed decisions when a project deviates from plans or requirements.

Software Construction

- Coding and Unit Testing
- Planning
- **Quality Assurance**
- Tools
- Code Design and Documentation
- Code Tuning
- Source Code Organization
- Data Design
- Error Processing
- System Integration and Deployment



Software Quality Assurance is a planned and systematic pattern of all actions necessary to provide adequate confidence that the software and the delivered documentation conforms to established technical requirements.
Construction Quality Assurance are those QA techniques necessary to assure that coding is being done according to coding standards.
A Quality Attribute is a requirement that specifies the degree to which an attribute affects the quality of the software; for example – reliability, maintainability, portability, and correctness.

ISSUES

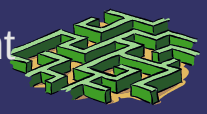
Quality is point of view dependent. How well does the software meet the users expectations. Does the software conform to specifications. How does the software compare to other products. Ease of maintenance and upgrade.

Good Practices

To improve the quality of the product you must improve the quality of the process. Institute a set of coding standards, focused code reviews, a verification and validation process. Encourage individual metric tracking.
The earlier a defect is caught, the cheaper it is to fix.

Software Construction

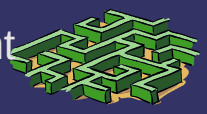
- Coding and Unit Testing
- Planning
- Quality Assurance
- **Tools**
- Code Design and Documentation
- Code Tuning
- Source Code Organization
- Data Design
- Error Processing
- System Integration and Deployment



Construction Tools are used to improve productivity and software quality.
Design tools usually involve graphic development and drawing.
Source code tools include:
editors, file comparators, code beautifiers, templates or macros, and cross-reference tools.
Quality analysis tools produce information on code dependency, syntax and semantics checks, metric reports, code translation or data dictionary descriptions.
Executable-code tools include linkers, libraries, code generators, macro preprocessors, debuggers, and profilers.

Software Construction

- Coding and Unit Testing
- Planning
- Quality Assurance
- Tools
- **Code Design and Documentation**
- Code Tuning
- Source Code Organization
- Data Design
- Error Processing
- System Integration and Deployment



Code Design lays out the detailed functionality of the individual routines. For each routine; define the problem that will be solved with enough detail to specify what information the routine will hide, the inputs and outputs, how errors will be handled, and how to test it. Also name the routine with a clear, unambiguous name, that states or implies its purpose.

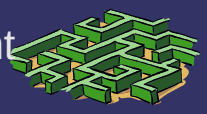
Good documentation is a sign of professional pride. Documentation can be internal, low-level more concrete in code comments or on-line help. External documentation is more abstract in a users', operators or maintenance manual.

Detailed design documentation contain:
Identification, type, purpose, function, collaboration, constraints, resources, rules and data.

Code comments should summarize the code's intent, any undocumented features or error work-around, and constraints.

Software Construction

- Coding and Unit Testing
- Planning
- Quality Assurance
- Tools
- Code Design and Documentation
- **Code Tuning**
- Source Code Organization
- Data Design
- Error Processing
- System Integration and Deployment

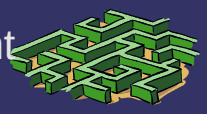


Coding tuning is the practice of modifying code to make it run more efficiently. It focuses on program design, module and routine design, operating-system interactions, code compilation, and hardware dependencies. Tuning can make code unreadable or maintainable.

Measure the system to pinpoint the areas of inefficiency. Small parts of the program can affect a disproportionate share of run time. Determine whether the real performance bottleneck comes from the design, data structures, or algorithms and whether the code is appropriately tuned. Measure each change and remove it if it does not work.

Software Construction

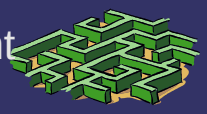
- Coding and Unit Testing
- Planning
- Quality Assurance
- Tools
- Code Design and Documentation
- Code Tuning
- **Source Code Organization**
- Data Design
- Error Processing
- System Integration and Deployment



Reasons for organizing code is routine or specification dependencies. Statements that operate on the same data, or perform similar tasks. Variables are specified close to where they are used and are active for a short amount of time. This makes code more readable and consistent.

