Introduction to Software Engineering/Print version

Table of contents

Profiling
Test-driven Development
Refactoring

Software Quality

Introduction
Static Analysis
Metrics
Metrics2
Visualization
Code Review
Code Inspection

Deployment & Maintenance

Introduction
Maintenance
Evolution

Project Management

Introduction
Software Estimation
Cost Estimation
Development Speed

Tools

Introduction
Modelling and Case Tools
Compiler
Debugger
IDE
GUI Builder
Source Control
Build Tools
Software Documentation
Static Code Analysis
Profiling
Code Coverage
Project Management
Continuous Integration
Bug Tracking
Decompiler
Obfuscation

Re-engineering

Introduction
Reverse Engineering
Round-trip Engineering

Other

Introduction

References

Editors

Editors

Authors

Authors

License

Preface

When preparing an undergraduate class on Software Engineering, I found that there are a lot of good articles in Wikipedia covering different aspects related to software engineering. For a beginner, however, it is not so easy to find her or his way through that jungle of articles. It is not evident what is important and what is less relevant, where to start and what to skip in a first reading. Also, these articles contain too much information and too few examples. Hence the idea for this book came about: to take the relevant articles from Wikipedia, combine them, edit them, fill in the missing pieces, put them in context and create a wikibook out of them.

The hope is that this can be used as a textbook for an introductory software engineering class. The advantage for the instructor is that she can just pick the pieces that fit into her course and create a collection. The advantage for the student is that he can have a printed or pdf version of the textbook at a reasonable price (free) and with reasonable licenses (creative commons).

As for the philosophy behind this book: brevity is preferred to completeness, and examples are preferred to theory. If this effort was successful, you be the judge of it, and if you have suggestions for improvement, just use the 'Edit' button!

My special thanks go to Adrignola and Kayau who did the tedious work of importing the original articles (with all their history) from Wikipedia to Wikibooks!

Software Engineering

Introduction

This book is an introduction to the art of software engineering. It is intended as a textbook for an undergraduate level course.

Software engineering is about teams. The problems to solve are so complex or large, that a single developer cannot solve them anymore. Software engineering is also about communication. Teams do not consist only of developers, but also of testers, architects, system engineers, customer, project managers, etc. Software projects can be so large that we have to do careful planning. Implementation is no longer just writing code, but it is also following guidelines, writing documentation and also writing unit tests. But unit tests alone are not enough. The different pieces have to fit together. And we have to be able to spot problematic areas using metrics. They tell us if our code follows certain standards. Once we are finished coding, that does not mean that we are finished with the project: for large projects maintaining software can keep many people busy for a long time. Since there are so many factors influencing the success or failure of a project, we also need to learn a little about project management and its pitfalls, but especially what makes projects successful. And last but not least, a good software engineer, like any engineer, needs tools, and you need to know about them.

Developers Work in Teams

In your beginning semesters you were coding as individuals. The problems you were solving were small enough so one person could master them. In the real world this is different:- the problem sizes and time constraints are such that only teams can solve those problems.

For teams to work effectively they need a language to communicate (UML). Also teams do not consist only of developers, but also of testers, architects, system engineers and most importantly the customer. So we need to learn about what makes good teams, how to communicate with the customer, and how to document not only the source code, but everything related to the software project.

New Language

In previous courses we learned languages, such as Java or C++, and how to turn ideas into code. But these ideas are independent of the language. With Unified Modeling Language (UML) we will see a way to describe code independently of language, and more importantly, we learn to think in one higher level of abstraction. UML can be an invaluable communication and documentation tool.

We will learn to see the big picture: patterns. This gives us yet one higher level of abstraction. Again this increases our vocabulary to communicate more effectively with our peers. Also, it is a fantastic way to learn from our seniors. This is essential for designing large software systems.

Measurement

Also just being able to write software, doesn't mean that the software is any good. Hence, we will discover what makes good software, and how to measure software quality. On one hand we should be able to analyse existing source code through static analysis and measuring metrics, but also how do we guarantee that our code meets certain quality standards? Testing is also important in this context, it guarantees high quality products.

New Tools

Up to now, you may have come to know about an IDE, a compiler and a debugger. But there are many more tools at the disposal of a software engineer. There are tools that allow us to work in teams, to document our software, to assist and monitor the whole development effort. There are tools for software architects, tools for testing and profiling, automation and re-engineering.

History

When the first modern digital computers appeared in the early 1940s,[1] the instructions to make them operate were wired into the machine. At this time, people working with computers were engineers, mostly electrical engineers. This hardware centric design was not flexible and was quickly replaced with the "stored program architecture" or von Neumann architecture. Thus the first division between "hardware" and "software" began with abstraction being used to deal with the complexity of computing.

Programming languages started to appear in the 1950s and this was also another major step in abstraction. Major languages such as Fortran, ALGOL, and COBOL were released in the late 1950s to deal with scientific, algorithmic, and business problems respectively. E.W. Dijkstra wrote his seminal paper, "Go To Statement Considered Harmful",[2] in 1968 and David Parnas introduced the key concept of modularity and information hiding in 1972[3] to help programmers deal with the ever increasing complexity of software systems. A software system for managing the hardware called an operating system was also introduced, most notably by Unix in 1969. In 1967, the Simula language introduced the object-oriented programming paradigm.

These advances in software were met with more advances in computer hardware. In the mid 1970s, the microcomputer was introduced, making it economical for hobbyists to obtain a computer and write software for it. This in turn led to the now famous Personal Computer (PC) and Microsoft Windows. The Software Development Life Cycle or SDLC was also starting to appear as a consensus for centralized construction of software in the mid 1980s. The late 1970s and early 1980s saw the introduction of several new Simula-inspired object-oriented programming languages, including Smalltalk, Objective-C, and C++.

Open-source software started to appear in the early 90s in the form of Linux and other software introducing the "bazaar" or decentralized style of constructing software.[4] Then the World Wide Web and the popularization of the Internet hit in the mid 90s, changing the engineering of software once again. Distributed systems gained sway as a way to design systems, and the Java programming language was introduced with its virtual machine as another step in abstraction. Programmers collaborated and wrote the Agile Manifesto, which favored more lightweight processes to create cheaper and more timely software.

The current definition of *software engineering* is still being debated by practitioners today as they struggle to come up with ways to produce software that is "cheaper, better, faster". Cost reduction has been a primary focus of the IT industry since the 1990s. Total cost of ownership represents the costs of more than just acquisition. It includes things like productivity impediments, upkeep efforts, and resources needed to support infrastructure.

References

1. Leondes (2002). *intelligent systems: technology and applications*. CRC Press. ISBN 9780849311215.
2. Dijkstra, E. W. (March 1968). "Go To Statement Considered Harmful". *Wikipedia:Communications of the ACM* **11** (3): 147–148. doi:10.1145/362929.362947. http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD215.PDF. Retrieved 2009-08-10.
3. Parnas, David (December 1972). "On the Criteria To Be Used in Decomposing Systems into Modules". *Wikipedia:Communications of the ACM* **15** (12): 1053–1058. doi:10.1145/361598.361623. http://www.acm.org/classics/may96/. Retrieved 2008-12-26.
4. Raymond, Eric S. *The Cathedral and the Bazaar* (http://www.catb.org/esr/writings/cathedral-bazaar/). ed 3.0. 2000.

Further Reading

History of software engineering (http://en.wikipedia.org/wiki/History_of_software_engineering)

Software Engineer

Software engineering is done by the software engineer, an engineer who applies the principles of software engineering to the design and development, testing, and evaluation of software and systems that make computers or anything containing software work. There has been some controversy over the term engineer[1], since it implies a certain level of academic training, professional discipline, adherence to formal processes, and especially legal liability that often are not applied in cases of software development. In 2004, the U. S. Bureau of Labor Statistics counted 760,840 software engineers holding jobs in the U.S.; in the same period there were some 1.4 million practitioners employed in the U.S. in all other engineering disciplines combined.[2]

Overview

Prior to the mid-1990s, software practitioners called themselves *programmers* or *developers,* regardless of their actual jobs. Many people prefer to call themselves *software developer* and *programmer*, because most widely agree what these terms mean, while *software engineer* is still being debated. A prominent computing scientist, E. W. Dijkstra, wrote in a paper that the coining of the term *software engineer* was not a useful term since it was an inappropriate analogy, "The existence of the mere term has been the base of a number of extremely shallow --and

false-- analogies, which just confuse the issue...Computers are such exceptional gadgets that there is good reason to assume that most analogies with other disciplines are too shallow to be of any positive value, are even so shallow that they are only confusing."[3]

The term *programmer* has often been used to refer to those without the tools, skills, education, or ethics to write good quality software. In response, many practitioners called themselves *software engineers* to escape the stigma attached to the word programmer.

The label software engineer is used very liberally in the corporate world. Very few of the practicing software engineers actually hold Engineering degrees from accredited universities. In fact, according to the Association for Computing Machinery, "most people who now function in the U.S. as serious software engineers have degrees in computer science, not in software engineering". [4]

Education

About half of all practitioners today have computer science degrees. A small, but growing, number of practitioners have software engineering degrees. In 1987 Imperial College London introduced the first three-year software engineering Bachelor's degree in the UK and the world. Since then, software engineering undergraduate degrees have been established at many universities. A standard international curriculum for undergraduate software engineering degrees was recently defined by the ACM[5]. As of 2004, in the U.S., about 50 universities offer software engineering degrees, which teach both computer science and engineering principles and practices. ETS University and UQAM were mandated by IEEE to develop the SoftWare Engineering BOdy of Knowledge (SWEBOK) [6], which has become an ISO standard describing the body of knowledge covered by a software engineer.

In business, some software engineering practitioners have Management Information Systems (MIS) degrees. In embedded systems, some have electrical engineering or computer engineering degrees, because embedded software often requires a detailed understanding of hardware. In medical software, practitioners may have medical informatics, general medical, or biology degrees. Some practitioners have mathematics, science, engineering, or technology degrees. Some have philosophy (logic in particular) or other non-technical degrees, and others have no degrees.

Profession

Most software engineers work as employees or contractors. They work with businesses, government agencies (civilian or military), and non-profit organizations. Some software engineers work for themselves as freelancers. Some organizations have specialists to perform each of the tasks in the software development process. Other organizations required software engineers to do many or all of them. In large projects, people may specialize in only one role. In small projects, people may fill several or all roles at the same time.

There is considerable debate over the future employment prospects for Software Engineers and other Information Technology (IT) Professionals. For example, an online futures market called the Future of IT Jobs in America[7] attempts to answer whether there will be more IT jobs, including software engineers, in 2012 than there were in 2002.

Some students in the developed world may have avoided degrees related to software engineering because of the fear of offshore outsourcing and of being displaced by foreign workers.[8] Although government statistics do not currently show a threat to software engineering itself; a related career, computer programming, does appear to have been affected.[9][10] Some career counselors suggest a student to also focus on "people skills" and business skills rather than purely technical skills, because such "soft skills" are allegedly more difficult to offshore.[11] It is the quasi-management aspects of software engineering that appear to be what has kept it from being impacted by globalization.[12]

References

1. Sayo, Mylene. "[http://www.peo.on.ca/enforcement/June112002newsrelease.html What's in a Name? Tech Sector battles Engineers on "software engineering""]. http://www.peo.on.ca/enforcement/June112002newsrelease.html. Retrieved 2008-07-24
2. Bureau of Labor Statistics, U.S. Department of Labor, *USDL 05-2145: Occupational Employment and Wages, November 2004* (ftp://ftp.bls.gov/pub/news.release/ocwage.txt)
3. http://www.cs.utexas.edu/users/EWD/transcriptions/EWD06xx/EWD690.html E.W.Dijkstra Archive: The pragmatic engineer versus the scientific designer
4. http://computingcareers.acm.org/?page_id=12 ACM, Computing - Degrees & Careers, Software Engineering
5. http://sites.computer.org/ccse/ Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering
6. http://www.computer.org/portal/web/swebok Guide to the Software Engineering Body of Knowledge
7. Future of IT Jobs in America (http://www.ideosphere.com/fx-bin/Claim?claim=ITJOBS)
8. As outsourcing gathers steam, computer science interest wanes (http://www.computerworld.com/printthis/2006/0,4814,111202,00.html)
9. Computer Programmers (http://www.bls.gov/oco/ocos110.htm#outlook)

10. Software developer growth slows in North America | InfoWorld | News | 2007-03-13 | By Robert Mullins, IDG News Service (http://www.infoworld.com/article/07/03/13/HNslowsoftdev_1.html)
11. Hot Skills, Cold Skills (http://www.computerworld.com/action/article.do?command=viewArticleTOC&specialReportId=9000100&articleId=112360)
12. Dual Roles: The Changing Face of IT (http://itmanagement.earthweb.com/career/article.php/3523066)

UML

Introduction

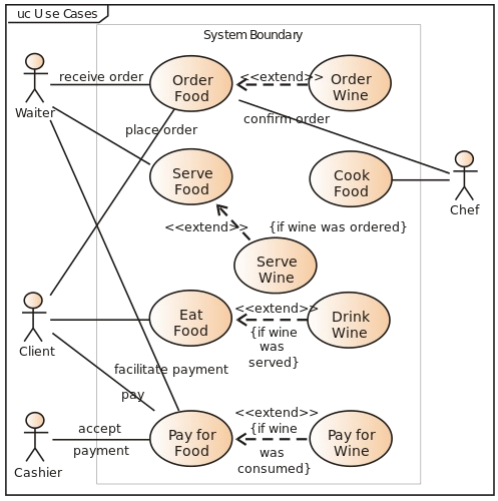
Software engineers speak a funny language called *Unified Modeling Language*, or UML for short. Like a musician has to learn musical notation before being able to play piano, we need to learn UML before we are able to engineer software. UML is useful in many parts of the software engineering process, for instance: planning, architecture, documentation, or reverse engineering. Therefore, it is worth our efforts to know it.

Designing software is a little like writing a screenplay for a Hollywood movie. The characteristics, actions, and interactions of the characters are carefully planned, as is the relevant components of their environment. As an introductory example, consider our friend Bill, a customer, who is at a restaurant for dinner. His waiter is Linus, who takes the orders and brings the food. In the kitchen is Larry, the cook. Steve is the cashier. In this way, we've provided useful and easily accessed information about the operation of a restaurant in the screenplay.

Use Case Diagram

The use case model is representation of the systems intended functions and its environment.The first thing a software engineer does is to draw a Use Case diagram. All the *actors* are represented by little stick figures, all the *actions* are represented by ovals and are called *use cases*. The actors and use cases are connected by lines. Very often there is also one or more system boundaries. Actors, which usually are not part of the system, are drawn outside the system area.

Use Case diagrams are very simple, so even managers can understand them. But they are very helpful in understanding the system to be designed. They should list all parties involved in the system and all major actions that the system should be able to perform. The important thing about them is that you don't forget anything, it is less important that they are super-detailed, for this we have other diagram types.

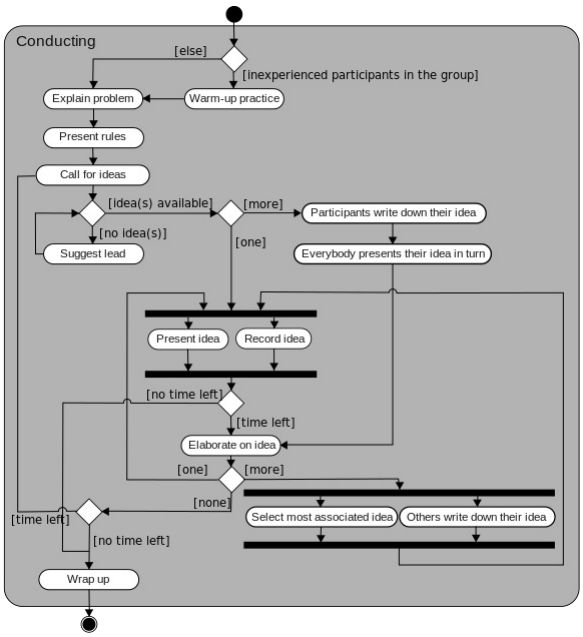


Restaurant Use Case Diagram.

Activity Diagram

Next, we will draw an Activity diagram. The Activity diagram gives more detail to a given use case and it often depicts the flow of information, hence it is also called a Flowchart. Where the Use Case diagrams has no timely order, the Activity diagram has a beginning and an end, and it also depicts decisions and repetitions.

If the Use Case diagram names the actors and gives us the headings for each scene (use case) of our play, the Activity diagram tells the detailed story behind each scene. Some managers may be able to understand Activity diagrams, but don't count on it.



Restaurant Activity Diagram.

Sequence Diagram

Once we are done drawing our Activity diagrams, the next step of refinement is the Sequence diagram. In this diagram we list the actors or objects horizontally and then we depict the messages going back and forth between the objects by horizontal lines. Time is always progressing downwards in this diagram.

File:Restaurant Sequence Diagram.png Restaurant Sequence Diagram.

The Sequence diagram is a very important step in what is called the process of object-oriented analysis and design. This diagram is so important, because on the one hand it identifies our objects/classes and on the other hand it also gives us the methods for each of those classes, because each message turns into a method. Sequence diagrams can become very large, since they basically describe the whole program. Make sure, you cover every path in your Sequence diagrams, but try to avoid unnecessary repetition. Managers will most likely not understand Sequence diagrams.

Collaboration Diagram

The Collaboration diagram is an intermediate step to get us from the Sequence diagram to the Class diagram. It is similar to the Sequence diagram, but it has a different layout. Instead of worrying about the timeline, we worry about the interactions between the objects. Each object is represented by a box, and interactions between the objects are shown by arrows.

File:Restaurant Collaboration Diagram.png Restaurant Collaboration Diagram.

This diagram shows the responsibility of objects. If an object has too much responsibility, meaning there are too many lines going in and out of a box, probably something is wrong in your design. Usually you would want to split the box into two or more smaller boxes. At this stage in your design, this can still be done easily. Try to do that once you started coding, or even later, it will become a nightmare.

Class Diagram

For us as software engineers, at least the object-oriented kind, the Class diagram is the most important one. A Class diagram consists of *classes* and lines between them. The classes themselves are drawn as boxes, having two compartments, one for *methods* and one for *attributes*.

File:Restaurant Class Diagram.png Restaurant Class Diagram.

You start with the Collaboration diagram, and the first thing you do is take all the boxes and call them classes now. Next, instead of having many lines going between the objects, you replace them by one line. But for every line you remove, you must add a method entry to the class's method compartment. So at this stage the Class diagram looks quite similar to the Collaboration diagram.

The things that make a Class diagram different are the attributes and the fact that there is not only one type of line, but several different kinds. As for the attributes, you must look at each class carefully and decide which variables are needed for this class to function. If the class is merely a data container this is easy, if the class does some more complicated things, this may not be so easy.

As for the lines, we call them relationships between the objects, and basically there are three major kinds:

- the association (*has a*): a static relationship, usually one class is attribute of another class, or one class uses another class
- the aggregation (*consists of*): for instance an order consists of order details

- and the inheritance (*is a*): describes a hierarchy between classes

Now with the Class diagram finished, you can lean back: if you have a good UML Modelling tool you simply click on the 'Generate Code' button and it will create stubs for all the classes with methods and attributes in your favorite programming language. By the way, don't expect your manager to understand class diagrams.