Software Construction

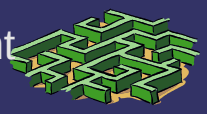
- Coding and Unit Testing
- Planning
- Quality Assurance
- Tools
- Code Design and Documentation
- Code Tuning
- Source Code Organization
- **Data Design**
- Error Processing
- System Integration and Deployment



Data design involves good choices of names and types. It requires understanding the proper use and scope of variables. In addition, error prone types and structures should be avoided.

Software Construction

- Coding and Unit Testing
- Planning
- Quality Assurance
- Tools
- Code Design and Documentation
- Code Tuning
- Source Code Organization
- Data Design
- **Error Processing**
- System Integration and Deployment



The definition of a software error is a human action that creates a software fault, which may result in a software failure.

Software Fault is the manifestation of an error in software OR an accidental condition that causes a functional unit to fail to perform its required function.

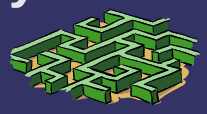
Software Failure is the inability of a system or system component to perform a required function within specified limits. OR a departure of program operation from the program requirements.

Debugging is the process of correcting syntactic and logical errors detected during coding. Or the process of locating, analyzing, and correcting suspected faults. this process is sometimes synonymous with unit testing.

Solving software problems requires:
Stabilizing the error – the error must be repeatable.
Locate the source of the error – testing.
Fix the error and test – determine the impact on other areas of the program.
Look for similar errors.

Software Construction

- Coding and Unit Testing
- Planning
- Quality Assurance
- Tools
- Code Design and Documentation
- Code Tuning
- Source Code Organization
- Data Design
- Error Processing
- **System Integration and Deployment**



System Integration can be handled two ways:

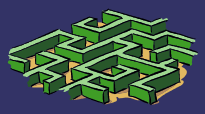
The Big Bang Integration is where all components, modules, and subsystems are combined and integrated at once. The system frequently blows up and it is next to impossible to determine the cause. In a hardware system, it is called the smoke test. A working system is not a guarantee that there are no software faults.

Incremental Integration is where a small part of the system is developed, tested, and validated. This is repeated with the next small part after which both parts are integrated, tested, and validated. Integration can be performed top-down, where the higher routines are developed and used with test stubs. Or integration can be performed bottom-up, where the low-level is developed and tested with throw away calling code. Usually, the harder problems are attacked first, and the easier problems are delayed.

Software deployment is the delivery and implementation of a software system. Alpha testing is performed by the development staff, and beta testing is performed externally on a try and see basis. Deployed systems should be placed under external configuration management.

Knowledge Areas

- Software Requirements
- Software Design
- Software Construction
- **Software Testing**
- Software Maintenance
- Software Configuration Management
- Software Engineering Management
- Software Engineering Process
- Software Tools and Methods
- Software Quality



Related terms

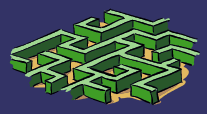
Software verification and validation

Software Testing is the process of executing a program with the intent of finding errors. A good test is one that has a high probability of finding an undiscovered error and uses a minimum number of tests to find a maximum number of errors. A successful test is one that finds an undiscovered error.

Testing can NOT show the absense of defects; it can only show that software errors are present.

Software Testing

- **Issues and Principles**
- Test Design
- Types of Tests
- Documentation and Management



ISSUES

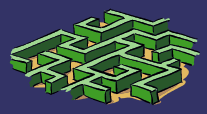
Convincing stakeholders, managers, and programmers that testing is vital to success and worthwhile, even when the project is running short on time and money.
Faulty requirements can result in faulty functional testing.
Testing can be time consuming and expensive.
Lack of explicit test planning, test procedures, and test cases can make reproducing and fixing errors difficult.

PRINCIPLES

All functional tests should be traceable to software requirements.
Tests should be planned at the beginning of the project and intermediate phases. This helps to define the scope and expectations.
Testing begins with software units and ends with system tests.
Exhaustive testing is not possible.

Software Testing

- Issues and Principles
- **Test Design**
- Types of Tests
- Documentation and Management



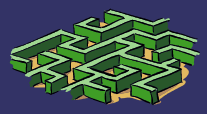
Attributes of a test design are:
Determine which features are to be tested or not tested.
Select the test cases to be used.
Select the process to test the features.
Determine the pass/fail criteria.
Design the software test as soon as possible after the establishment of the requirements.

A Software Testing Strategy is the overall plan and direction for performing testing.

A Software Testing Concept is a fundamental idea that can be applied to system testing.