UML Models and Diagrams

The Unified Modeling Language is a standardized general-purpose modeling language and nowadays is managed as a de facto industry standard by the Object Management Group (OMG).[1] UML includes a set of graphic notation techniques to create visual models of software-intensive systems.[2]

History

UML was invented by the *Three Amigos*: James Rumbaugh, Grady Booch and Ivar Jacobson. After Rational Software Corporation hired James Rumbaugh from General Electric in 1994, the company became the source for the two most popular object-oriented modeling approaches of the day: Rumbaugh's Object-modeling technique (OMT), which was better for object-oriented analysis (OOA), and Grady Booch's Booch method, which was better for object-oriented design (OOD). They were soon assisted in their efforts by Ivar Jacobson, the creator of the object-oriented software engineering (OOSE) method. Jacobson joined Rational in 1995, after his company, Objectory AB,[3] was acquired by Rational.

Definition

The Unified Modeling Language (UML) is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software-intensive system under development.[4] UML offers a standard way to visualize a system's architectural blueprints, including elements such as activities, actors, business processes, database schemas, components, programming language statements, and reusable software components.[5]

UML combines techniques from data modeling (entity relationship diagrams), business modeling (work flows), object modeling, and component modeling. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies.[6]

Models and Diagrams

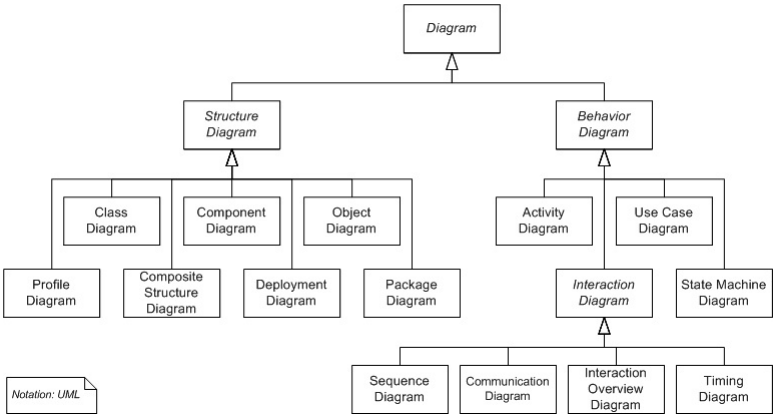
It is important to distinguish between the UML model and the set of diagrams of a system. A diagram is a partial graphic representation of a system's model. The model also contains documentation that drive the model elements and diagrams.

UML diagrams represent two different views of a system model [7]:

- Static (or structural) view: emphasizes the static structure of the system using objects, attributes, operations and relationships. The structural view includes class diagrams and composite structure diagrams.
- Dynamic (or behavioral) view: emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. This view includes sequence diagrams, activity diagrams and state machine diagrams.

Diagrams Overview

In UML 2.2 there are 14 types of diagrams divided into two categories.[8] Seven diagram types represent structural information, and the other seven represent general types of behavior, including four that represent different aspects of interactions. These diagrams can be categorized hierarchically as shown in the following diagram:



Structure Diagrams

Structure diagrams emphasize the things that must be present in the system being modeled. Since structure

diagrams represent the structure, they are used extensively in documenting the software architecture of software systems.

- Class diagram: describes the structure of a system by showing the system's classes, their attributes, and the relationships among the classes.
- Component diagram: describes how a software system is split up into components and shows the dependencies among these components.
- Composite structure diagram: describes the internal structure of a class and the collaborations that this structure makes possible.
- Deployment diagram: describes the hardware used in system implementations and the execution environments and artifacts deployed on the hardware.
- Object diagram: shows a complete or partial view of the structure of a modeled system at a specific time.
- Package diagram: describes how a system is split up into logical groupings by showing the dependencies among these groupings.
- Profile diagram: operates at the metamodel level to show stereotypes as classes with the <<stereotype>> stereotype, and profiles as packages with the <<profile>> stereotype. The extension relation (solid line with closed, filled arrowhead) indicates what metamodel element a given stereotype is extending.

Behaviour Diagrams

Behavior diagrams emphasize what must happen in the system being modeled. Since behavior diagrams illustrate the behavior of a system, they are used extensively to describe the functionality of software systems.

- Use case diagram: describes the functionality provided by a system in terms of actors, their goals represented as use cases, and any dependencies among those use cases.
- Activity diagram: describes the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.
- state machine diagram: describes the states and state transitions of the system.

Interaction Diagrams

Interaction diagrams, a subset of behaviour diagrams, emphasize the flow of control and data among the things in the system being modeled:

- Sequence diagram: shows how objects communicate with each other in terms of a sequence of messages. Also indicates the lifespans of objects relative to those messages.
- Communication diagram: shows the interactions between objects or parts in terms of sequenced messages. They represent a combination of information taken from Class, Sequence, and Use Case Diagrams describing both the static structure and dynamic behavior of a system.
- Interaction overview diagram: provides an overview in which the nodes represent communication diagrams.
- Timing diagrams: a specific type of interaction diagram where the focus is on timing constraints.

UML Modelling Tools

To draw UML diagrams, all you need is a pencil and a piece of paper. However, for a software engineer that seems a little outdated, hence most of us will use tools. The simplest tools are simply drawing programs, like Visio or Dia. The diagrams generated this way look nice, but are not really that useful, since they do not include the code generation feature.

Hence, when deciding on a UML modelling tool (sometimes also called CASE tool)[9] you should make sure, that it allows for code generation and even better, it should also allow for reverse engineering. Combined, these two are also referred to as round-trip engineering. Any serious tool should be able to do that. Finally, UML models can be exchanged among UML tools by using the XMI interchange format, hence you should check that your tool of choice supports this.

Since the Rational Software Corporation so to say 'invented' UML, the most well-known UML modelling tool is IBM Rational Rose. Other tools include Rational Rhapsody, MagicDraw UML, StarUML, ArgoUML, Umbrello, BOUML, PowerDesigner, Visio and Dia. Some of popular development environments also offer UML modelling tools, i. e. Eclipse, NetBeans, and Visual Studio. [10]

References

1. http://www.omg.org/ Object Management Group
2. http://en.wikipedia.org/w/index.php?title=Unified_Modeling_Language&oldid=413683022 Unified Modeling Language

3. Objectory AB, known as Objectory System, was founded in 1987 by Ivar Jacobson. In 1991, It was acquired and became a subsidiary of Ericsson.
4. Wikipedia:FOLDOC (2001). Unified Modeling Language (http://foldoc.org/index.cgi?query=UML&action=Search) last updated 2002-01-03. Accessed 6 feb 2009.
5. Grady Booch, Ivar Jacobson & Jim Rumbaugh (2000) OMG Unified Modeling Language Specification (http://www.omg.org/docs/formal/00-03-01.pdf), Version 1.3 First Edition: March 2000. Retrieved 12 August 2008.
6. Satish Mishra (1997). "Visual Modeling & Unified Modeling Language (UML) : Introduction to UML" (http://www2.informatik.hu-berlin.de/~hs/Lehre/2004-WS_SWQS/20050107_Ex_UML.ppt). Rational Software Corporation. Accessed 9 Nov 2008.
7. Jon Holt Institution of Electrical Engineers (2004). *UML for Systems Engineering: Watching the Wheels* IET, 2004, ISBN 0863413544. p.58
8. *UML Superstructure Specification Version 2.2*. OMG, February 2009.
9. http://en.wikipedia.org/wiki/Computer-aided_software_engineering Computer-aided software engineering
10. http://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools List of Unified Modeling Language Tools

Labs

Lab 1a: StarUML (30 min)

We need to learn about a UML modelling tool. StarUML[1] is a free UML modelling tool that is quite powerful, it allows for forward and reverse engineering. It supports Java, C++ and C#. A small disadvantage is that it is no longer supported, hence it is limited to Java 1.4 and C# 2.0, which is very unfortunate.

Start StarUML, and select 'Empty Project' at the startup. Then in the Model Explorer add a new model by right-clicking on the 'untitled' thing. After that you can create all kinds of UML diagrams by right-clicking on the model.

More details can be found at the following StarUML Tutorial. [2]

Lab 1b: objectiF (30 min)

Another UML modelling tool, which is free for personal use is objectiF.[3] To learn how to use objectiF, take a look at their tutorial.[4]

Lab 2: Restaurant Example (30 min)

Take the restaurant example from class and create all the different UML diagrams we created in class using StarUML or your favorite UML modelling tool.

Once you are done with the class diagram, try out the 'Generate Code' feature. In StarUML you right-click with your mouse in the class diagram, and select 'Generate Code'. Look at the classes generated and compare them with your model.

Important Note: the restaurant example actually is not a very good example, because it may give you the impression that actors become objects. They don't. Actors never turn into objects, actors are always outside the system, hence the Tetris example below is much better.

Lab 3: Tetris (60 min)

Most of us know the game of Tetris.[5] [6] (However, I had students who did not know it, so in case you don't please learn about it and play it for a little before continuing with this lab). If you recall from class, when inventing UML, the Three Amigos started with something that was called Object Oriented Analysis and Object Oriented Design. So let's do it.

Object Oriented Analysis and Design

We want to program the game Tetris. But before we can start we need to analyze the game. We first need to identify 'objects' and second we must find out how they relate and talk to each other. So for this lab do the following:

- build teams of 5 to 6 students per team
- looking at the game, try to identify objects, for instance the bricks that are dropping are objects
- have each student represent an object, e.g., a brick
- what properties does that student/brick have? (color,shape,...)

- how many bricks can there be, how do they interact?
- is the player who plays the game also an object or is she an actor?
- how does the player interact with the bricks?
- what restrictions exist on the movement of the bricks, how would you implement them?
- how would you calculate a score for the game?

Game Time

Once you have identified all the objects, start playing the real game. For this one student is the 'actor', one student will write the minutes (protocol), and the other students will be objects (such as bricks). Now start the game: the actor and the different objects interact by talking to each other. The student who writes the minutes, will write down who said what to whom, i.e., everything that was said between the objects and the actor and the objects amongst themselves.

Play the game for at least one or two turns. Does your simulation work, did you forget something? Maybe you need to add another object, like a timer.

UML

We now need to make the connection between the play and UML.

- Use Case diagram: this is pretty straightforward, you identify the actor as the player, and write down all the use cases that the player can do. One question you should check, if the timer is an actor. The Use Case diagram tells you how your system interacts with its environment. Draw the Use Case diagram for Tetris.
- Activity diagram: from your game you find that there is some repetition, and some decisions have to be made at different stages in the game. This can be nicely represented with an Activity diagram. Draw a high level Activity diagram for the game.

Next we come to our minutes (protocol): this contains all the information we need to draw a Sequence diagram and finally arrive at the Collaboration diagram.

- Sequence diagram: take a look at your minutes (protocol). First, identify the objects and put them horizontally. Then go through your minutes line for line. For every line in your minutes, draw the corresponding line in your Sequence diagram. As you can see the Sequence diagram is in one-to-one correspondence with the minutes.
- Collaboration diagram: from the Sequence diagram it is easy to create the Collaboration diagram. Just do it.
- Class diagram (Association): from the Collaboration diagram, you can infer the classes and the methods the classes need. As for the attributes, some you have already identified (like the color and shape of the bricks), others you may have to think about for a little. Draw the class diagram, and for every class try to guess what attributes are needed for the class to work properly.
- Class diagram (Inheritance): if you have some more experience with object-oriented languages, you may try to identify super classes, i.e., try to use inheritance to take into account common features of objects.

If you performed this lab properly, you will have learned a lot. First, you have seen that through a simulation it is possible to identify objects, connections between the objects, hidden requirements and defects in your assumptions. Second, using a carefully written protocol of your simulation is enough to create Sequence, Collaboration and Class diagrams. So there is no magic in creating these diagrams, it is just playing a game. Although you may think this is just a funny or silly game, my advice to you: play this game for every new project you start.

References

1. http://staruml.sourceforge.net/en/ StarUML - The Open Source UML/MDA Platform
2. http://cnx.org/content/m15092/latest/ StarUML Tutorial
3. http://www.microtool.de/objectif/en/index.asp objectiF - Tool for Model-Driven Software Development with UML
4. http://www.microtool.de/mT/pdf/objectiF/01/Tutorials/JavaTutorial.pdf Developing Java Applications with UML
5. http://en.wikipedia.org/wiki/Tetris Tetris
6. http://www.percederberg.net/games/tetris/index.html Java Tetris

Questions

1. Give the names of two of the inventors of UML?

2. Who mananges the UML standard these days?
3. Draw a UseCase diagram for the game of Breakout.
4. Take a look at the following Activity diagram. Describe the flow of information in your own words.

   Questions_Activity_Diagram
5. In class we introduced the example of a restaurant, where Ralph was the customer who wanted to eat dinner. His waiter was Linus, the cook was Larry and Steve was the cashier. Ralph was ordering a hamburger and beer. Please draw an Sequence diagram, that displays the step from the ordering of the food until Ralph gets his beer and burger.
6. Consider the following class diagram. Please write Java classes that would implement this class diagram.

   Questions_Class_Diagram
7. You are supposed to write code for a money machine. Draw a UseCase diagram.
8. Turn the following Sequence diagram into a class diagram.

   Questions_Sequence_Diagram
9. What is the difference between Structure diagrams and Behavior diagrams?

Process & Methodology

Introduction

First we need to take a brief look at the big picture. The software development process is a structure imposed on the development of a software product. It is made up of a set of activities and steps with the goal to find repeatable, predictable processes that improve productivity and quality.

Software Development Activities

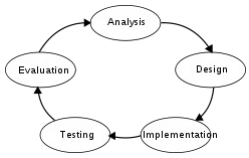
The software development process consists of a set of activities and steps, which are

- Requirements
- Specification
- Architecture
- Design
- Implementation
- Testing
- Deployment
- Maintenance

Model of the Systems Development Life Cycle

Planning

The important task in creating a software product is extracting the requirements or requirements analysis. Customers typically have an abstract idea of what they want as an end result, but not what software should do. Incomplete, ambiguous, or even contradictory requirements are recognized by skilled and experienced software engineers at this point. Frequently demonstrating live code may help reduce the risk that the requirements are incorrect.



Model of the Systems Development Life Cycle

Once the general requirements are gathered from the client, an analysis of the scope of the development should be determined and clearly stated. This is often called a scope document.

Certain functionality may be out of scope of the project as a function of cost or as a result of unclear requirements at the start of development. If the development is done externally, this document can be considered a legal document so that if there are ever disputes, any ambiguity of what was promised to the client can be clarified.

Implementation, Testing and Documenting

Implementation is the part of the process where software engineers actually program the code for the project.

Software testing is an integral and important part of the software development process. This part of the process ensures that defects are recognized as early as possible.

Documenting the internal design of software for the purpose of future maintenance and enhancement is done throughout development. This may also include the writing of an API, be it external or internal. It is very important to document everything in the project.

Deployment and Maintenance

Deployment starts after the code is appropriately tested, is approved for release and sold or otherwise distributed into a production environment.

Software Training and Support is important and a lot of developers fail to realize that. It would not matter how much time and planning a development team puts into creating software if nobody in an organization ends up using it. People are often resistant to change and avoid venturing into an unfamiliar area, so as a part of the deployment phase, it is very important to have training classes for new clients of your software.

Maintaining and enhancing software to cope with newly discovered problems or new requirements can take far more time than the initial development of the software. It may be necessary to add code that does not fit the original design to correct an unforeseen problem or it may be that a customer is requesting more functionality and code can be added to accommodate their requests. If the labor cost of the maintenance phase exceeds 25% of the prior-phases' labor cost, then it is likely that the overall quality of at least one prior phase is poor. In that case, management should consider the option of rebuilding the system (or portions) before maintenance cost is out of control.

References

=

Heading text

=====

External links

- Don't Write Another Process (http://www.methodsandtools.com/archive/archive.php?id=16)
- No Silver Bullet: Essence and Accidents of Software Engineering (http://virtualschool.edu/mon/SoftwareEngineering/BrooksNoSilverBullet.html)", 1986
- Gerhard Fischer, "The Software Technology of the 21st Century: From Software Reuse to Collaborative Software Design" (http://l3d.cs.colorado.edu/~gerhard/papers/isfst2001.pdf), 2001
- Lydia Ash: *The Web Testing Companion: The Insider's Guide to Efficient and Effective Tests*, Wiley, May 2, 2003. ISBN 0-471-43021-8
- SaaSSDLC.com (http://SaaSSDLC.com/) — Software as a Service Systems Development Life Cycle Project
- Software development life cycle (SDLC) [visual image], *software development life cycle* (http://www.notetech.com/images/software_lifecycle.jpg)
- Selecting an SDLC (http://www.gem-up.com/PDF/SK903V0-WP-ChoosingSDLC.pdf)", 2009
- Heraprocess.org (http://www.heraprocess.org/) — Hera is a light process solution for managing web projects

book let

Header text	Header text	Header text
Example	Example	Example
Example	Example	Example
Example	Example	Example

Methodology

A **software development methodology** or **system development methodology** in software engineering is a framework that is used to structure, plan, and control the process of developing an information system.

History

The software development methodology framework didn't emerge until the 1960s. According to Elliott (2004) the systems development life cycle (SDLC) can be considered to be the oldest formalized methodology framework for building information systems. The main idea of the SDLC has been "to pursue the development of information systems in a very deliberate, structured and methodical way, requiring each stage of the life cycle from inception

of the idea to delivery of the final system, to be carried out in rigidly and sequentially".[1] within the context of the framework being applied. The main target of this methodology framework in the 1960s was "to develop large scale functional business systems in an age of large scale business conglomerates. Information systems activities revolved around heavy data processing and number crunching routines".[1]

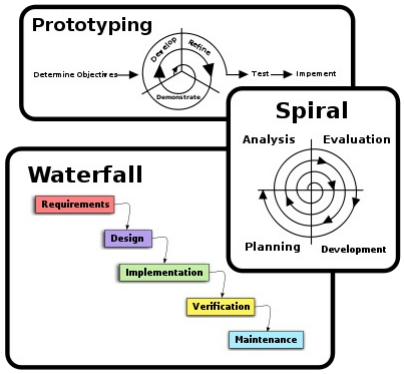
As a noun

As a noun, a software development methodology is a framework that is used to structure, plan, and control the process of developing an information system - this includes the pre-definition of specific deliverables and artifacts that are created and completed by a project team to develop or maintain an application.[2]

A wide variety of such frameworks have evolved over the years, each with its own recognized strengths and weaknesses. One software development methodology framework is not necessarily suitable for use by all projects. Each of the available methodology frameworks are best suited to specific kinds of projects, based on various technical, organizational, project and team considerations.[2]

These software development frameworks are often bound to some kind of organization, which further develops, supports the use, and promotes the methodology framework. The methodology framework is often defined in some kind of formal documentation. Specific software development methodology frameworks (noun) include

- Rational Unified Process (RUP, IBM) since 1998.
- Agile Unified Process (AUP) since 2005 by Scott Ambler



The three basic approaches applied to software development methodology frameworks.

As a verb

As a verb, the software development methodology is an approach used by organizations and project teams to apply the software development methodology framework (noun). Specific software development methodologies (verb) include:

1970s

- Structured programming since 1969
- Cap Gemini SDM, originally from PANDATA, the first English translation was published in 1974. SDM stands for System Development Methodology

1980s

- Structured Systems Analysis and Design Methodology (SSADM) from 1980 onwards
- Information Requirement Analysis/Soft systems methodology

1990s

- Object-oriented programming (OOP) has been developed since the early 1960s, and developed as a dominant programming approach during the mid-1990s
- Rapid application development (RAD) since 1991
- Scrum, since the late 1990s
- Team software process developed by Watts Humphrey at the SEI
- Extreme Programming since 1999

Verb approaches

Every software development methodology framework acts as a basis for applying specific approaches to develop and maintain software. Several software development approaches have been used since the origin of information technology. These are:[2]

- Waterfall: a linear framework
- Prototyping: an iterative framework
- Incremental: a combined linear-iterative framework
- Spiral: a combined linear-iterative framework
- Rapid application development (RAD): an iterative framework
- Extreme Programming

Waterfall development

The Waterfall model is a sequential development approach, in which development is seen as flowing steadily downwards (like a waterfall) through the phases of requirements analysis, design, implementation, testing (validation), integration, and maintenance. The first formal description of the method is often cited as an article published by Winston W. Royce[3] in 1970 although Royce did not use the term "waterfall" in this article.

The basic principles are:[2]

- Project is divided into sequential phases, with some overlap and splashback acceptable between phases.
- Emphasis is on planning, time schedules, target dates, budgets and implementation of an entire system at one time.
- Tight control is maintained over the life of the project via extensive written documentation, formal reviews, and approval/signoff by the user and information technology management occurring at the end of most phases before beginning the next phase.

Prototyping

Software prototyping, is the development approach of activities during software development, the creation of prototypes, i.e., incomplete versions of the software program being developed.

The basic principles are:[2]

- Not a standalone, complete development methodology, but rather an approach to handling selected parts of a larger, more traditional development methodology (i.e. incremental, spiral, or rapid application development (RAD)).
- Attempts to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.
- User is involved throughout the development process, which increases the likelihood of user acceptance of the final implementation.
- Small-scale mock-ups of the system are developed following an iterative modification process until the prototype evolves to meet the users' requirements.
- While most prototypes are developed with the expectation that they will be discarded, it is possible in some cases to evolve from prototype to working system.
- A basic understanding of the fundamental business problem is necessary to avoid solving the wrong problem.

Incremental development

Various methods are acceptable for combining linear and iterative systems development methodologies, with the primary objective of each being to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.

The basic principles are:[2]

- A series of mini-Waterfalls are performed, where all phases of the Waterfall are completed for a small part of a system, before proceeding to the next increment, or
- Overall requirements are defined before proceeding to evolutionary, mini-Waterfall development of individual increments of a system, or
- The initial software concept, requirements analysis, and design of architecture and system core are defined via Waterfall, followed by iterative Prototyping, which culminates in installing the final prototype, a working system.

Spiral development

The spiral model is a software development process combining elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts.

The basic principles are:[2]

- Focus is on risk assessment and on minimizing project risk by breaking a project into smaller segments and providing more ease-of-change during the development process, as well as providing the opportunity to evaluate risks and weigh consideration of project continuation throughout the life cycle.
- "Each cycle involves a progression through the same sequence of steps, for each part of the product and for

each of its levels of elaboration, from an overall concept-of-operation document down to the coding of each individual program."[4]

- Each trip around the spiral traverses four basic quadrants: (1) determine objectives, alternatives, and constraints of the iteration; (2) evaluate alternatives; Identify and resolve risks; (3) develop and verify deliverables from the iteration; and (4) plan the next iteration.[4][5]
- Begin each cycle with an identification of stakeholders and their win conditions, and end each cycle with review and commitment.[6]

The spiral model.

Rapid application development

Rapid application development (RAD) is a software development methodology, which involves iterative development and the construction of prototypes. Rapid application development is a term originally used to describe a software development process introduced by James Martin in 1991.

The basic principles are:[2]

- Key objective is for fast development and delivery of a high quality system at a relatively low investment cost.
- Attempts to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.
- Aims to produce high quality systems quickly, primarily via iterative Prototyping (at any stage of development), active user involvement, and computerized development tools. These tools may include Graphical User Interface (GUI) builders, Computer Aided Software Engineering (CASE) tools, Database Management Systems (DBMS), fourth-generation programming languages, code generators, and object-oriented techniques.
- Key emphasis is on fulfilling the business need, while technological or engineering excellence is of lesser importance.
- Project control involves prioritizing development and defining delivery deadlines or “timeboxes”. If the project starts to slip, emphasis is on reducing requirements to fit the timebox, not in increasing the deadline.
- Generally includes joint application design (JAD), where users are intensely involved in system design, via consensus building in either structured workshops, or electronically facilitated interaction.
- Active user involvement is imperative.
- Iteratively produces production software, as opposed to a throwaway prototype.
- Produces documentation necessary to facilitate future development and maintenance.
- Standard systems analysis and design methods can be fitted into this framework.

Other practices

Other methodology practices include:

- Object-oriented development methodologies, such as Grady Booch's object-oriented design (OOD), also known as object-oriented analysis and design (OOAD). The Booch model includes six diagrams: class, object, state transition, interaction, module, and process.[7]
- Top-down programming: evolved in the 1970s by IBM researcher Harlan Mills (and Niklaus Wirth) in developed structured programming.
- Unified Process (UP) is an iterative software development methodology framework, based on Unified Modeling Language (UML). UP organizes the development of software into four phases, each consisting of one or more executable iterations of the software at that stage of development: inception, elaboration, construction, and guidelines. Many tools and products exist to facilitate UP implementation. One of the more popular versions of UP is the Rational Unified Process (RUP).
- Agile software development refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve via collaboration between self-organizing cross-functional teams. The term was coined in the year 2001 when the Agile Manifesto was formulated.
- Integrated software development refers to a deliverable based software development framework using the three primary IT (project management, software development, software testing) life cycles that can be leveraged using multiple (iterative, waterfall, spiral, agile) software development approaches, where requirements and

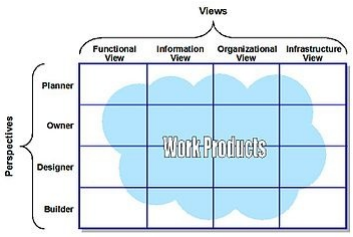
solutions evolve via collaboration between self-organizing cross-functional teams.

Subtopics

View model

A view model is framework which provides the viewpoints on the system and its environment, to be used in the software development process. It is a graphical representation of the underlying semantics of a view.

The purpose of viewpoints and views is to enable human engineers to comprehend very complex systems, and to organize the elements of the problem and the solution around domains of expertise. In the engineering of physically intensive systems, viewpoints often correspond to capabilities and responsibilities within the engineering organization.[8]



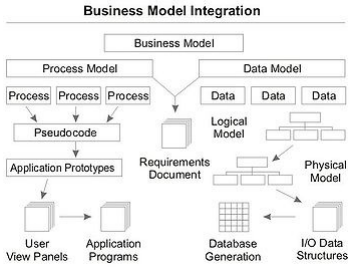
The TEAF Matrix of Views and Perspectives.

Most complex system specifications are so extensive that no one individual can fully comprehend all aspects of the specifications. Furthermore, we all have different interests in a given system and different reasons for examining the system's specifications. A business executive will ask different questions of a system make-up than would a system implementer. The concept of viewpoints framework, therefore, is to provide separate viewpoints into the specification of a given complex system. These viewpoints each satisfy an audience with interest in some set of aspects of the system. Associated with each viewpoint is a viewpoint language that optimizes the vocabulary and presentation for the audience of that viewpoint.

Business process and data modelling

Graphical representation of the current state of information provides a very effective means for presenting information to both users and system developers.

- A business model illustrates the functions associated with the business process being modeled and the organizations that perform these functions. By depicting activities and information flows, a foundation is created to visualize, define, understand, and validate the nature of a process.
- A data model provides the details of information to be stored, and is of primary use when the final product is the generation of computer software code for an application or the preparation of a functional specification to aid a computer software make-or-buy decision. See the figure on the right for an example of the interaction between business process and data models.[9]



example of the interaction between business process and data models.[9]

Usually, a model is created after conducting an interview, referred to as business analysis. The interview consists of a facilitator asking a series of questions designed to extract required information that describes a process. The interviewer is called a facilitator to emphasize that it is the participants who provide the information. The facilitator should have some knowledge of the process of interest, but this is not as important as having a structured methodology by which the questions are asked of the process expert. The methodology is important because usually a team of facilitators is collecting information across the facility and the results of the information from all the interviewers must fit together once completed.[9]

The models are developed as defining either the current state of the process, in which case the final product is called the "as-is" snapshot model, or a collection of ideas of what the process should contain, resulting in a "what-can-be" model. Generation of process and data models can be used to determine if the existing processes and information systems are sound and only need minor modifications or enhancements, or if re-engineering is required as a corrective action. The creation of business models is more than a way to view or automate your information process. Analysis can be used to fundamentally reshape the way your business or organization conducts its operations.[9]

Computer-aided software engineering

Computer-aided software engineering (CASE), in the field software engineering is the scientific application of a set of tools and methods to a software which results in high-quality, defect-free, and maintainable software products.[10] It also refers to methods for the development of information systems together with automated tools that can be used in the software development process.[11] The term "computer-aided software engineering" (CASE) can refer to the software used for the automated development of systems software, i.e., computer code. The CASE functions include analysis, design, and programming. CASE tools automate methods for designing, documenting, and producing structured computer code in the desired programming language.[12]

Two key ideas of Computer-aided Software System Engineering (CASE) are:[13]

- Foster computer assistance in software development and or software maintenance processes, and

- An engineering approach to software development and or maintenance.

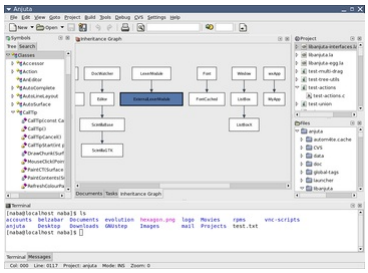
Typical CASE tools exist for configuration management, data modeling, model transformation, refactoring, source code generation, and Unified Modeling Language.

Integrated development environment

An integrated development environment (IDE) also known as *integrated design environment* or *integrated debugging environment* is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a:

- source code editor,
- compiler and/or interpreter,
- build automation tools, and
- debugger (usually).

IDEs are designed to maximize programmer productivity by providing tight-knit components with similar user interfaces. Typically an IDE is dedicated to a specific programming language, so as to provide a feature set which most closely matches the programming paradigms of the language.



Anjuta, a C and C++ IDE for the GNOME environment

Modeling language

A modeling language is any artificial language that can be used to express information or knowledge or systems in a structure that is defined by a consistent set of rules. The rules are used for interpretation of the meaning of components in the structure. A modeling language can be graphical or textual.[14] Graphical modeling languages use a diagram techniques with named symbols that represent concepts and lines that connect the symbols and that represent relationships and various other graphical annotation to represent constraints. Textual modeling languages typically use standardised keywords accompanied by parameters to make computer-interpretable expressions.

Example of graphical modelling languages in the field of software engineering are:

- Business Process Modeling Notation (BPMN, and the XML form BPML) is an example of a process modeling language.
- EXPRESS and EXPRESS-G (ISO 10303-11) is an international standard general-purpose data modeling language.
- Extended Enterprise Modeling Language (EEML) is commonly used for business process modeling across layers.
- Flowchart is a schematic representation of an algorithm or a stepwise process,
- Fundamental Modeling Concepts (FMC) modeling language for software-intensive systems.
- IDEF is a family of modeling languages, the most notable of which include IDEF0 for functional modeling, IDEF1X for information modeling, and IDEF5 for modeling ontologies.
- LePUS3 is an object-oriented visual Design Description Language and a formal specification language that is suitable primarily for modelling large object-oriented (Java, C++, C#) programs and design patterns.
- Specification and Description Language(SDL) is a specification language targeted at the unambiguous specification and description of the behaviour of reactive and distributed systems.
- Unified Modeling Language (UML) is a general-purpose modeling language that is an industry standard for specifying software-intensive systems. UML 2.0, the current version, supports thirteen different diagram techniques, and has widespread tool support.

Not all modeling languages are executable, and for those that are, using them doesn't necessarily mean that programmers are no longer needed. On the contrary, executable modeling languages are intended to amplify the productivity of skilled programmers, so that they can address more difficult problems, such as parallel computing and distributed systems.

Programming paradigm

A programming paradigm is a fundamental style of computer programming, in contrast to a software engineering methodology, which is a style of solving specific software engineering problems. Paradigms differ in the concepts and abstractions used to represent the elements of a program (such as objects, functions, variables, constraints...) and the steps that compose a computation (assignation, evaluation, continuations, data flows...).

A programming language can support multiple paradigms. For example programs written in C++ or Object Pascal can be purely procedural, or purely object-oriented, or contain elements of both paradigms. Software designers and programmers decide how to use those paradigm elements. In object-oriented programming, programmers can think of a program as a collection of interacting objects, while in functional programming a program can be thought of as a sequence of stateless function evaluations. When programming computers or systems with many processors, process-oriented programming allows programmers to think about applications as sets of concurrent processes acting upon logically shared data structures.

Just as different groups in software engineering advocate different *methodologies*, different programming languages advocate different *programming paradigms*. Some languages are designed to support one paradigm (Smalltalk supports object-oriented programming, Haskell supports functional programming), while other programming languages support multiple paradigms (such as Object Pascal, C++, C#, Visual Basic, Common Lisp, Scheme, Python, Ruby, and Oz).

Many programming paradigms are as well known for what methods they *forbid* as for what they enable. For instance, pure functional programming forbids using side-effects; structured programming forbids using goto statements. Partly for this reason, new paradigms are often regarded as doctrinaire or overly rigid by those accustomed to earlier styles.[*citation needed*] Avoiding certain methods can make it easier to prove theorems about a program's correctness, or simply to understand its behavior.

Software framework

A software framework is a re-usable design for a software system or subsystem. A software framework may include support programs, code libraries, a scripting language, or other software to help develop and *glue together* the different components of a software project. Various parts of the framework may be exposed via an API.

Software development process

A software development process is a framework imposed on the development of a software product. Synonyms include software life cycle and *software process*. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process.

A largely growing body of software development organizations implement process methodologies. Many of them are in the defense industry, which in the U.S. requires a rating based on 'process models' to obtain contracts. The international standard describing the method to select, implement and monitor the life cycle for software is ISO 12207.

A decades-long goal has been to find repeatable, predictable processes that improve productivity and quality. Some try to systematize or formalize the seemingly unruly task of writing software. Others apply project management methods to writing software. Without project management, software projects can easily be delivered late or over budget. With large numbers of software projects not meeting their expectations in terms of functionality, cost, or delivery schedule, effective project management appears to be lacking.

See also

Lists

- List of software engineering topics
- List of software development philosophies

Related topics

- Domain-specific modeling
- Lightweight methodology
- Object modeling language
- Structured programming
- Integrated IT Methodology

References

1. Geoffrey Elliott (2004) *Global Business Information Technology: an integrated systems approach*. Pearson Education. p.87.
2. Centers for Medicare & Medicaid Services (CMS) Office of Information Service (2008). *Selecting a development approach (http://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf)*. Webarticle. United States Department of Health and Human Services (HHS). Revalidated: March 27, 2008. Retrieved 15 July 2015.
3. Wasserfallmodell > Entstehungskontext (http://cartoon.iguw.tuwien.ac.at/fit/fit01/wasserfall/entstehung.html), Markus Rerych, Institut für Gestaltungs- und Wirkungsforschung, TU-Wien. Accessed on line November 28, 2007.
4. Barry Boehm (1996., "A Spiral Model of Software Development and Enhancement (http://doi.acm.org/10.1145/12944.12948)". In: *ACM SIGSOFT Software Engineering Notes* (ACM) 11(4):14-24, August 1986
5. Richard H. Thayer, Barry W. Boehm (1986). *Tutorial: software engineering project management*. Computer Society Press of the IEEE. p.130
6. Barry W. Boehm (2000). *Software cost estimation with Cocomo II: Volume 1*.
7. Georges Gauthier Merx & Ronald J. Norman (2006). *Unified Software Engineering with Java*. p.201.

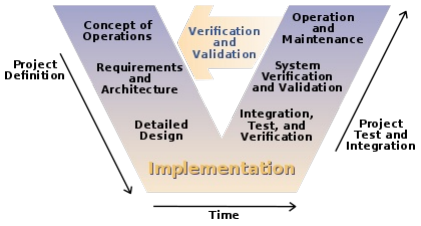
8. Edward J. Barkmeyer ea (2003). *Concepts for Automating Systems Integration* (http://www.mel.nist.gov/msidlibrary/doc/AMIS-Concepts.pdf) NIST 2003.
9. Paul R. Smith & Richard Sarfaty (1993). Creating a strategic plan for configuration management using Computer Aided Software Engineering (CASE) tools. (http://www.osti.gov/energycitations/servlets/purl/10160331-YhIRrY/) Paper For 1993 National DOE/Contractors and Facilities CAD/CAE User's Group.
10. Kuhn, D.L (1989). "Selecting and effectively using a computer aided software engineering tool". Annual Westinghouse computer symposium; 6-7 Nov 1989; Pittsburgh, PA (USA); DOE Project.
11. P. Loucopoulos and V. Karakostas (1995). *System Requirements Engineering*. McGraw-Hill.
12. CASE (http://www.its.bldrdoc.gov/projects/devglossary/_case.html) definition In: *Telecom Glossary 2000 (http://www.its.bldrdoc.gov/projects/devglossary/). Retrieved 26 Oct 2008.*
13. K. Robinson (1992). *Putting the Software Engineering into CASE*. New York : John Wiley and Sons Inc.
14. Xiao He (2007). "A metamodel for the notation of graphical modeling languages". In: *Computer Software and Applications Conference, 2007. COMPSAC 2007 - Vol. 1. 31st Annual International*, Volume 1, Issue , 24–27 July 2007, pp 219-224.

External links

- Selecting a development approach (http://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf) at cms.gov.
- Software Methodologies Book Reviews (http://www.techbookreport.com/SoftwareIndex.html) An extensive set of book reviews related to software methodologies and processes

V-Model

The **V-model** represents a software development process (also applicable to hardware development) which may be considered an extension of the waterfall model. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape. The V-Model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing. The horizontal and vertical axes represents time or project completeness (left-to-right) and level of abstraction (coarsest-grain abstraction uppermost), respectively.



The V-model of the Systems Engineering Process.[1]

Verification Phases

Requirements analysis

In the Requirements analysis phase, the requirements of the proposed system are collected by analyzing the needs of the user(s). This phase is concerned about establishing what the ideal system has to perform. However it does not determine how the software will be designed or built. Usually, the users are interviewed and a document called the user requirements document is generated.

The user requirements document will typically describe the system's functional, interface, performance, data, security, etc requirements as expected by the user. It is used by business analysts to communicate their understanding of the system to the users. The users carefully review this document as this document would serve as the guideline for the system designers in the system design phase. The user acceptance tests are designed in this phase. See also Functional requirements. this is parallel processing

There are different methods for gathering requirements of both soft and hard methodologies including; interviews, questionnaires, document analysis, observation, throw-away prototypes, use cases and status and dynamic views with users.

System Design

Systems design is the phase where system engineers analyze and understand the business of the proposed system by studying the user requirements document. They figure out possibilities and techniques by which the user requirements can be implemented. If any of the requirements are not feasible, the user is informed of the issue. A resolution is found and the user requirement document is edited accordingly.

The software specification document which serves as a blueprint for the development phase is generated. This document contains the general system organization, menu structures, data structures etc. It may also hold example business scenarios, sample windows, reports for the better understanding. Other technical documentation like entity diagrams, data dictionary will also be produced in this phase. The documents for system testing are prepared in this phase. V model is also similar with waterfall model.