Test Design

- **Testing Strategies**
- Test Coverage Specifications
- Test Case Development



Functional Testing focuses on the functional requirements of the software based on the external view of the system. This is also called “black box testing”.

Structural Testing focuses on the control structure of the system design based on the internal structures of the system. This is also known as “white box or glass box testing”.

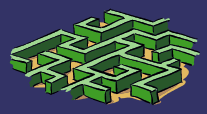
Static Analysis directly analyzes the form and structure of a product without executing the product. Static analysis tools scan the source text to detect possible faults and anomalies.

Dynamic Testing is the process of evaluating a program based on the execution of the program. The resulting data is compared with computed or expected data.

Testing can be done using top-down or bottom-up integration.

Test Design

- Testing Strategies
- **Test Coverage Specifications**
- Test Case Development



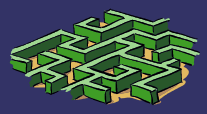
In black box testing, test data is derived from the software requirements specifications. Test coverage is the degree to which a given test or set of tests addresses all specified requirements for a given system or component.

In white box testing, test data is derived from the software design descriptions. Test coverage can be based on:
Path Testing – Exercising all independent paths within a module at least once.
Branch Testing - Exercising all logical decisions on both their true and false sides.
Loop Testing – Executing all loops at their boundaries and their operational bounds.
Interface Testing – Test the interfaces between modules and groups of modules.
Code Based Testing - Test data derived from a flowchart.

Code coverage is the percentage of the total statements executed or paths traversed.

Test Design

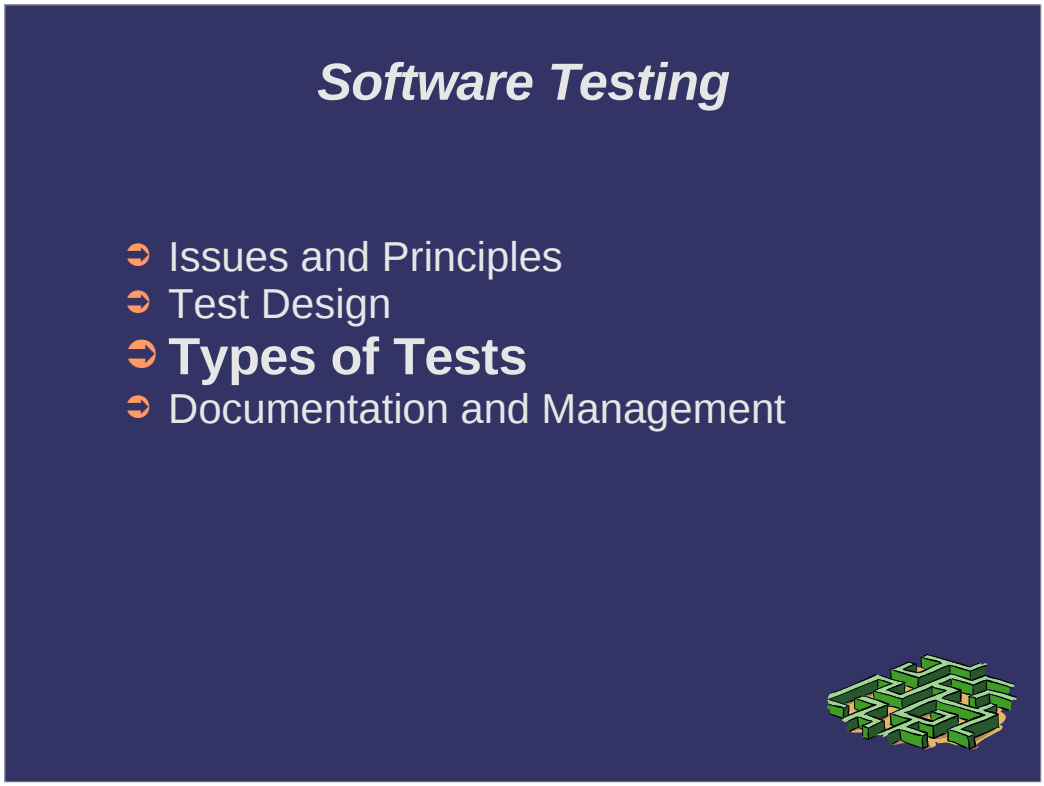
- Testing Strategies
- Test Coverage Specifications
- **Test Case Development**



Test Cases is the data that is used to simulate the actual data used to test a software system. They can be defined as:
Path Testing – The use of test cases to test each path through a software system.
Path Domain - The set of test cases that satisfies the path conditions.
Domain Error – The class of errors that result in the test case data following the wrong path.
Computational Test Data - Data used to test the computations within a software system.