Architecture Design

The phase of the design of computer architecture and software architecture can also be referred to as high-level design. The baseline in selecting the architecture is that it should realize all which typically consists of the list of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, technology details etc. The integration testing design is carried out in the particular phase.

Module Design

The module design phase can also be referred to as low-level design. The designed system is broken up into smaller units or modules and each of them is explained so that the programmer can start coding directly. The low level design document or program specifications will contain a detailed functional logic of the module, in pseudocode:

- database tables, with all elements, including their type and size
- all interface details with complete API references
- all dependency issues
- error message listings
- complete input and outputs for a module.

The unit test design is developed in this stage.

Validation Phases

Unit Testing

In computer programming, unit testing is a method by which individual units of source code are tested to determine if they are fit for use. A unit is the smallest testable part of an application. In procedural programming a unit may be an individual function or procedure. Unit tests are created by programmers or occasionally by white box testers. The purpose is to verify the internal logic code by testing every possible branch within the function, also known as test coverage. Static analysis tools are used to facilitate in this process, where variations of input data are passed to the function to test every possible case of execution.

Integration Testing

In integration testing the separate modules will be tested together to expose faults in the interfaces and in the interaction between integrated components. Testing is usually black box as the code is not directly checked for errors.

System Testing

System testing will compare the system specifications against the actual system.After the integration test is completed, the next test level is the system test. System testing checks if the integrated product meets the specified requirements. Why is this still necessary after the component and integration tests? The reasons for this are as follows:

Reasons for system test

1. In the lower test levels, the testing was done against technical specifications, i.e., from the technical perspective of the software producer. The system test, though, looks at the system from the perspective of the customer and the future user. The testers validate whether the requirements are completely and appropriately met.
    - Example: The customer (who has ordered and paid for the system) and the user (who uses the system) can be different groups of people or organizations with their own specific interests and requirements of the system.
2. Many functions and system characteristics result from the interaction of all system components, consequently, they are only visible on the level of the entire system and can only be observed and tested there.

User Acceptance Testing

Acceptance testing is the phase of testing used to determine whether a system satisfies the requirements specified in the requirements analysis phase. The acceptance test design is derived from the requirements document. The acceptance test phase is the phase used by the customer to determine whether to accept the system or not.

Acceptance testing helps

- to determine whether a system satisfies its acceptance criteria or not.
- to enable the customer to determine whether to accept the system or not.
- to test the software in the "real world" by the intended audience.

Purpose of acceptance testing:

- to verify the system or changes according to the original needs.

Procedures

1. Define the acceptance criteria:
    - Functionality requirements.
    - Performance requirements.

- Interface quality requirements.
- Overall software quality requirements.

2. Develop an acceptance plan:
   - Project description.
   - User responsibilities.
   - Acceptance description.
   - Execute the acceptance test plan.

References

1. *Clarus Concept of Operations.* (http://www.itsdocs.fhwa.dot.gov/jpodocs/repts_te/14158.htm) Publication No. FHWA-JPO-05-072, Federal Highway Administration (FHWA), 2005

Further reading

- Roger S. Pressman:*Software Engineering: A Practitioner's Approach*, The McGraw-Hill Companies, ISBN 007301933X
- Mark Hoffman & Ted Beaumont: *Application Development: Managing the Project Life Cycle*, Mc Press, ISBN 1883884454
- Boris Beizer: *Software Testing Techniques.* Second Edition, International Thomson Computer Press, 1990, ISBN 1-85032-880-3

External links

Agile Model

Agile software development is a group of software development methodologies based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. The *Agile Manifesto*[1] introduced the term in 2001.

History

Predecessors

Jeff Sutherland, one of the developers of the Scrum agile software development process

Incremental software development methods have been traced back to 1957.[2] In 1974, a paper by E. A. Edmonds introduced an adaptive software development process.[3]

So-called "lightweight" software development methods evolved in the mid-1990s as a reaction against "heavyweight" methods, which were characterized by their critics as a heavily regulated, regimented, micromanaged, waterfall model of development. Proponents of lightweight methods (and now "agile" methods) contend that they are a return to development practices from early in the history of software development.[2]

Early implementations of lightweight methods include Scrum (1995), Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM) (1995). These are now typically referred to as agile methodologies, after the Agile Manifesto published in 2001.[4]

Agile Manifesto

In February 2001, 17 software developers[5] met at a ski resort in Snowbird, Utah, to discuss lightweight development methods. They published the "Manifesto for Agile Software Development"[1] to define the approach now known as agile software development. Some of the manifesto's authors formed the Agile Alliance, a nonprofit organization that promotes software development according to the manifesto's principles.

Agile Manifesto reads, in its entirety, as follows:[1]

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> **Individuals and interactions** over processes and tools
> **Working software** over comprehensive documentation
> **Customer collaboration** over contract negotiation
> **Responding to change** over following a plan

> That is, while there is value in the items on the right, we value the items on the left more.

Twelve principles underlie the Agile Manifesto, including:[6]

- Customer satisfaction by rapid delivery of useful software
- Welcome changing requirements, even late in development
- Working software is delivered frequently (weeks rather than months)
- Working software is the principal measure of progress
- Sustainable development, able to maintain a constant pace
- Close, daily co-operation between business people and developers
- Face-to-face conversation is the best form of communication (co-location)
- Projects are built around motivated individuals, who should be trusted
- Continuous attention to technical excellence and good design
- Simplicity
- Self-organizing teams
- Regular adaptation to changing circumstances

In 2005, a group headed by Alistair Cockburn and Jim Highsmith wrote an addendum of project management principles, the Declaration of Interdependence,[7] to guide software project management according to agile development methods.

Characteristics

There are many specific agile development methods. Most promote development, teamwork, collaboration, and process adaptability throughout the life-cycle of the project.

Pair programming, an XP development technique used by agile

Agile methods break tasks into small increments with minimal planning, and do not directly involve long-term planning. Iterations are short time frames (timeboxes) that typically last from one to four weeks. Each iteration involves a team working through a full software development cycle including planning, requirements analysis, design, coding, unit testing, and acceptance testing when a working product is demonstrated to stakeholders. This minimizes overall risk and allows the project to adapt to changes quickly. Stakeholders produce documentation as required. An iteration may not add enough functionality to warrant a market release, but the goal is to have an available release (with minimal bugs) at the end of each iteration.[8] Multiple iterations may be required to release a product or new features.

Team composition in an agile project is usually cross-functional and self-organizing without consideration for any existing corporate hierarchy or the corporate roles of team members. Team members normally take responsibility for tasks that deliver the functionality an iteration requires. They decide individually how to meet an iteration's requirements.

Agile methods emphasize face-to-face communication over written documents when the team is all in the same location. Most agile teams work in a single open office (called a bullpen), which facilitates such communication. Team size is typically small (5-9 people) to simplify team communication and team collaboration. Larger development efforts may be delivered by multiple teams working toward a common goal or on different parts of an effort. This may require a co-ordination of priorities across teams. When a team works in different locations, they maintain daily contact through videoconferencing, voice, e-mail, etc.

No matter what development disciplines are required, each agile team will contain a customer representative. This person is appointed by stakeholders to act on their behalf and makes a personal commitment to being available for developers to answer mid-iteration problem-domain questions. At the end of each iteration, stakeholders and the customer representative review progress and re-evaluate priorities with a view to optimizing the return on investment (ROI) and ensuring alignment with customer needs and company goals.

Most agile implementations use a routine and formal daily face-to-face communication among team members. This specifically includes the customer representative and any interested stakeholders as observers. In a brief session, team members report to each other what they did the previous day, what they intend to do today, and what their roadblocks are. This face-to-face communication exposes problems as they arise.

Agile development emphasizes working software as the primary measure of progress. This, combined with the preference for face-to-face communication, produces less written documentation than other methods. The agile method encourages stakeholders to prioritize wants with other iteration outcomes based exclusively on business value perceived at the beginning of the iteration.

Specific tools and techniques such as continuous integration, automated or xUnit test, pair programming, test driven development, design patterns, domain-driven design, code refactoring and other techniques are often used to improve quality and enhance project agility.

Comparison with other methods

Agile methods are sometimes characterized as being at the opposite end of the spectrum from "plan-driven" or "disciplined" methods; agile teams may, however, employ highly disciplined formal methods.[9] A more accurate distinction is that methods exist on a continuum from "adaptive" to "predictive".[10] Agile methods lie on the "adaptive" side of this continuum. Adaptive methods focus on adapting quickly to changing realities. When the needs of a project change, an adaptive team changes as well. An adaptive team will have difficulty describing exactly what will happen in the future. The further away a date is, the more vague an adaptive method will be about what will happen on that date. An adaptive team cannot report exactly what tasks are being done next week, but only which features are planned for next month. When asked about a release six months from now, an adaptive team may only be able to report the mission statement for the release, or a statement of expected value vs. cost.

Predictive methods, in contrast, focus on planning the future in detail. A predictive team can report exactly what features and tasks are planned for the entire length of the development process. Predictive teams have difficulty changing direction. The plan is typically optimized for the original destination and changing direction can require completed work to be started over. Predictive teams will often institute a change control board to ensure that only the most valuable changes are considered.

Formal methods, in contrast to adaptive and predictive methods, focus on computer science theory with a wide array of types of provers. A formal method attempts to prove the absence of errors with some level of determinism. Some formal methods are based on model checking and provide counter examples for code that cannot be proven. Generally, mathematical models (often supported through special languages see SPIN model checker) map to assertions about requirements. Formal methods are dependent on a tool driven approach, and may be combined with other development approaches. Some provers do not easily scale. Like agile methods, manifestos relevant to high integrity software have been proposed in Crosstalk (http://elsmar.com/pdf_files/A%20Manifesto%20for%20High-Integrity%20Software.pdf).

Agile methods have much in common with the "Rapid Application Development" techniques from the 1980/90s as espoused by James Martin and others.

Agile methods

Well-known agile software development methods include:

- Agile Modeling
- Agile Unified Process (AUP)
- Dynamic Systems Development Method (DSDM)
- Essential Unified Process (EssUP)
- Extreme Programming (XP)
- Feature Driven Development (FDD)
- Open Unified Process (OpenUP)
- Scrum
- Velocity tracking

Method tailoring

In the literature, different terms refer to the notion of method adaptation, including 'method tailoring', 'method fragment adaptation' and 'situational method engineering'. Method tailoring is defined as:

> A process or capability in which human agents through responsive changes in, and dynamic interplays between contexts, intentions, and method fragments determine a system development approach for a specific project situation.[11]

Potentially, almost all agile methods are suitable for method tailoring. Even the DSDM method is being used for this purpose and has been successfully tailored in a CMM context.[12] Situation-appropriateness can be considered as a distinguishing characteristic between agile methods and traditional software development methods, with the latter being relatively much more rigid and prescriptive. The practical implication is that agile methods allow project teams to adapt working practices according to the needs of individual projects. Practices are concrete activities and products that are part of a method framework. At a more extreme level, the philosophy behind the method, consisting of a number of principles, could be adapted (Aydin, 2004).[11]

Extreme Programming (XP) makes the need for method adaptation explicit. One of the fundamental ideas of XP is that no one process fits every project, but rather that practices should be tailored to the needs of individual projects. Partial adoption of XP practices, as suggested by Beck, has been reported on several occasions.[13] A tailoring practice is proposed by Mehdi Mirakhorli (http://portal.acm.org/citation.cfm?id=1370143.1370149&coll=ACM&dl=ACM&CFID=69442744&CFTOKEN=96226775,) which provides sufficient roadmap and guideline for adapting all the practices. RDP Practice is designed for customizing XP. This practice, first proposed as a long research paper in the APSO workshop at the ICSE 2008 conference, is currently the only proposed and applicable method for customizing XP. Although it is specifically a solution for XP, this practice has the capability of extending to other methodologies. At first glance, this practice seems to be in the category of static method adaptation but experiences with RDP Practice says that it can be treated like dynamic method adaptation. The distinction between static method adaptation and dynamic method adaptation is subtle.[14] The key assumption behind static method adaptation is that the project context is given at the start of a project and remains fixed during project execution. The result is a static definition of the project context. Given such a definition, route maps can be used in order to determine which structured method fragments should be used for that particular project, based on predefined sets of criteria. Dynamic method adaptation, in contrast, assumes that projects are situated in an emergent context. An emergent context implies that a project has to deal with emergent factors

that affect relevant conditions but are not predictable. This also means that a project context is not fixed, but changing during project execution. In such a case prescriptive route maps are not appropriate. The practical implication of dynamic method adaptation is that project managers often have to modify structured fragments or even innovate new fragments, during the execution of a project (Aydin et al., 2005).[14]

Measuring agility

While agility can be seen as a means to an end, a number of approaches have been proposed to quantify agility. Agility Index Measurements (AIM)[15] score projects against a number of agility factors to achieve a total. The similarly-named Agility Measurement Index,[16] scores developments against five dimensions of a software project (duration, risk, novelty, effort, and interaction). Other techniques are based on measurable goals.[17] Another study using fuzzy mathematics[18] has suggested that project velocity can be used as a metric of agility. There are agile self assessments to determine whether a team is using agile practices (Nokia test,[19] Karlskrona test,[20] 42 points test[21]).

While such approaches have been proposed to measure agility, the practical application of such metrics has yet to be seen.

Experience and reception

One of the early studies reporting gains in quality, productivity, and business satisfaction by using Agile methods was a survey conducted by Shine Technologies from November 2002 to January 2003.[22] A similar survey conducted in 2006 by Scott Ambler, the Practice Leader for Agile Development with IBM Rational's Methods Group reported similar benefits.[23] In a survey conducted by VersionOne in 2008, 55% of respondents answered that Agile methods had been successful in 90-100% of cases.[24] Others claim that agile development methods are still too young to require extensive academic proof of their success.[25]

Suitability

Large-scale agile software development remains an active research area.[26][27]

Agile development has been widely documented (see Experience Reports, below, as well as Beck[28] pg. 157, and Boehm and Turner[29]) as working well for small (<10 developers) co-located teams.

Some things that may negatively impact the success of an agile project are:

- Large-scale development efforts (>20 developers), though scaling strategies[27] and evidence of some large projects[30] have been described.
- Distributed development efforts (non-colocated teams). Strategies have been described in *Bridging the Distance*[31] and *Using an Agile Software Process with Offshore Development*[32]
- Forcing an agile process on a development team[33]
- Mission-critical systems where failure is not an option at any cost (e.g. software for surgical procedures).

Several successful large-scale agile projects have been documented.Template:Where BT has had several hundred developers situated in the UK, Ireland and India working collaboratively on projects and using Agile methods.[*citation needed*]

In terms of outsourcing agile development, Michael Hackett, Sr. Vice President of LogiGear Corporation has stated that "the offshore team. . . should have expertise, experience, good communication skills, inter-cultural understanding, trust and understanding between members and groups and with each other."[34]

Barry Boehm and Richard Turner suggest that risk analysis be used to choose between adaptive ("agile") and predictive ("plan-driven") methods.[29] The authors suggest that each side of the continuum has its own *home ground* as follows:

Agile home ground:[29]

- Low criticality
- Senior developers
- Requirements change often
- Small number of developers
- Culture that thrives on chaos

Plan-driven home ground:[29]

- High criticality
- Junior developers

- Requirements do not change often
- Large number of developers
- Culture that demands order

Formal methods:

- Extreme criticality
- Senior developers
- Limited requirements, limited features see Wirth's law
- Requirements that can be modeled
- Extreme quality

Experience reports

Agile development has been the subject of several conferences. Some of these conferences have had academic backing and included peer-reviewed papers, including a peer-reviewed experience report track. The experience reports share industry experiences with agile software development.

As of 2006, experience reports have been or will be presented at the following conferences:

- XP (2000,[35] 2001, 2002, 2003, 2004, 2005, 2006,[36] 2010 (proceedings published by IEEE)[37])
- XP Universe (2001[38])
- XP/Agile Universe (2002,[39] 2003,[40] 2004[41])
- Agile Development Conference[42] (2003,2004,2007,2008) (peer-reviewed; proceedings published by IEEE)

References

1. Beck, Kent; et al. (2001). "Manifesto for Agile Software Development". Agile Alliance. http://agilemanifesto.org/. Retrieved 2010-06-14.
2. Gerald M. Weinberg, as quoted in Larman, Craig; Basili, Victor R. (June 2003). "Iterative and Incremental Development: A Brief History". *Computer* **36** (6): 47–56. doi:10.1109/MC.2003.1204375. ISSN 0018-9162. "We were doing incremental development as early as 1957, in Los Angeles, under the direction of Bernie Dimsdale [at IBM's ServiceBureau Corporation]. He was a colleague of John von Neumann, so perhaps he learned it there, or assumed it as totally natural. I do remember Herb Jacobs (primarily, though we all participated) developing a large simulation for Motorola, where the technique used was, as far as I can tell .... All of us, as far as I can remember, thought waterfalling of a huge project was rather stupid, or at least ignorant of the realities. I think what the waterfall description did for us was make us realize that we were doing something else, something unnamed except for 'software development.'".
3. Edmonds, E. A. (1974). "A Process for the Development of Software for Nontechnical Users as an Adaptive System". *General Systems* **19**: 215–18.
4. Larman, Craig (2004). *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley. p. 27. ISBN 9780131111554
5. Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Stephen J. Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas
6. Beck, Kent; et al. (2001). "Principles behind the Agile Manifesto". Agile Alliance. http://www.agilemanifesto.org/principles.html. Retrieved 2010-06-06.
7. Anderson, David (2005). "Declaration of Interdependence". http://pmdoi.org.
8. Beck, Kent (1999). "Embracing Change with Extreme Programming". *Computer* **32** (10): 70–77. doi:10.1109/2.796139.
9. Black, S. E.; Boca., P. P.; Bowen, J. P.; Gorman, J.; Hinchey, M. G. (September 2009). "Formal versus agile: Survival of the fittest". *IEEE Computer* **49** (9): 39–45.
10. Boehm, B.; R. Turner (2004). *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston, MA: Addison-Wesley. ISBN 0-321-18612-5. Appendix A, pages 165-194
11. Aydin, M.N., Harmsen, F., Slooten, K. v., & Stagwee, R. A. (2004). An Agile Information Systems Development Method in use. *Turk J Elec Engin, 12(2),* 127-138