Test case are generated to cover a percentage of paths, or to fall within a group of equivalence classes or domains like positive numbers, negative numbers, or strings without blanks.

Mutation Analysis is a process to provide a measure of how completely the program has been tested. It creates many nearly identical programs; each with just one change. Each mutant program is tested against the original program with same set of test data. The outputs are compared and analyzed.



Types of tests

Unit Testing is the testing of the smallest component of the software system. This is usually done by the programmer and is a type of white box testing.

Integration Testing has two purposes: Finding errors in implementing the design and finding errors in interfacing between components.

System Testing involves:

Functional Testing-which compares the systems performance with its requirements. It is normally done one function at a time and include both valid and invalid inputs.

Performance Testing-evaluates the system under stress (variables at the extremes), volume (handling large amounts of data), timing (time to respond to a user or perform a function), recovery (response to the presence of faults and effects of failures), and quality(reliability, maintainability, availability, usability, security,etc).

Stress Testing-pushes the system past the specifications.

Regression Testing-which identifies new faults that have been introduced as old faults are corrected.

Acceptance Testing is performed by the end user or on their behalf to determine the acceptance or rejection of the system. There are four types:

Benchmark-a set of test cases that reflect expected uses.

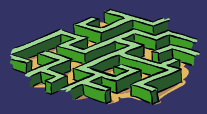
Pilot-a system installed on an experimental basis.

Alpha-development staff assuming the role of the user.

Beta-external test with a select subset of the eventual users.

Software Testing

- Issues and Principles
- Test Design
- Types of Tests
- **Documentation and Management**



Software Testing Documents consist of:
Test Plan that describes the scope, approach, resources, and schedule for testing.

Test Design Specification which specifies the details of the test approach.

Test Procedure Specification that specifies the sequence of actions for the execution of tests.

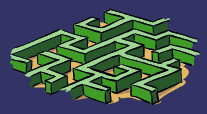
Test Case Specification specifies input, predicted results, and a set of execution conditions for each test.

Test Log which is a chronological record of the test incident report. This reports any event which requires investigation.

Test Summary Report summarizes the testing activities and results.

Knowledge Areas

- Software Requirements
- Software Design
- Software Construction
- Software Testing
- **Software Maintenance**
- Software Configuration Management
- Software Engineering Management
- Software Engineering Process
- Software Tools and Methods
- Software Quality



Related terms

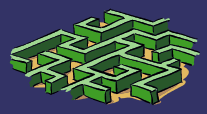
Software modification
Software reengineering

Software Maintencace is the process of modifying a software system or component, after it is accepted and placed into production, to correct faults, improve performance, add new capabilities, or adapt to a changed environment.

Corrective Maintenance **is performed to correct faults.**
Adaptive Maintenance **is performed to make aa program useful in a changed environment.**
Perfective Maintenance **improves a program's performance or functionality, likely in response to user suggestions and requests.**
Preventive Maintenance **is designing a system that is easy to maintain. Or it is the continuous upgrading of a system to enable the system to cope with current and future changes.**

Knowledge Areas

- Software Requirements
- Software Design
- Software Construction
- Software Testing
- Software Maintenance
- **Software Configuration Management**
- Software Engineering Management
- Software Engineering Process
- Software Tools and Methods
- Software Quality



Related terms

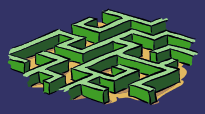
Software change management

Software Configuration Management is the process of applying administrative and technical procedures throughout the software lifecycle to:
- identify and define software items in a system
- control modifications and releases of the items
- record and report the status of the items and modification requests
- ensure the completeness, consistency, and correctness of the items
- control storage, handling, and delivery of the items.

Usually a Configuration Control Board is responsible for assessing the impact, approving, prioritizing, allocating resources, monitoring, and notifyingall parties of changes.

Knowledge Areas

- Software Requirements
- Software Design
- Software Construction
- Software Testing
- Software Maintenance
- Software Configuration Management
- **Software Engineering Management**
- Software Engineering Process
- Software Tools and Methods
- Software Quality



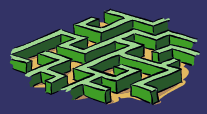
Related terms

Software project management
Software estimating
Software risk management

Software Engineering Management involves the activities and tasks udertaken by one or more persons for the purpose of planning and controlling the activities of others to achieve objectives that could not be achieved by the others acting alone.
Project Management is a system of management procedures, practices, technologies, skills, and experience necessary to successfully manage an engineering project.
The Manager plans, organizes, staffs, lead, controls, and evaluates the project.

Knowledge Areas

- Software Requirements
- Software Design
- Software Construction
- Software Testing
- Software Maintenance
- Software Configuration Management
- Software Engineering Management
- **Software Engineering Process**
- Software Tools and Methods
- Software Quality



Related terms

Software process

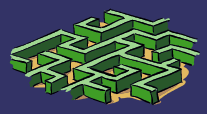
Organizational change

This Knowledge Area has witnessed dramatic growth over the last decade. Software engineering process issues are closely related to other disciplines, namely those in the management sciences, but with different terminology. Two levels; the first level encompasses the technical and managerial activities that are performed during software acquisition, development, maintenance, and retirement. The second is the meta-level, which is concerned with the definition, implementation, measurement, management, change and improvement of the process itself. The latter is termed software process engineering.

Human resources management and Systems engineering processes are outside the direct scope of this KA.

Software Engineering Process

- Establish Process Infrastructure
- Planning of Process Implementation and Change
- Process Implementation and Change
- Process Evaluation



Software process engineering consists of four activities and are sequenced in an iterative cycle allowing for continuous feedback and improvement of the software process.

The “Establish Process Infrastructure” activity consists of establishing commitment to process implementation and change (including obtaining management buy-in), and putting in place an appropriate infrastructure (resources and responsibilities) to make it happen.

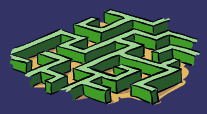
The activities “Planning of Process Implementation and Change” and “Process Implementation and Change” are the core ones in process engineering, in that they are essential for any long-lasting benefit from process engineering to accrue. In the planning activity the objective is to understand the current business objectives and process needs of the organization1, identify its strengths and weaknesses, and make a plan for process implementation and change. In “Process Implementation and Change”, the objective is to execute the plan, deploy new processes (which may involve, for example, the deployment of tools and training of staff), and/or change existing processes.

The fourth activity, “Process Evaluation” is concerned with finding out how well the implementation and change went; whether the expected benefits materialized. This is then used as input for subsequent cycles.

“Process Experience Base”, captures previous experiences to be adapted and reused. It is also important to continuously re-assess the experience base to ensure that obsolete information does not accumulate.

Knowledge Areas

- Software Requirements
- Software Design
- Software Construction
- Software Testing
- Software Maintenance
- Software Configuration Management
- Software Engineering Management
- Software Engineering Process
- **Software Tools and Methods**
- Software Quality



Related terms

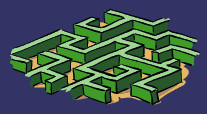
Computer aided software engineering
Software development environment
Software programming tools

Software development environments are the computer-based tools that are intended to assist the software development process. Tools allow repetitive, well-defined actions to be automated, thus reducing the cognitive load on the software engineer. The engineer is then free to concentrate on the creative aspects of the process.

Development methods impose structure on the software development activity with the goal of making the activity systematic and ultimately more likely to be successful. Methods usually provide a notation and vocabulary, procedures for performing identifiable tasks and guidelines for checking both the process and the product.

Knowledge Areas

- Software Requirements
- Software Design
- Software Construction
- Software Testing
- Software Maintenance
- Software Configuration Management
- Software Engineering Management
- Software Engineering Process
- Software Tools and Methods
- **Software Quality**



Related terms

Safety-critical Systems
Software reliability
Software measurement
Software metrics

Software Quality Assurance is a planned and systematic pattern of all actions necessary to provide adequate confidence that the software and the delivered documentation conforms to established technical standards.
Quality Assurance may make use of of supporting processes, such as verification, validation, joint reviews, audits, metrics, and problem resolution. They are not responsible for testing, just forr determining that it was done correctly.
Verification determines that a product fulfills the requirments.
Validation determineswheter the requirements and the final product fulfills its specific intended purpose.