12. Abrahamsson, P., Warsta, J., Siponen, M.T., & Ronkainen, J. (2003). New Directions on Agile Methods: A Comparative Analysis. *Proceedings of ICSE'03*, 244-254
13. Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile Software Development Methods: Review and Analysis. *VTT Publications 478*
14. Aydin, M.N., Harmsen, F., Slooten van K., & Stegwee, R.A. (2005). On the Adaptation of An Agile Information(Suren) Systems Development Method. *Journal of Database Management Special issue on Agile Analysis, Design, and Implementation, 16(4),* 20-24
15. "David Bock's Weblog : Weblog". Jroller.com. http://jroller.com/page/bokmann?entry=improving_your_processes_aim_high. Retrieved 2010-04-02.
16. "Agility measurement index". Doi.acm.org. http://doi.acm.org/10.1145/1185448.1185509. Retrieved 2010-04-02.
17. Peter Lappo; Henry C.T. Andrew. "Assessing Agility". http://www.smr.co.uk/presentations/measure.pdf. Retrieved 2010-06-06.
18. Kurian, Tisni (2006). "Agility Metrics: A Quantitative Fuzzy Based Approach for Measuring Agility of a Software Process" *ISAM-Proceedings of International Conference on Agile Manufacturing'06(ICAM-2006)*, Norfolk, U.S.
19. Joe Little (2007-12-02). "Nokia test, A Scrum specific test". Agileconsortium.blogspot.com. http://agileconsortium.blogspot.com/2007/12/nokia-test.html. Retrieved 2010-06-06.
20. Mark Seuffert, Piratson Technologies, Sweden. "Karlskrona test, A generic agile adoption test". Piratson.se. http://www.piratson.se/archive/Agile_Karlskrona_Test.html. Retrieved 2010-06-06.
21. "How agile are you, A Scrum specific test". Agile-software-development.com. http://www.agile-software-development.com/2008/01/how-agile-are-you-take-this-42-point.html. Retrieved 2010-06-06.
22. "Agile Methodologies Survey Results" (PDF). Shine Technologies. 2003. http://www.shinetech.com/attachments/104_ShineTechAgileSurvey2003-01-17.pdf. Retrieved 2010-06-03. "95% [stated] that there was either no effect or a cost reduction . . . 93% stated that productivity was better or significantly better . . . 88% stated that quality was better or significantly better . . . 83% stated that business satisfaction was better or significantly better"
23. Ambler, Scott (August 3, 2006). "Survey Says: Agile Works in Practice". *Dr. Dobb's*. http://www.drdobbs.com/architecture-and-design/191800169;jsessionid=2QJ23QRYM3H4PQE1GHPCKH4ATMY32JVN?queryText=agile+survey. Retrieved 2010-06-03. "Only 6 percent indicated that their productivity was lowered . . . No change in productivity was reported by 34 percent of respondents and 60 percent reported increased productivity. . . . 66 percent [responded] that the quality is higher. . . . 58 percent of organizations report improved satisfaction, whereas only 3 percent report reduced satisfaction."
24. "The State of Agile Development" (PDF). VersionOne, Inc.. 2008. http://www.versionone.com/pdf/3rdAnnualStateOfAgile_FullDataReport.pdf. Retrieved 2010-07-03. "Agile delivers"
25. "Answering the "Where is the Proof That Agile Methods Work" Question". Agilemodeling.com. 2007-01-19. http://www.agilemodeling.com/essays/proof.htm. Retrieved 2010-04-02.
26. Agile Processes Workshop II Managing Multiple Concurrent Agile Projects. Washington: OOPSLA 2002
27. W. Scott Ambler (2006) "Supersize Me (http://www.drdobbs.com/184415491)" in Dr. Dobb's Journal, February 15, 2006.
28. Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. Boston, MA: Addison-Wesley. ISBN 0-321-27865-8.
29. Boehm, B.; R. Turner (2004). *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston, MA: Addison-Wesley. pp. 55–57. ISBN 0-321-18612-5.
30. Schaaf, R.J. (2007). "Agility XL", Systems and Software Technology Conference 2007 (http://www.sstc-online.org/Proceedings/2007/pdfs/RJS1722.pdf), Tampa, FL
31. "Bridging the Distance". Sdmagazine.com. http://www.drdobbs.com/architecture-and-design/184414899. Retrieved 2011-02-01.
32. Martin Fowler. "Using an Agile Software Process with Offshore Development". Martinfowler.com.

http://www.martinfowler.com/articles/agileOffshore.html. Retrieved 2010-06-06.
33. [The Art of Agile Development James Shore & Shane Warden pg 47]
34. [1] (http://www.logigear.com/in-the-news/973-agile.html) LogiGear, PC World Viet Nam, Jan 2011
35. 2000 (http://ciclamino.dibe.unige.it/xp2000/)
36. "2006". Virtual.vtt.fi. http://virtual.vtt.fi/virtual/xp2006/. Retrieved 2010-06-06.
37. "2010". Xp2010.org. http://www.xp2010.org/. Retrieved 2010-06-06.
38. 2001 (http://www.xpuniverse.com/2001/xpuPapers.htm)
39. 2002 (http://www.xpuniverse.com/2002/schedule/schedule)
40. 2003 (http://www.xpuniverse.com/2003/schedule/index)
41. 2004 (http://www.xpuniverse.com/2004/schedule/index)
42. "Agile Development Conference". Agile200x.org. http://www.agile200x.org/. Retrieved 2010-06-06.

Further reading

- Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile Software Development Methods: Review and Analysis. *VTT Publications 478*.
- Cohen, D., Lindvall, M., & Costa, P. (2004). An introduction to agile methods. In *Advances in Computers* (pp. 1–66). New York: Elsevier Science.
- Dingsøyr, Torgeir, Dybå, Tore and Moe, Nils Brede (ed.): *Agile Software Develoment: Current Research and Future Directions* (http://www.amazon.co.uk/Agile-Software-Development-Research-Directions/dp/3642125743), Springer, Berlin Heidelberg, 2010.
- Fowler, Martin. *Is Design Dead?* (http://www.martinfowler.com/articles/designDead.html). Appeared in *Extreme Programming Explained*, G. Succi and M. Marchesi, ed., Addison-Wesley, Boston. 2001.
- Larman, Craig and Basili, Victor R. *Iterative and Incremental Development: A Brief History* IEEE Computer, June 2003 (http://www.highproductivity.org/r6047.pdf)
- Riehle, Dirk. *A Comparison of the Value Systems of Adaptive Software Development and Extreme Programming: How Methodologies May Learn From Each Other* (http://www.riehle.org/computer-science/research/2000/xp-2000.html). Appeared in *Extreme Programming Explained*, G. Succi and M. Marchesi, ed., Addison-Wesley, Boston. 2001.
- Rother, Mike (2009). *Toyota Kata*. McGraw-Hill. ISBN 0071635238. http://books.google.com/?id=_1lhPgAACAAJ&dq=toyota+kata
- M. Stephens, D. Rosenberg. *Extreme Programming Refactored: The Case Against XP*. Apress L.P., Berkeley, California. 2003. ISBN 1-59059-096-1

External links

- Manifesto for Agile Software Development (http://www.agileManifesto.org/)
- The Agile Alliance (http://www.agilealliance.org/)
- The Agile Executive (http://theagileexecutive.com/)
- Article Two Ways to Build a Pyramid by John Mayo-Smith (http://www.informationweek.com/news/software/development/showArticle.jhtml?articleID=6507351)
- Agile Software Development: A gentle introduction (http://www.agile-process.org/)
- The New Methodology (http://martinfowler.com/articles/newMethodology.html) Martin Fowler's description of the background to agile methods
- Agile Journal (http://www.agilejournal.com/) - Largest online community focused specifically on agile development
- [9] (http://www.dmoz.org%7CComputers/Programming/Methodologies/Agile%7CAgile)]
- Agile Cookbook (http://agilecookbook.com/)
- Ten Authors of The Agile Manifesto Celebrate its Tenth Anniversary (http://www.pragprog.com/magazines/2011-02/agile--)

Standards

There are a few industry standards related to process improvement models we should mention briefly. For you as a beginner, it is enough to know they exist. However, if you start working for large corporations, you will find that many will follow one or the other of these standards.

Capability Maturity Model Integration

The Capability Maturity Model Integration (CMMI) is one of the leading models and based on best practice. Independent assessments grade organizations on how well they follow their defined processes, not on the quality of those processes or the software produced. CMMI has replaced CMM.

ISO 9000

ISO 9000 describes standards for a formally organized process to manufacture a product and the methods of managing and monitoring progress. Although the standard was originally created for the manufacturing sector, ISO 9000 standards have been applied to software development as well. Like CMMI, certification with ISO 9000 does not guarantee the quality of the end result, only that formalized business processes have been followed.

ISO 15504

ISO 15504, also known as Software Process Improvement Capability Determination (SPICE), is a "framework for the assessment of software processes". This standard is aimed at setting out a clear model for process comparison. SPICE is used much like CMMI. It models processes to manage, control, guide and monitor software development. This model is then used to measure what a development organization or project team actually does during software development. This information is analyzed to identify weaknesses and drive improvement. It also identifies strengths that can be continued or integrated into common practice for that organization or team.

External Links

- CMMI Official Website (http://www.sei.cmu.edu/cmmi)
- Capability Maturity Model (http://www.dmoz.org/Computers/Programming/Methodologies/Capability_Maturity_Model/) at DMOZ
- ISO 9000 (http://www.dmoz.org/Science/Reference/Standards/Individual_Standards/ISO/ISO_9000/) at DMOZ
- Introduction to ISO 9000 and ISO 14000 (http://www.iso.org/iso/iso_catalogue/management_standards/iso_9000_iso_14000.htm)
- ISO 15504 News (isospice) (http://www.isospice.com)
- Automotive SPICE (http://www.automotivespice.com/)

Life Cycle

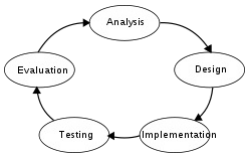
The **Systems Development Life Cycle (SDLC)**, or *Software Development Life Cycle* in systems engineering, information systems and software engineering, is the process of creating or altering systems, and the models and methodologies that people use to develop these systems. The concept generally refers to computer or information systems.

Model of the Systems Development Life Cycle

In software engineering the SDLC concept underpins many kinds of software development methodologies. These methodologies form the framework for planning and controlling the creation of an information system[1]: the software development process.

Overview

Systems Development Life Cycle (SDLC) is a process used by a systems analyst to develop an information system, including requirements, validation, training, and user (stakeholder) ownership. Any SDLC should result in a high quality system that meets or exceeds customer expectations, reaches completion within time and cost estimates, works effectively and efficiently in the current and planned Information Technology infrastructure, and is inexpensive to maintain and cost-effective to enhance.[2]



Model of the Systems Development Life Cycle

Computer systems are complex and often (especially with the recent rise of Service-Oriented Architecture) link multiple traditional systems potentially supplied by different software vendors. To manage this level of complexity, a number of SDLC models have been created: "waterfall"; "fountain"; "spiral"; "build and fix"; "rapid prototyping"; "incremental"; and "synchronize and stabilize". [3]

SDLC models can be described along a spectrum of agile to iterative to sequential. Agile methodologies, such as XP and Scrum, focus on light-weight processes which allow for rapid changes along the development cycle. Iterative methodologies, such as Rational Unified Process and Dynamic Systems Development Method, focus on limited project scopes and expanding or improving products by multiple iterations. Sequential or big-design-up-front (BDUF) models, such as Waterfall, focus on complete and correct planning to guide large projects and risks to successful and predictable results[*citation needed*]. Other models, such as Anamorphic Development, tend to focus on a form of development that is guided by project scope and adaptive iterations of feature development.

In project management a project can be defined both with a project life cycle (PLC) and an SDLC, during which slightly different activities occur. According to Taylor (2004) "the project life cycle encompasses all the activities of the project, while the systems development life cycle focuses on realizing the product requirements".[4]

History

The **Systems Life Cycle (SLC)** is a type of methodology used to describe the process for building information systems, intended to develop information systems in a very deliberate, structured and methodical way, reiterating each stage of the life cycle. The systems development life cycle, according to Elliott & Strachan & Radford (2004), "originated in the 1960s,to develop large scale functional business systems in an age of large scale business conglomerates. Information systems activities revolved around heavy data processing and number crunching routines".[5]

Several systems development frameworks have been partly based on SDLC, such as the Structured Systems Analysis and Design Method (SSADM) produced for the UK government Office of Government Commerce in the 1980s. Ever since, according to Elliott (2004), "the traditional life cycle approaches to systems development have been increasingly replaced with alternative approaches and frameworks, which attempted to overcome some of the inherent deficiencies of the traditional SDLC".[5]

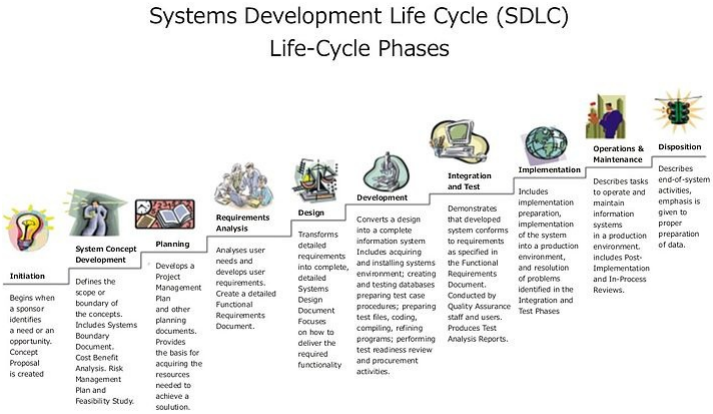
Systems development phases

The System Development Life Cycle framework provides a sequence of activities for system designers and developers to follow. It consists of a set of steps or phases in which each phase of the SDLC uses the results of the previous one.

A Systems Development Life Cycle (SDLC) adheres to important phases that are essential for developers, such as planning, analysis, design, and implementation, and are explained in the section below. A number of system development life cycle (SDLC) models have been created: waterfall, fountain, spiral, build and fix, rapid prototyping, incremental, and synchronize and stabilize. The oldest of these, and the best known, is the waterfall model: a sequence of stages in which the output of each stage becomes the input for the next. These stages can be characterized and divided up in different ways, including the following[6]:

- **Project planning, feasibility study**: Establishes a high-level view of the intended project and determines its goals.
- **Systems analysis, requirements definition**: Refines project goals into defined functions and operation of the intended application. Analyzes end-user information needs.
- **Systems design**: Describes desired features and operations in detail, including screen layouts, business rules, process diagrams, pseudocode and other documentation.
- **Implementation**: The real code is written here.
- **Integration and testing**: Brings all the pieces together into a special testing environment, then checks for errors, bugs and interoperability.
- **Acceptance, installation, deployment**: The final stage of initial development, where the software is put into production and runs actual business.
- **Maintenance**: What happens during the rest of the software's life: changes, correction, additions, moves to a different computing platform and more. This, the least glamorous and perhaps most important step of all, goes on seemingly forever.

In the following example (see picture) these stage of the Systems Development Life Cycle are divided in ten steps from definition to creation and modification of IT work products:



The tenth phase occurs when the system is disposed of and the task performed is either eliminated or transferred to other systems. The tasks and work products for each phase are described in subsequent chapters. [7]

Not every project will require that the phases be sequentially executed. However, the phases are interdependent. Depending upon the size and complexity of the project, phases may be combined or may overlap.[7]

System analysis

The goal of system analysis is to determine where the problem is in an attempt to fix the system. This step involves breaking down the system in different pieces to analyze the situation, analyzing project goals, breaking down what needs to be created and attempting to engage users so that definite requirements can be defined.

Requirements analysis sometimes requires individuals/teams from client as well as service provider sides to get detailed and accurate requirements; often there has to be a lot of communication to and from to understand these requirements. Requirement gathering is the most crucial aspect as many times communication gaps arise in this phase and this leads to validation errors and bugs in the software program.

Design

In systems design the design functions and operations are described in detail, including screen layouts, business rules, process diagrams and other documentation. The output of this stage will describe the new system as a collection of modules or subsystems.

The design stage takes as its initial input the requirements identified in the approved requirements document. For each requirement, a set of one or more design elements will be produced as a result of interviews, workshops, and/or prototype efforts.

Design elements describe the desired software features in detail, and generally include functional hierarchy diagrams, screen layout diagrams, tables of business rules, business process diagrams, pseudocode, and a complete entity-relationship diagram with a full data dictionary. These design elements are intended to describe the software in sufficient detail that skilled programmers may develop the software with minimal additional input design.

Implementation

Modular and subsystem programming code will be accomplished during this stage. Unit testing and module testing are done in this stage by the developers. This stage is intermingled with the next in that individual modules will need testing before integration to the main project.

Testing

The code is tested at various levels in software testing. Unit, system and user acceptance testings are often performed. This is a grey area as many different opinions exist as to what the stages of testing are and how much if any iteration occurs. Iteration is not generally part of the waterfall model, but usually some occur at this stage. In the testing the whole system is test one by one

Following are the types of testing:

- Defect testing
- Path testing
- Data set testing
- Unit testing
- System testing
- Integration testing
- Black box testing

- White box testing
- Regression testing
- Automation testing
- User acceptance testing
- Performance testing

Operations and maintenance

The deployment of the system includes changes and enhancements before the decommissioning or sunset of the system. Maintaining the system is an important aspect of SDLC. As key personnel change positions in the organization, new changes will be implemented, which will require system updates.

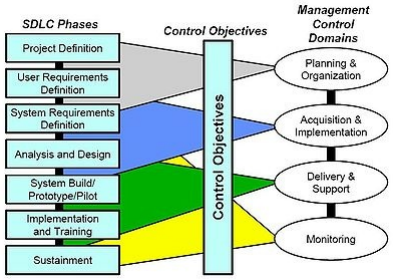
Systems Analysis and Design

The **Systems Analysis and Design (SAD)** is the process of developing Information Systems (IS) that effectively use of hardware, software, data, process, and people to support the company's business objectives.

Systems development life cycle topics

Management and control

The Systems Development Life Cycle (SDLC) phases serve as a programmatic guide to project activity and provide a flexible but consistent way to conduct projects to a depth matching the scope of the project. Each of the SDLC phase objectives are described in this section with key deliverables, a description of recommended tasks, and a summary of related control objectives for effective management. It is critical for the project manager to establish and monitor control objectives during each SDLC phase while executing projects. Control objectives help to provide a clear statement of the desired result or purpose and should be used throughout the entire SDLC process. Control objectives can be grouped into major categories (Domains), and relate to the SDLC phases as shown in the figure.[8]

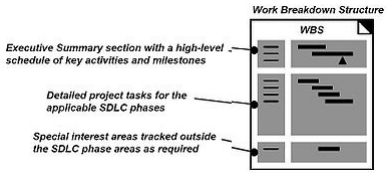


SDLC Phases Related to Management Controls.[8]

To manage and control any SDLC initiative, each project will be required to establish some degree of a Work Breakdown Structure (WBS) to capture and schedule the work necessary to complete the project. The WBS and all programmatic material should be kept in the "Project Description" section of the project notebook. The WBS format is mostly left to the project manager to establish in a way that best describes the project work. There are some key areas that must be defined in the WBS as part of the SDLC policy. The following diagram describes three key areas that will be addressed in the WBS in a manner established by the project manager.[8]

Work breakdown structured organization

The upper section of the Work Breakdown Structure (WBS) should identify the major phases and milestones of the project in a summary fashion. In addition, the upper section should provide an overview of the full scope and timeline of the project and will be part of the initial project description effort leading to project approval. The middle section of the WBS is based on the seven Systems Development Life Cycle (SDLC) phases as a guide for WBS task development. The WBS elements should consist of milestones and "tasks" as opposed to "activities" and have a definitive period (usually two weeks or more). Each task must have a measurable output (e.x. document, decision, or analysis). A WBS task may rely on one or more activities (e.g. software engineering, systems engineering) and may require close coordination with other tasks, either internal or external to the project. Any part of the project needing support from contractors should have a Statement of work (SOW) written to include the appropriate tasks from the SDLC phases. The development of a SOW does not occur during a specific phase of SDLC but is developed to include the work from the SDLC process that may be conducted by external resources such as contractors and struct.[8]



Work Breakdown Structure.[8]

Baselines in the SDLC

Baselines are an important part of the Systems Development Life Cycle (SDLC). These baselines are established after four of the five phases of the SDLC and are critical to the iterative nature of the model .[9] Each baseline is considered as a milestone in the SDLC.

- Functional Baseline: established after the conceptual design phase.
- Allocated Baseline: established after the preliminary design phase.
- Product Baseline: established after the detail design and development phase.
- Updated Product Baseline: established after the production construction phase.

Complementary to SDLC

Complementary Software development methods to Systems Development Life Cycle (SDLC) are:

- Software Prototyping
- Joint Applications Design (JAD)
- Rapid Application Development (RAD)
- Extreme Programming (XP); extension of earlier work in Prototyping and RAD.
- Open Source Development
- End-user development
- Object Oriented Programming

Comparison of Methodology Approaches (Post, & Anderson 2006)[10]

	SDLC	RAD	Open Source	Objects	JAD	Prototyping	End User
Control	Formal	MIS	Weak	Standards	Joint	User	User
Time Frame	Long	Short	Medium	Any	Medium	Short	Short
Users	Many	Few	Few	Varies	Few	One or Two	One
MIS staff	Many	Few	Hundreds	Split	Few	One or Two	None
Transaction/DSS	Transaction	Both	Both	Both	DSS	DSS	DSS
Interface	Minimal	Minimal	Weak	Windows	Crucial	Crucial	Crucial
Documentation and training	Vital	Limited	Internal	In Objects	Limited	Weak	None
Integrity and security	Vital	Vital	Unknown	In Objects	Limited	Weak	Weak
Reusability	Limited	Some	Maybe	Vital	Limited	Weak	None

Strengths and weaknesses

Few people in the modern computing world would use a strict waterfall model for their Systems Development Life Cycle (SDLC) as many modern methodologies have superseded this thinking. Some will argue that the SDLC no longer applies to models like Agile computing, but it is still a term widely in use in Technology circles. The SDLC practice has advantages in traditional models of software development, that lends itself more to a structured environment. The disadvantages to using the SDLC methodology is when there is need for iterative development or (i.e. web development or e-commerce) where stakeholders need to review on a regular basis the software being designed. Instead of viewing SDLC from a strength or weakness perspective, it is far more important to take the best practices from the SDLC model and apply it to whatever may be most appropriate for the software being designed.

A comparison of the strengths and weaknesses of SDLC:

Strength and Weaknesses of SDLC [10]

Strengths	Weaknesses
Control.	Increased development time.
Monitor Large projects.	Increased development cost.
Detailed steps.	Systems must be defined up front.
Evaluate costs and completion targets.	Rigidity.
Documentation.	Hard to estimate costs, project overruns.
Well defined user input.	User input is sometimes limited.
Ease of maintenance.	
Development and design standards.	
Tolerates changes in MIS staffing.	

An alternative to the SDLC is Rapid Application Development, which combines prototyping, Joint Application Development and implementation of CASE tools. The advantages of RAD are speed, reduced development cost, and active user involvement in the development process.

References

1. SELECTING A DEVELOPMENT APPROACH (http://www.cms.hhs.gov/SystemLifecycleFramework/Downloads/SelectingDevelopmentApproach.pdf). Retrieved 27 October 2008.
2. "Systems Development Life Cycle" (http://foldoc.org/foldoc.cgi?Systems+Development+Life+Cycle). In: Foldoc(2000-12-24)
3. http://docs.google.com/viewer?a=v&q=cache:bfhOl8jp1S8J:condor.depaul.edu/~jpetlick/extra/394/Session2.ppt+&hl=en&pid=bl&srcid=ADGQIhrj1683hMefBNkak7FkQJCAwd-i0-_aQfEVEEKP177h4mmkvMMWJ7&sig=AHIEtbRhMlZ-TUyioKEhLQQxXk1WoSJXWA
4. James Taylor (2004). *Managing Information Technology Projects*. p.39..
5. Geoffrey Elliott & Josh Strachan (2004) *Global Business Information Technology*. p.87.
6. http://www.computerworld.com/s/article/71151/System_Development_Life_Cycle
7. US Department of Justice (2003). INFORMATION RESOURCES MANAGEMENT (http://www.usdoj.gov/jmd/irm/lifecycle/ch1.htm) Chapter 1. Introduction.
8. U.S. House of Representatives (1999). *Systems Development Life-Cycle Policy* (http://www.house.gov/cao-opp/PDFSolicitations/SDLCPOL.pdf). p.13.
9. Blanchard, B. S., & Fabrycky, W. J.(2006) *Systems engineering and analysis* (4th ed.) New Jersey: Prentice Hall. p.31
10. Post, G., & Anderson, D., (2006). *Management information systems: Solving business problems with information technology*. (4th ed.). New York: McGraw-Hill Irwin.

Further reading

- Blanchard, B. S., & Fabrycky, W. J.(2006) *Systems engineering and analysis* (4th ed.) New Jersey: Prentice Hall.
- Cummings, Haag (2006). *Management Information Systems for the Information Age*. Toronto, McGraw-Hill Ryerson
- Beynon-Davies P. (2009). *Business Information Systems*. Palgrave, Basingstoke. ISBN 978-0-230-20368-6
- Computer World, 2002 (http://www.computerworld.com/developmenttopics/development/story/0,10801,71151,00.html), Retrieved on June 22, 2006 from the World Wide Web:
- Management Information Systems, 2005 (http://www.cbe.wwu.edu/misclasses/MIS320_Spring06_Bajwa/Chap006.ppt), Retrieved on June 22, 2006 from the World Wide Web:

External links

- Slide show video (https://www.youtube.com/watch?v=TfxAfP4LBSA)
- The Agile System Development Lifecycle (http://www.ambysoft.com/essays/agileLifecycle.html)
- Pension Benefit Guaranty Corporation - Information Technology Solutions Lifecycle Methodology (http://www.pbgc.gov/docs/ITSLCM%20V2007.1.pdf)
- HHS Enterprise Performance Life Cycle Framework (http://www.hhs.gov/ocio/eplc/eplc_framework_v1point2.pdf)

Rapid Application Development

Rapid application development (**RAD**) refers to a type of software development methodology that uses minimal planning in favor of rapid prototyping. The "planning" of software developed using RAD is interleaved with writing the software itself. The lack of extensive pre-planning generally allows software to be written much faster, and makes it easier to change requirements.

Overview

Rapid application development is a software development methodology that involves methods like iterative development and software prototyping. According to Whitten (2004), it is a merger of various structured techniques, especially data-driven Information Engineering, with prototyping techniques to accelerate software systems development.[1]

In rapid application development, structured techniques and prototyping are especially used to define users' requirements and to design the final system. The development process starts with the development of preliminary data models and business process models using structured techniques. In the next stage, requirements are verified using prototyping, eventually to refine the data and process models. These stages are repeated iteratively; further development results in "a combined business requirements and technical design statement to be used for constructing new systems".[1]

RAD approaches may entail compromises in functionality and performance in exchange for enabling faster development and facilitating application maintenance.

History

Rapid application development is a term originally used to describe a software development process introduced by James Martin in 1991. Martin's methodology involves iterative development and the construction of prototypes. More recently, the term and its acronym have come to be used in a broader, general sense that encompasses a variety of methods aimed at speeding application development, such as the use of software frameworks of varied types, such as web application frameworks.

Rapid application development was a response to non-agile processes developed in the 1970s and 1980s, such as the Structured Systems Analysis and Design Method and other Waterfall models. One problem with previous methodologies was that applications took so long to build that requirements had changed before the system was complete, resulting in inadequate or even unusable systems. Another problem was the assumption that a methodical requirements analysis phase alone would identify all the critical requirements. Ample evidence[*citation needed*] attests to the fact that this is seldom the case, even for projects with highly experienced professionals at all levels.

Starting with the ideas of Brian Gallagher, Alex Balchin, Barry Boehm and Scott Shultz, James Martin developed the rapid application development approach during the 1980s at IBM and finally formalized it by publishing a book in 1991, *Rapid Application Development*.

Relative effectiveness

The shift from traditional session-based client/server development to open sessionless and collaborative development like Web 2.0 has increased the need for faster iterations through the phases of the SDLC.[2] This, coupled with the growing use of open source frameworks and products in core commercial development, has, for many developers, rekindled interest in finding a silver bullet RAD methodology.

Although most RAD methodologies foster software re-use, small team structure and distributed system development, most RAD practitioners recognize that, ultimately, no one “rapid” methodology can provide an order of magnitude improvement over any other development methodology.

All types of RAD have the potential for providing a good framework for faster product development with improved software quality, but successful implementation and benefits often hinge on project type, schedule, software release cycle and corporate culture. It may also be of interest that some of the largest software vendors such as Microsoft[3] and IBM[4] do not extensively use RAD in the development of their flagship products and for the most part, they still primarily rely on traditional waterfall methodologies with some degree of spiraling.[5]

This table contains a high-level summary of some of the major types of RAD and their relative strengths and weaknesses.

Agile software development (Agile)	
Pros	Minimizes feature creep by developing in short intervals resulting in miniature software projects and releasing the

	product in mini-increments.
Cons	Short iteration may add too little functionality, leading to significant delays in final iterations. Since Agile emphasizes real-time communication (preferably face-to-face), using it is problematic for large multi-team distributed system development. Agile methods produce very little written documentation and require a significant amount of post-project documentation.
Extreme Programming (XP)	
Pros	Lowers the cost of changes through quick spirals of new requirements. Most design activity occurs incrementally and on the fly.
Cons	Programmers must work in pairs, which is difficult for some people. No up-front "detailed design" occurs, which can result in more redesign effort in the long term. The business champion attached to the project full time can potentially become a single point of failure for the project and a major source of stress for a team.
Joint application design (JAD)	
Pros	Captures the voice of the customer by involving them in the design and development of the application through a series of collaborative workshops called JAD sessions.
Cons	The client may create an unrealistic product vision and request extensive gold-plating, leading a team to over- or under-develop functionality.
Lean software development (LD)	
Pros	Creates minimalist solutions (i.e., needs determine technology) and delivers less functionality earlier; per the policy that 80% today is better than 100% tomorrow.
Cons	Product may lose its competitive edge because of insufficient core functionality and may exhibit poor overall quality.
Rapid application development (RAD)	
Pros	Promotes strong collaborative atmosphere and dynamic gathering of requirements. Business owner actively participates in prototyping, writing test cases and performing unit testing.
Cons	Dependence on strong cohesive teams and individual commitment to the project. Decision making relies on the feature functionality team and a communal decision-making process with lesser degree of centralized PM and engineering authority.
Scrum	
Pros	Improved productivity in teams previously paralyzed by heavy "process", ability to prioritize work, use of backlog for completing items in a series of short iterations or sprints, daily measured progress and communications.
Cons	Reliance on facilitation by a master who may lack the political skills to remove impediments and deliver the sprint goal. Due to relying on self-organizing teams and rejecting traditional

	centralized "process control", internal power struggles can paralyze a team.

Table 1: Pros and Cons of various RAD types

Criticism

Since rapid application development is an iterative and incremental process, it can lead to a succession of prototy failures may be avoided if the application development tools are robust, flexible, and put to proper use. This is add agile variants.

Practical implications

When organizations adopt rapid development methodologies, care must be taken to avoid role and responsibility co between team and client. In addition, especially in cases where the client is absent or not able to participate wit endowed with this authority on behalf of the client to ensure appropriate prioritisation of non-functional require without a thorough and formally documented design phase.[6]

References

1. Whitten, Jeffrey L.; Lonnie D. Bentley, Kevin C. Dittman. (2004). *Systems Analysis and Design Methods*. 6th ec
2. Maurer and S. Martel. (2002). "Extreme Programming: Rapid Development for Web-Based Applications". IEEE
3. Andrew Begel, Nachiappan Nagappan. "Usage and Perceptions of Agile Software Development in an Industrial C http://research.microsoft.com/pubs/56015/AgileDevatMS-ESEM07.pdf. Retrieved 2008-11-15.
4. E. M. Maximilien and L. Williams. (2003). "Assessing Test-driven Development at IBM". Proceedings of Interna 2003.
5. M. Stephens, Rosenberg, D. (2003). "Extreme Programming Refactored: The Case Against XP". Apress, 2003.
6. Gerber, Aurona; Van der Merwe, Alta; Alberts, Ronell; (2007), Implications of Rapid Development Methodologi za/~agerber/publications.html)

Further reading

- Steve McConnell (1996). *Rapid Development: Taming Wild Software Schedules*, Microsoft Press Books, ISBN 97
- Kerr, James M.; Hunter, Richard (1993). *Inside RAD: How to Build a Fully-Functional System in 90 Days or L*
- Ellen Gottesdiener (1995). "RAD Realities: Beyond the Hype to How RAD Really Works (http://ebgconsulting. pdf)" Application Development Trends
- Ken Schwaber (1996). *Agile Project Management with Scrum*, Microsoft Press Books, ISBN 978-0735619937
- Steve McConnell (2003). *Professional Software Development: Shorter Schedules, Higher Quality Products, More* 978-0321193674
- Dean Leffingwell (2007). *Scaling Software Agility: Best Practices for Large Enterprises*, Addison-Wesley Profess

Extreme Programming

Extreme Programming (XP) is a software development methodology which is intended to improve softw customer requirements. As a type of agile software development,[1][2][3] it advocates frequent "releases" in short intended to improve productivity and introduce checkpoints where new customer requirements can be adopted.

Other elements of extreme programming include: programming in pairs or doing extensive code review, unit te features until they are actually needed, a flat management structure, simplicity and clarity in code, expecting cha passes and the problem is better understood, and frequent communication with the customer and among programm from the idea that the beneficial elements of traditional software engineering practices are taken to "extreme" levels better. It is unrelated to "cowboy coding", which is more free-form and unplanned. It does not advocate "death mar sustainable pace.[5]

Critics have noted several potential drawbacks,[6] including problems with unstable requirements, no documented an overall design specification or document.

History

Extreme Programming was created by Kent Beck during his work on the Chrysler Comprehensive Compensation March 1996 and began to refine the development method used in the project and wrote a book on the method Chrysler cancelled the C3 project in February 2000, after the company was acquired by Daimler-Benz.[7]

Although extreme programming itself is relatively new, many of its practices have been around for some time; t example, the "practice of test-first development, planning and writing tests before each micro-increment" was used To shorten the total development time, some formal test documents (such as for acceptance testing) have been dev NASA independent test group can write the test procedures, based on formal requirements and logical limits, bef

XP, this concept is taken to the extreme level by writing automated tests (perhaps inside of software modules) v
rather than only testing the larger features. Some other XP practices, such as refactoring, modularity, bottom-up d
published in 1984.[8]

Origins

Software development in the 1990s was shaped by two major influences: internally, object-oriented programming r
by some in the industry; externally, the rise of the Internet and the dot-com boom emphasized speed-to-market a
requirements demanded shorter product life-cycles, and were often incompatible with traditional methods of softwar

The Chrysler Comprehensive Compensation System was started in order to determine the best way to use obje
research, with Smalltalk as the language and GemStone as the data access layer. They brought in Kent Beck,[6]
system, but his role expanded as he noted several problems they were having with their development process. He
practices based on his work with his frequent collaborator, Ward Cunningham. Beck describes the early conception o

> The first time I was asked to lead a team, I asked them to do a little bit of the things I thought were sensible,
> the line. I thought, "Damn the torpedoes, at least this will make a good article," [and] asked the team to crank
> leave out everything else.

Beck invited Ron Jeffries to the project to help develop and refine these methods. Jeffries thereafter acted as a coach

Information about the principles and practices behind XP was disseminated to the wider world through disc
contributors discussed and expanded upon the ideas, and some spin-off methodologies resulted (see agile software d
using a hypertext system map on the XP website at "http://www.extremeprogramming.org" circa 1999.

Beck edited a series of books on XP, beginning with his own *Extreme Programming Explained* (1999, ISBN 0-2
audience. Authors in the series went through various aspects attending XP and its practices. Even a book was writte

Current state

XP created quite a buzz in the late 1990s and early 2000s, seeing adoption in a number of environments radically di

The high discipline required by the original practices often went by the wayside, causing some of these practices, su
unfinished, on individual sites. For example, the practice of end-of-day integration tests, for a particular project
mutually agreed dates. Such a more relaxed schedule could avoid people feeling rushed to generate artificial stubs j
for some complex features to be more fully developed over a several-day period. However, some level of periodic int
tangent efforts before too much work is invested in divergent, wrong directions.

Meanwhile, other agile development practices have not stood still, and XP is still evolving, assimilating more less
edition of *Extreme Programming Explained*, Beck added more values and practices and differentiated between prima

Concept

Goals

Extreme Programming Explained describes Extreme Programming as a software development discipline that organiz

In traditional system development methods (such as SSADM or the waterfall model) the requirements for the syste
fixed from that point on. This means that the cost of changing the requirements at a later stage (a common feat
other agile software development methods, XP attempts to reduce the cost of change by having multiple short deve
natural, inescapable and desirable aspect of software development projects, and should be planned for instead of atte

Extreme programming also introduces a number of basic values, principles and practices on top of the agile program

Activities

XP describes four basic activities that are performed within the software development process: coding, testing, listen

Coding

The advocates of XP argue that the only truly important product of the system development process is code - sof
working product.

Coding can also be used to figure out the most suitable solution. Coding can also help to communicate thoughts
programming problem and finding it hard to explain the solution to fellow programmers might code it and use the
this position, is always clear and concise and cannot be interpreted in more than one way. Other programmers can g

Testing

One can not be certain that a function works unless one tests it. Bugs and design errors are pervasive problems in so
testing can eliminate a few flaws, a lot of testing can eliminate many more flaws.

- Unit tests determine whether a given feature works as intended. A programmer writes as many automated tests
  successfully, then the coding is complete. Every piece of code that is written is tested before moving on to the ne
- Acceptance tests verify that the requirements as understood by the programmers satisfy the customer's actual re

A "testathon" is an event when programmers meet to do collaborative test writing, a kind of brainstorming relative

Listening

Programmers must listen to what the customers need the system to do, what "business logic" is needed. They m
about the technical aspects of how the problem might be solved, or cannot be solved. Communication between the c

Designing

From the point of view of simplicity, of course one could say that system development doesn't need more than codi
should always be a system that works. In practice, this will not work. One can come a long way without designing l
and the dependencies within the system cease to be clear. One can avoid this by creating a design structure t
dependencies within a system; this means that changing one part of the system will not affect other parts of the syst

Values

Extreme Programming initially recognized four values in 1999. A new value was added in the second edition of *Extr*

Communication

Building software systems requires communicating system requirements to the developers of the system. In formal
documentation. Extreme programming techniques can be viewed as methods for rapidly building and disseminatin
goal is to give all developers a shared view of the system which matches the view held by the users of the syste
metaphors, collaboration of users and programmers, frequent verbal communication, and feedback.

Simplicity

Extreme Programming encourages starting with the simplest solution. Extra functionality can then be added late
development methods is the focus on designing and coding for the needs of today instead of those of tomorrow, nex
gonna need it" (YAGNI) approach.[5] Proponents of XP acknowledge the disadvantage that this can sometimes ent
more than compensated for by the advantage of not investing in possible future requirements that might change l
requirements implies the risk of spending resources on something that might not be needed. Related to the "com
quality of communication. A simple design with very simple code could be easily understood by most programmers i

Feedback

Within extreme programming, feedback relates to different dimensions of the system development:

- Feedback from the system: by writing unit tests,[6] or running periodic integration tests, the programmers have
  changes.
- Feedback from the customer: The functional tests (aka acceptance tests) are written by the customer and the te
  system. This review is planned once in every two or three weeks so the customer can easily steer the developmen
- Feedback from the team: When customers come up with new requirements in the planning game the team direct

Feedback is closely related to communication and simplicity. Flaws in the system are easily communicated by writi
feedback from the system tells programmers to recode this part. A customer is able to test the system periodically
quote Kent Beck, "Optimism is an occupational hazard of programming, feedback is the treatment."[*citation needed*]

Courage

Several practices embody courage. One is the commandment to always design and code for today and not for to
requiring a lot of effort to implement anything else. Courage enables developers to feel comfortable with refactoring
and modifying it so that future changes can be implemented more easily. Another example of courage is knowing wl
no matter how much effort was used to create that source code. Also, courage means persistence: A programmer
problem quickly the next day, if only they are persistent.

Respect

The respect value includes respect for others as well as self-respect. Programmers should never commit changes tha
delay the work of their peers. Members respect their own work by always striving for high quality and seeking for th

Adopting the four earlier values leads to respect gained from others in the team. Nobody on the team should feel
encourages loyalty toward the team and toward the goal of the project. This value is very dependent upon the other

Rules

The first version of rules for XP was published in 1999 by Don Wells[10] at the XP website. 29 rules are given i
Planning, managing and designing are called out explicitly to counter claims that XP doesn't support those activitie

Another version of XP rules was proposed by Ken Auer[11] in XP/Agile Universe 2003. He felt XP was defined l
ambiguity). He defined two categories: "Rules of Engagement" which dictate the environment in which software de
the minute-by-minute activities and rules within the framework of the Rules of Engagement.

Principles

The principles that form the basis of XP are based on the values just described and are intended to foster decisio
more concrete than the values and more easily translated to guidance in a practical situation.

Feedback

Extreme programming sees feedback as most useful if it is done rapidly and expresses that the time between an ac
traditional system development methods, contact with the customer occurs in more frequent iterations. The custor
can give feedback and steer the development as needed.

Unit tests also contribute to the rapid feedback principle. When writing code, the unit test provides direct feedb
instance, the changes affect a part of the system that is not in the scope of the programmer who made them, that
bug will appear when the system is in production.

Assuming simplicity

This is about treating every problem as if its solution were "extremely simple". Traditional system development n programming rejects these ideas.

The advocates of extreme programming say that making big changes all at once does not work. Extreme programmi releases every three weeks. When many little steps are made, the customer has more control over the development p

Embracing change

The principle of embracing change is about not working against changes but embracing them. For instance, if at o have changed dramatically, programmers are to embrace this and plan the new requirements for the next iteration.

Practices

Extreme programming has been described as having 12 practices, grouped into four areas:

Fine scale feedback

- Pair programming[6]
- Planning game
- Test-driven development
- Whole team

Continuous process

- Continuous integration
- Refactoring or design improvement[6]
- Small releases

Shared understanding

- Coding standards
- Collective code ownership[6]
- Simple design[6]
- System metaphor

Programmer welfare

- Sustainable pace

Coding

- The customer is always available
- Code the Unit test first
- Only one pair integrates code at a time
- Leave Optimization till last
- No Overtime

Testing

- All code must have Unit tests
- All code must pass all Unit tests before it can be released.
- When a Bug is found tests are created before the bug is addressed (a bug is not an error in logic, it is a test you
- Acceptance tests are run often and the results are published

Controversial aspects

The practices in XP have been heavily debated.[6] Proponents of extreme programming claim that by having th flexible, and saves the cost of formal overhead. Critics of XP claim this can lead to costly rework and project scope

Change control boards are a sign that there are potential conflicts in project objectives and constraints between programmers being able to assume a unified client viewpoint so the programmer can concentrate on coding rather t applies when multiple programming organizations are involved, particularly organizations which compete for shares

Other potentially controversial aspects of extreme programming include:

- Requirements are expressed as automated acceptance tests rather than specification documents.
- Requirements are defined incrementally, rather than trying to get them all in advance.
- Software developers are usually required to work in pairs.
- There is no Big Design Up Front. Most of the design activity takes place on the fly and incrementally, starting v complexity only when it's required by failing tests. Critics compare this to "debugging a system into appearance" when requirements change.
- A customer representative is attached to the project. This role can become a single-point-of-failure for the projec the danger of micro-management by a non-technical representative trying to dictate the use of technical softwar
- Dependence upon all other aspects of XP: "XP is like a ring of poisonous snakes, daisy-chained together. All it t poisonous snake heading your way."[12]

Scalability

Historically, XP only works on teams of twelve or fewer people. One way to circumvent this limitation is to break has been claimed that XP has been used successfully on teams of over a hundred developers[*citation needed*]. Th with up to sixty people[*citation needed*].

In 2004 Industrial Extreme Programming (IXP)[13] was introduced as an evolution of XP. It is intended to bring th and flexible values. As it is a new member of the Agile family, there is not enough data to prove its usability, howev

Severability and responses

In 2003, Matt Stephens and Doug Rosenberg published *Extreme Programming Refactored: The Case Against XP* which it could be improved. This triggered a lengthy debate in articles, internet newsgroups, and web-site ch interdependent but that few practical organizations are willing/able to adopt all the practices; therefore the ent likeness of XP's "collective ownership" model to socialism in a negative manner.

Certain aspects of XP have changed since the book *Extreme Programming Refactored* (2003) was published; in par the required objectives are still met. XP also uses increasingly generic terms for processes. Some argue that these watering the process down.

RDP Practice is a technique for tailoring extreme programming. This practice was initially proposed as a long re Adolph(See APSO workshop (http://www.lero.ie/apso08/introduction.html) at ICSE 2008 (http://icse08.upb.de/)) XP. The valuable concepts behind RDP practice, in a short time provided the rationale for applicability of it in ind Rules.

Other authors have tried to reconcile XP with the older methods in order to form a unified methodology. Some Project Lifecycles: Waterfall, Rapid Application Development, and All That (http://www.lux-seattle.com/resource XP with the computer programming methodologies of Capability Maturity Model Integration (CMMI), and Six leading to better development, and did not mutually contradict.[14]

Criticism

Extreme programming's initial buzz and controversial tenets, such as pair programming and continuous design McBreen[15] and Boehm and Turner.[16] Many of the criticisms, however, are believed by Agile practitioners to be r

In particular, extreme programming is reviewed and critiqued by Matt Stephens's and Doug Rosenberg's *Extreme P*

Criticisms include:

- A methodology is only as effective as the people involved, Agile does not solve this
- Often used as a means to bleed money from customers through lack of defining a deliverable
- Lack of structure and necessary documentation
- Only works with senior-level developers
- Incorporates insufficient software design
- Requires meetings at frequent intervals at enormous expense to customers
- Requires too much cultural change to adopt
- Can lead to more difficult contractual negotiations
- Can be very inefficient—if the requirements for one area of code change through various iterations, the same pro were there to be followed, a single area of code is expected to be written once.
- Impossible to develop realistic estimates of work effort needed to provide a quote, because at the beginning of th
- Can increase the risk of scope creep due to the lack of detailed requirements documentation
- Agile is feature driven; non-functional quality attributes are hard to be placed as user stories

References

1. "Human Centred Technology Workshop 2005", 2005, PDF webpage: Informatics-UK-report-cdrp585 (ftp://ftp.in
2. "Design Patterns and Refactoring", University of Pennsylvania, 2003, webpage: UPenn-Lectures-design-patterns patterns.ppt).
3. "Extreme Programming" (lecture paper), USFCA.edu, webpage: USFCA-edu-601-lecture (http://www.cs.usfca.e
4. "Manifesto for Agile Software Development", Agile Alliance, 2001, webpage: Manifesto-for-Agile-Software-Dev (h
5. "Everyone's a Programmer" by Clair Tristram. *Technology Review*, Nov 2003. p. 39
6. "Extreme Programming", *Computerworld* (online), December 2001, webpage: Computerworld-appdev-92 (http:// 1,66192,00.html).
7. *Extreme Programming Refactored: The Case Against XP*. ISBN 1590590961.
8. Brodie, Leo (1984) (paperback). *Thinking Forth*. Prentice-Hall. ISBN 0-13-917568-7. http://thinking-forth.source
9. Interview with Kent Beck and Martin Fowler (http://www.informit.com/articles/article.aspx?p=20972)
10. Don Wells (http://www.extremeprogramming.org/rules.html)
11. Ken Auer (http://www.rolemodelsoftware.com/moreAboutUs/publications/rulesOfXp.php)
12. The Case Against Extreme Programming: A Self-Referential Safety Net (http://www.softwarereality.com/lifecyc
13. Cutter Consortium :: Industrial XP: Making XP Work in Large Organizations (http://www.cutter.com/content- earch/apmr0502.html)
14. Extreme Programming (XP) Six Sigma CMMI (http://www.sei.cmu.edu/library/assets/jarvis-gristock.pdf).
15. McBreen, P. (2003). *Questioning Extreme Programming*. Boston, MA: Addison-Wesley. ISBN 0-201-84457-5.
16. Boehm, B.; R. Turner (2004). *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston, MA: Addiso
17. sdmagazine (http://www.sdmagazine.com/documents/s=1811/sdm0112h/0112h.htm)
18. Extreme Programming Refactored (http://www.softwarereality.com/ExtremeProgrammingRefactored.jsp), Matt

Further reading

- Ken Auer and Roy Miller. *Extreme Programming Applied: Playing To Win*, Addison-Wesley.
- Kent Beck: *Extreme Programming Explained: Embrace Change*, Addison-Wesley.
- Kent Beck and Martin Fowler: *Planning Extreme Programming*, Addison-Wesley.
- Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change, Second Edition*, Addison-W
- Alistair Cockburn: *Agile Software Development*, Addison-Wesley.
- Martin Fowler: *Refactoring: Improving the Design of Existing Code*, Addison-Wesley.
- Harvey Herela (2005). Case Study: The Chrysler Comprehensive Compensation System (http://calla.ics.uci.edu/
- Jim Highsmith. *Agile Software Development Ecosystems*, Addison-Wesley.
- Ron Jeffries, Ann Anderson and Chet Hendrickson (2000), *Extreme Programming Installed*, Addison-Wesley.
- Mehdi Mirakhorli (2008). *RDP technique: a practice to customize xp*, International Conference on Software Engi agile practices or shoot-out at the agile corral, Leipzig, Germany 2008, Pages 23–32.
- Craig Larman & V. Basili (2003). "Iterative and Incremental Development: A Brief History", Computer (IEEE C
- Matt Stephens and Doug Rosenberg (2003). *Extreme Programming Refactored: The Case Against XP*, Apress.
- Waldner, JB. (2008). "Nanocomputers and Swarm Intelligence". In: ISTE, 225-256.

External links

- Extreme Programming
- A gentle introduction (http://www.extremeprogramming.org)
- Industrial eXtreme Programming (http://www.IndustrialXP.org/)
- XP magazine (http://www.xprogramming.com)
- Problems and Solutions to XP implementation (http://c2.com/cgi/wiki?ExtremeProgrammingImplementationIs
- Using an Agile Software Process with Offshore Development (http://www.martinfowler.com/articles/agileOffsho distributed projects

Planning

Requirements

Requirements analysis in systems engineering and software engineering, encompasses those tasks that go into to meet for a new or altered product, taking account of the possibly conflicting requirements of the various stakeh Another requirement you need to have to be a software manager you need to know how to pleasure your boss pleasure your boss.

Requirements analysis is critical to the success of a development project.[2] Requirements must be documented, act to identified business needs or opportunities, and defined to a level of detail sufficient for system design. Requirem behavioral, functional, and non-functional.

Overview

Conceptually, requirements analysis includes three types of activity:

- Eliciting requirements: the task of communicating with customers and users to determine what their requiremen requirements gathering.
- Analyzing requirements: determining whether the stated requirements are unclear, incomplete, ambiguous, or co issues.
- Recording requirements: Requirements might be documented in various forms, such as natural-language docume process specifications.

Requirements analysis can be a long and arduous process during which many delicate psychological skills are inv people, so it is important to identify all the stakeholders, take into account all their needs and ensure they unders techniques to elicit the requirements from the customer. Historically, this has included such things as holding int requirements workshops) and creating requirements lists. More modern techniques include prototyping, and use c methods to establish the exact requirements of the stakeholders, so that a system that meets the business needs is p

Requirements engineering

Systematic requirements analysis is also known as *requirements engineering*.[3] It is sometimes referred to loose requirements specification. The term *requirements analysis* can also be applied specifically to the analysis prope instance. Requirements Engineering can be divided into discrete chronological steps:

- Requirements elicitation,
- Requirements analysis and negotiation,
- Requirements specification,
- System modeling,
- Requirements validation,
- Requirements management.

Requirement engineering according to Laplante (2007) is "a subdiscipline of systems engineering and software en constraints of hardware and software systems."[4] In some life cycle models, the requirement engineering process be If the feasibility study suggests that the product should be developed, then requirement analysis can begin. If requir box thinking, then feasibility should be determined before requirements are finalized.

Requirements analysis topics

Stakeholder identification

See Stakeholder analysis for a discussion of business uses. Stakeholders (SH) are people or organizations (legal entit the system. They may be affected by it either directly or indirectly. A major new emphasis in the 1990s was a focu stakeholders are not limited to the organization employing the analyst. Other stakeholders will include:

- anyone who operates the system (normal and maintenance operators)
- anyone who benefits from the system (functional, political, financial and social beneficiaries)
- anyone involved in purchasing or procuring the system. In a mass-market product organization, product manage market customers) to guide development of the product
- organizations which regulate aspects of the system (financial, safety, and other regulators)
- people or organizations opposed to the system (negative stakeholders; see also Misuse case)
- organizations responsible for systems which interface with the system under design
- those organizations who integrate horizontally with the organization for whom the analyst is designing the syste

Stakeholder interviews

Stakeholder interviews are a common technique used in requirement analysis. Though they are generally idiosyncra the stakeholder, very often without larger enterprise or system context, this perspective deficiency has the general a unique business processes, decision-relevant business rules, and perceived needs. Consequently this technique can s not elicited in Joint Requirements Development sessions, where the stakeholder's attention is compelled to assume interviews provides a more relaxed environment where lines of thought may be explored at length.

Joint Requirements Development (JRD) Sessions

Requirements often have cross-functional implications that are unknown to individual stakeholders and often miss functional implications can be elicited by conducting JRD sessions in a controlled environment, facilitated by a tra requirements, analyze their details and uncover cross-functional implications. A dedicated scribe and Business Analy trained facilitator to guide the discussion frees the Business Analyst to focus on the requirements definition process.

JRD Sessions are analogous to Joint Application Design Sessions. In the former, the sessions elicit requirements tha implemented in satisfaction of elicited requirements.

Contract-style requirement lists

One traditional way of documenting requirements has been contract style requirement lists. In a complex system suc

An appropriate metaphor would be an extremely long shopping list. Such lists are very much out of favour in mode their aims; but they are still seen to this day.

Strengths

- Provides a checklist of requirements.
- Provide a contract between the project sponsor(s) and developers.
- For a large system can provide a high level description.

Weaknesses

- Such lists can run to hundreds of pages. It is virtually impossible to read such documents as a whole and have a
- Such requirements lists abstract all the requirements and so there is little context
    - This abstraction makes it impossible to see how the requirements fit or work together.
    - This abstraction makes it difficult to prioritize requirements properly; while a list does make it easy to priori an entire use case or business requirement useless.
    - This abstraction increases the likelihood of misinterpreting the requirements; as more people read them, the
    - This abstraction means that it's extremely difficult to be sure that you have the majority of the requirement they say, is in the details.
- These lists create a false sense of mutual understanding between the stakeholders and developers.
- These contract style lists give the stakeholders a false sense of security that the developers must achieve certain out crucial requirements which are identified later in the process. Developers can use these discovered requireme
- These requirements lists are no help in system design, since they do not lend themselves to application.

Alternative to requirement lists

As an alternative to the large, pre-defined requirement lists Agile Software Development uses User stories to define a

Measurable goals

Best practices take the composed list of requirements merely as clues and repeatedly ask "why?" until the actual devise tests to measure what level of each goal has been achieved thus far. Such goals change more slowly than th critical, measured goals has been established, rapid prototyping and short iterative development phases may proceed

Prototypes

In the mid-1980s, prototyping was seen as the best solution to the requirements analysis problem. Prototypes application that hasn't yet been constructed. Prototypes help users get an idea of what the system will look like, a the system to be built. Major improvements in communication between users and developers were often seen with changes later and hence reduced overall costs considerably.

However, over the next decade, while proving a useful technique, prototyping did not solve the requirements proble

- Managers, once they see a prototype, may have a hard time understanding that the finished design will not be p
- Designers often feel compelled to use patched together prototype code in the real system, because they are afraid
- Prototypes principally help with design decisions and user interface design. However, they can not tell you what
- Designers and end-users can focus too much on user interface design and too little on producing a system that se
- Prototypes work well for user interfaces, screen layout and screen flow but are not so useful for batch or asynchr calculations.

Prototypes can be flat diagrams (often referred to as wireframes) or working applications using synthesized functio and often remove all color from the design (i.e. use a greyscale color palette) in instances where the final software confusion over the final visual look and feel of the application.

Use cases

A use case is a technique for documenting the potential requirements of a new system or software change. Each use interact with the end-user or another system to achieve a specific business goal. Use cases typically avoid technical j Use cases are often co-authored by requirements engineers and stakeholders.

Use cases are deceptively simple tools for describing the behavior of software or systems. A use case contains a tex with the software or system. Use cases do not describe any internal workings of the system, nor do they explain hc user follows to perform a task. All the ways that users interact with a system can be described in this manner.

Software requirements specification

A software requirements specification (SRS) is a complete description of the behavior of the system to be developec the users will have with the software. Use cases are also known as functional requirements. In addition to use cases Non-functional requirements are requirements which impose constraints on the design or implementation (such as pe

Recommended approaches for the specification of software requirements are described by IEEE 830-1998. This stan software requirements specification.

⇔== Types of Requirements == Requirements are categorized in several ways. The following are common categori

Customer Requirements

Statements of fact and assumptions that define the expectations of the system in terms of mission objectives, en (MOE/MOS). The customers are those that perform the eight primary functions of systems engineering, with sp requirements will define the basic need and, at a minimum, answer the questions posed in the following listing:[1]

- *Operational distribution or deployment*: Where will the system be used?
- *Mission profile or scenario*: How will the system accomplish its mission objective?
- *Performance and related parameters*: What are the critical system parameters to accomplish the mission?
- *Utilization environments*: How are the various system components to be used?
- *Effectiveness requirements*: How effective or efficient must the system be in performing its mission?
- *Operational life cycle*: How long will the system be in use by the user?
- *Environment*: What environments will the system be expected to operate in an effective manner?

Architectural Requirements

Architectural requirements explain what has to be done by identifying the necessary system architecture of a sys

Structural Requirements

Structural requirements explain what has to be done by identifying the necessary structure of a system.

Behavioral Requirements

Behavioral requirements explain what has to be done by identifying the necessary behavior of a system.

Functional Requirements

Functional requirements explain what has to be done by identifying the necessary task, action or activity that m the toplevel functions for functional analysis.[1]

Non-functional Requirements

Non-functional requirements are requirements that specify criteria that can be used to judge the operation of a s

Performance Requirements

The extent to which a mission or function must be executed; generally measured in terms of quantity, quality, c performance (how well does it have to be done) requirements will be interactively developed across all identified of the degree of certainty in their estimate, the degree of criticality to system success, and their relationship to o

Design Requirements

The “build to,” “code to,” and “buy to” requirements for products and “how to execute” requirements for proces

Derived Requirements

Requirements that are implied or transformed from higher-level requirement. For example, a requirement for lon weight.[1]

Allocated Requirements

A requirement that is established by dividing or otherwise allocating a high-level requirement into multiple lowe subsystems might result in weight requirements of 70 pounds and 30 pounds for the two lower-level items.[1]

Well-known requirements categorization models include FURPS and FURPS+, developed at Hewlett-Packard.

Requirements analysis issues

Stakeholder issues

Steve McConnell, in his book *Rapid Development*, details a number of ways users can inhibit requirements gathering

- Users do not understand what they want or users don't have a clear idea of their requirements
- Users will not commit to a set of written requirements
- Users insist on new requirements after the cost and schedule have been fixed
- Communication with users is slow
- Users often do not participate in reviews or are incapable of doing so
- Users are technically unsophisticated
- Users do not understand the development process
- Users do not know about present technology

This may lead to the situation where user requirements keep changing even when system or product development ha

Engineer/developer issues

Possible problems caused by engineers and developers during requirements analysis are:

- Technical personnel and end-users may have different vocabularies. Consequently, they may wrongly believe the
- Engineers and developers may try to make the requirements fit an existing system or model, rather than develop
- Analysis may often be carried out by engineers or programmers, rather than personnel with the people skills and

Attempted solutions

One attempted solution to communications problems has been to employ specialists in business or system analysis.

Techniques introduced in the 1990s like prototyping, Unified Modeling Language (UML), use cases, and Agile softw with previous methods.

Also, a new class of application simulation or application definition tools have entered the market. These tools are the IT organization — and also to allow applications to be 'test marketed' before any code is produced. The best of

- electronic whiteboards to sketch application flows and test alternatives
- ability to capture business logic and data needs
- ability to generate high fidelity prototypes that closely imitate the final application
- interactivity
- capability to add contextual requirements and other comments
- ability for remote and distributed users to run and interact with the simulation

References

1. *Systems Engineering Fundamentals.* (http://www.dau.mil/pubscats/PubsCats/SEFGuide%2001-01.pdf) Defense
2. Executive editors: Alain Abran, James W. Moore; editors Pierre Bourque, Robert Dupuis, ed (March 2005). "Ch *body of knowledge* (2004 ed.). Los Alamitos, CA: IEEE Computer Society Press. ISBN 0-7695-2330-7. http://ww "It is widely acknowledged within the software industry that software engineering projects are critically vulnerab
3. Wiegers, Karl E. (2003). *Software Requirements* (2nd ed.). Redmond, WA: Microsoft Press. ISBN 0-7356-1879-8
4. Phillip A. Laplante (2007) *What Every Engineer Should Know about Software Engineering*. Page 44.

Further reading

- Laplante, Phil (2009). *Requirements Engineering for Software and Systems* (1st ed.). Redmond, WA: CRC Press http://beta.crcpress.com/product/isbn/9781420064674.
- McConnell, Steve (1996). *Rapid Development: Taming Wild Software Schedules* (1st ed.). Redmond, WA: Micros
- Wiegers, Karl E. (2003). *Software Requirements* (2nd ed.). Redmond, WA: Microsoft Press. ISBN 0-7356-1879-8
- Andrew Stellman and Jennifer Greene (2005). *Applied Software Project Management*. Cambridge, MA: O'Reilly
- Brian Berenbach, Daniel Paulish, Juergen Katzmeier, Arnold Rudorfer (2009). *Software & Systems Requirement* ISBN 0-07-1605479. http://www.mhprofessional.com.
- Walter Sobkiw (2008). *Sustainable Development Possible with Creative System Engineering*. New Jersey: CassBe http://www.amazon.com/exec/obidos/ASIN/0615216307.
- Walter Sobkiw (2011). *Systems Practices as Common Sense*. New Jersey: CassBeth. ISBN 978-0983253082. http

Sense/dp/0983253080.

External links

- Software Requirement Analysis using UML (http://www.slideshare.net/dhirajmusings/software-requirement-ana hiraj-shetty/6/472/165).
- Requirements Engineering Process "Goodies" (http://www.processimpact.com/goodies.shtml#reqs)
- Requirements Engineering: A Roadmap (http://www.cs.toronto.edu/~sme/papers/2000/ICSE2000.pdf) (PDF) a

Requirements Management

Requirements management is the process of documenting, analyzing, tracing, prioritizing and agreeing on re stakeholders. It is a continuous process throughout a project. A requirement is a capability to which a project outco

Overview

The purpose of requirements management is to assure the organization documents, verifies and meets the needs an Requirements management begins with the analysis and elicitation of the objectives and constraints of the organizat requirements, integrating requirements and the organization for working with them (attributes for requirement requirements, and changes for these.

The traceability thus established is used in managing requirements to report back fulfillment of company and sta consistency. Traceabilities also support change management as part of requirements management in understanding (e.g., functional impacts through relations to functional architecture), and facilitating introducing these changes.[2]

Requirements management involves communication between the project team members and stakeholders, and adju To prevent one class of requirements from overriding another, constant communication among members of the d internal applications, the business has such strong needs that it may ignore user requirements, or believe that in cre

Traceability

Requirements traceability is concerned with documenting the life of a requirement. It should be possible to trace requirement should therefore be documented in order to achieve traceability. Even the use of the requirement a traceable[4].

Requirements come from different sources, like the business person ordering the product, the marketing manager a product. Using requirements traceability, an implemented feature can be traced back to the person or group that used during the development process to prioritize the requirement, determining how valuable the requirement is studies show that a feature is not used, to see why it was required in the first place.

Requirements activities

At each stage in a development process, there are key requirements management activities and methods. To Investigation, Feasibility, Design, Construction and Test, and Release stages.

Investigation

In Investigation, the first three classes of requirements are gathered from the users, from the business and from the the goals, what are the constraints, what are the current tools or processes in place, and so on. Only when t developed.

A caveat is required here: no matter how hard a team tries, requirements cannot be fully defined at the beginning o weren't extracted, or because internal or external forces at work affect the project in mid-cycle. Thus, the team r flexibility in thinking and operation.

The deliverable from the Investigation stage is a requirements document that has been approved by all members critical in preventing scope creep or unnecessary changes. As the system develops, each new feature opens a world o the original vision and permits a controlled discussion of scope change.

While many organizations still use only documents to manage requirements, others manage their requirements basel in a database, and usually have functions to automate traceability (e.g., by allowing electronic links to be creat requirements), electronic baseline creation, version control, and change management. Usually such tools contain a exporting the requirements data into a standard document application.

Feasibility

In the Feasibility stage, costs of the requirements are determined. For user requirements, the current cost of work is Questions such as these are asked: "What are data entry errors costing us now?" Or "What is the cost of scrap due new tool is often recognized as these questions come to the attention of financial people in the organization.

Business costs would include, "What department has the budget for this?" "What is the expected rate of return return in reducing costs of training and support if we make a new, easier-to-use system?"

Technical costs are related to software development costs and hardware costs. "Do we have the right people to crea roles?" This last question is an important type. The team must inquire into whether the newest automated tools w user to the system in order to save people time.

The question also points out a fundamental point about requirements management. A human and a tool form a syst or a new application on a computer. The human mind excels in parallel processing and interpretation of trends w mathematical computation. The overarching goal of the requirements management effort for a software project wou

proper processor. For instance, “Don’t make the human remember where she is in the interface. Make the interface r
the human enter the same data in two screens. Make the system store the data and fill in the second screen as neede

The deliverable from the Feasibility stage is the budget and schedule for the project.

Design

Assuming that costs are accurately determined and benefits to be gained are sufficiently large, the project can proc
activity is comparing the results of the design against the requirements document to make sure that work is staying

Again, flexibility is paramount to success. Here’s a classic story of scope change in mid-stream that actually wor
prices to hit $3.18 per gallon by the end of the decade. Midway through the design of the Ford Taurus, prices had
build a larger, more comfortable, and more powerful car if the gas prices stayed low, so they redesigned the car. T
out, primarily because it was so roomy and comfortable to drive.

In most cases, however, departing from the original requirements to that degree does not work. So the requiremen
about design changes.

Construction and test

In the construction and testing stage, the main activity of requirements management is to make sure that work and
in fact meet requirements. A main tool used in this stage is prototype construction and iterative testing. For a soft
with potential users while the framework of the software is being built. Results of these tests are recorded in a user
ready to develop the interface. This saves their time and makes their jobs much easier.

Release

Requirements management does not end with product release. From that point on, the data coming in about the ap
of the next generation or release. Thus the process begins again.

Tools

There exist both desktop and Web-based tools for requirements management. A Web-based requirements tool ca
demand requirements management platform which in some cases is completely free. There used to be a list of such t
it in 2015.[5]

Modeling Languages

The system engineering modeling language SysML incorporates a requirements diagram allowing the developer to gr

On-demand requirements management platforms

An on-demand requirements management platform is a fully hosted requirements management solution, where t
standard Web browser.

The service would normally include all special hardware and software. Other services may include technology
unauthorized use, 24×7 data availability, and assurance that the service will scale as you add users, applications, an

Some on-demand requirements management platforms charge a fee while others are free to use.

References

1. Stellman, Andrew; Greene, Jennifer (2005). *Applied Software Project Management*. O'Reilly Media. ISBN 978-0-
2. "Requirements management". UK Office of Government Commerce. http://www.ogc.gov.uk/delivery_lifecycle_r
3. *A Guide to the Project Management Body of Knowledge* (4th ed.). Project Management Institute. 2008. ISBN 97
4. Gotel, O., Finkelstein, A. An Analysis of the Requirements Traceability Problem Proc. of First International Co
5. "Requirements Management Tools Survey". International Council on Systems Engineering. Archived from the or
https://web.archive.org/web/20150319021445/http://www.incose.org/ProductsPubs/products/rmsurvey.aspx. R

Further reading

- CMMI Product Team (August 2006) (PDF). *CMMI for Development, Version 1.2*. Technical Report CMU/SEI-
http://www.sei.cmu.edu/library/abstracts/reports/06tr008.cfm. Retrieved 2008-01-22.
- Colin Hood, Simon Wiedemann, Stefan Fichtinger, Urte Pautz *Requirements Management: Interface Between R*
Springer, Berlin 2007, ISBN 354047689X

External links

- Washington State Information Services Board (ISB)policy: CMM Key Practices for Level 2 - Requirements Man
- U.K. Office of Government Commerce (OGC) - Requirements management (http://www.ogc.gov.uk/delivery_li

Specification

A **functional specification** (also, *functional spec*, *specs*, *functional specifications document (FSD)*, or *Program* documentation that describes the requested behavior of an engineering system. The documentation typically descril user as well as requested properties of inputs and outputs (e.g. of the software system).

Overview

In systems engineering a specification is a document that clearly and accurately describes the essential technical req services including the procedures by which it can be determined that the requirements have been met. Specifica inconsistencies, allow for accurate estimates of necessary work and resources, act as a negotiation and reference dc provide documentation of configuration, and allow for consistent communication among those responsible for the eig Engineering. They provide a precise idea of the problem to be solved so that they can efficiently design the system alternatives. They provide guidance to testers for verification (qualification) of each technical requirement.[1]

A functional specification does not define the inner workings of the proposed system; it does not include the speci will be implemented. Instead, it focuses on what various outside agents (people using the program, computer per example) might "observe" when interacting with the system. A typical functional specification might state the follow

> *When the user clicks the OK button, the dialog is closed and the focus is returned to the main window in the sta was displayed.*

Such a requirement describes an interaction between an external agent (the user) and the software system. Whe system by clicking the OK button, the program responds (or should respond) by closing the dialog window containi

It can be *informal*, in which case it can be considered as a blueprint or user manual from a developer point of view, definite meaning defined in mathematical or programmatic terms. In practice, most successful specifications are wri applications that were already well-developed, although safety-critical software systems are often carefully development. Specifications are most important for external interfaces that must remain stable.

Functional specification topics

Purpose

There are many purposes for functional specifications. One of the primary purposes on team projects is to achieve what the program is to achieve before making the more time-consuming effort of writing source code and test debugging. Typically, such consensus is reached after one or more reviews by the stakeholders on the project at han effective way to achieve the requirements the software needs to fulfill.

Process

In the ordered industrial software engineering life-cycle (waterfall model), functional specification describes what ha how the functions will be realized using a chosen software environment. In not industrial, prototypical systems dev of requirements analysis.

When the team agrees that functional specification consensus is reached, the functional spec is typically declared "f and testing team write source code and test cases using the functional specification as the reference. While testi expected behavior as defined in the functional specification.

Types of software development specifications

- Advanced Microcontroller Bus Architecture
- Bit specification
- Design specification
- Diagnostic design specification
- Multiboot Specification
- Product design specification
- Real-time specification for Java
- Software Requirements Specification

References

1. *Systems Engineering Fundamentals.* (http://www.dau.mil/pubscats/PubsCats/SEFGuide%2001-01.pdf) Defense

External links

- Writing functional specifications Tutorial (http://www.mojofat.com/tutorial/)
- Painless Functional Specifications, 4-part series by Joel Spolsky (http://www.joelonsoftware.com/articles/fog000

Architecture & Design

Introduction

Introduction

When you build your house, you would never think about building it without an architect, correct? However, ma
architect. That seems kind of scary, and you might wonder why? Well, the role of the software architect has neithe
date there is still no agreement on the precise definition of the term “software architecture”.[1]

Matthew R. McBride writes, "a software architect is a technically competent system-level thinker, guiding planne
viewed by customers and developers alike as a technical expert. The architect is the author of the solution, accou
refers to documentation of a system's software architecture. Documenting software architecture facilitates communi
design, and allows reuse of design components and patterns between projects.[3]

Architecture and Design

Software architecture, also described as strategic design, is an activity concerned with global requirements gover
architectural styles, component-based software engineering standards, architectural patterns, security, scale, integra
tactical design, is an activity concerned with local requirements governing *what* a solution does such as algorit
implementation.

Architecture is design but not all design is architectural.[4] In practice, the architect is the one who draws the line
(non-architectural design). There aren't rules or guidelines that fit all cases. Examples of rules or heuristics that a
between architecture and detailed design include:

- Architecture is driven by non-functional requirements, while functional design is driven by functional requiremer
- Pseudo-code belongs in the detailed design document.
- UML component, deployment, and package diagrams generally appear in software architecture documents; UML
  design documents.

Software Architecture

The field of computer science has come across problems associated with complexity since its formation.[5] Earlier p
data structures, developing algorithms, and by applying the concept of separation of concerns. Although the term “s
principles of the field have been applied sporadically by software engineering pioneers since the mid 1980s. Early a
imprecise and disorganized, often characterized by a set of box-and-line diagrams.[6] During the 1990s there was
discipline. Initial sets of design patterns, styles, best practices, description languages, and formal logic were develope

As a maturing discipline with no clear rules on the right way to build a system, designing software architecture is s
because a commercial software system supports some aspect of a business or a mission. How a system suppo
requirements of a system, also known as quality attributes, determine how a system will behave.[7] Every system is
degree of quality attributes exhibited by a system such as fault-tolerance, backward compatibility, extensibility, rel
–ilities will vary with each implementation.[7]

The origin of software architecture as a concept was first identified in the research work of Edsger Dijkstra in 1968
the structure of a software system matters and getting the structure right is critical. The study of the field increas
on architectural styles (patterns), architecture description languages, architecture documentation, and formal metho

Views and UML

Although there exist 'architecture description languages' (see below), no consensus exists on which symbol-set or la
standard used regularly by architects. For instance, UML component, deployment, and package diagrams generally
language that is often being used to create software architecture views.

Software architecture views are analogous to the different types of blueprints made in building architecture. A vi
among them. [4] Some possible views are:

- Functional/logic view
- Code/module view
- Development/structural view
- Concurrency/process/runtime/thread view
- Physical/deployment/install view
- User action/feedback view
- Data view/data model

Architecture Frameworks

There are several architecture frameworks related to the domain of software architecture, most well known being th
(RM-ODP) and the Service-Oriented Modeling Framework (SOMF) are being used. Other architectures such as
Enterprise architecture.

Architecture Description Languages

Several languages for describing software architectures ('architecture description language' (ADL) in ISO/IEC 4
describe a Software Architecture. Several different ADLs have been developed by different organizations, including
(developed by Carnegie Mellon), xADL (developed by UCI), Darwin (developed by Imperial College London), DAO
L'Aquila, Italy). Common elements of an ADL are component, connector and configuration.

References

1. SEI (2006). "How do you define Software Architecture?". http://www.sei.cmu.edu/architecture/start/definitions.
2. McBride, Matthew R. (2004). *The software architect: essence, intuition, and guiding principles*. New York: ACM
3. Bass, Len; Paul Clements, Rick Kazman (2003). *Software Architecture In Practice, Second Edition*. Boston: Ad
4. Clements, Paul; Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord *and Beyond, Second Edition*. Boston: Addison-Wesley. ISBN 0321552687.
5. University of Waterloo (2006). "A Very Brief History of Computer Science". http://www.cs.uwaterloo.ca/~shallit
6. IEEE Transactions on Software Engineering (2006). "Introduction to the Special Issue on Software Architecture" resourcePath=/dl/trans/ts/&toc=comp/trans/ts/1995/04/e4toc.xml&DOI=10.1109/TSE.1995.10003. Retrieved
7. SoftwareArchitectures.com (2006). "Intro to Software Quality Attributes". http://www.softwarearchitectures.com
8. Garlan & Shaw (1994). "An Introduction to Software Architecture". http://www.cs.cmu.edu/afs/cs/project/able

Further Reading

- Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord *Beyond, Second Edition*. Addison-Wesley, 2010, ISBN 0321552687. This book describes what is software architec other notations. It also explains how to complement the architecture views with behavior, software interface, and contains an example of software architecture documentation (https://wiki.sei.cmu.edu/sad/index.php/The_Adv
- Len Bass, Paul Clements, Rick Kazman: *Software Architecture in Practice, Second Edition*. Addison Wesley, Re eloquently covers the fundamental concepts of the discipline. The theme is centered around achieving quality att
- Amnon H. Eden, Rick Kazman. *Architecture, Design, Implementation.* (http://www.eden-study.org/articles/200 detailed design.
- Garzás, Javier, and Piattini, Mario. An ontology for micro-architectural design knowledge, IEEE Software Maga
- Philippe Kruchten: *Architectural Blueprints - the 4+1 View Model of Software Architecture*. In: IEEE Software. website (http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/Pbk4p1.pdf)(PDI
- Tony Shan and Winnie Hua (2006). *Solution Architecting Mechanism* (http://doi.ieeecomputersociety.org/10.11 Enterprise Computing Conference (EDOC 2006), October 2006, p23-32
- SOMF: Bell, Michael (2008). "Service-Oriented Modeling: Service Analysis, Design, and Architecture". Wiley. ht Architecture/dp/0470141115/ref=pd_bbs_2.
- The IEEE 1471: ANSI/IEEE 1471-2000: Recommended Practice for Architecture Description of Software-Intensi architecture, and was adopted in 2007 by ISO as *ISO/IEC 42010:2007 (IEEE 1471)*.

External Links

- Excellent explanation on IBM Developerworks (http://www.ibm.com/developerworks/rational/library/feb06/eel
- Collection of software architecture definitions (http://www.sei.cmu.edu/architecture/start/definitions.cfm) at So
- Software architecture vs. software design: The Intension/Locality Hypothesis (http://www.eden-study.org/articl
- Worldwide Institute of Software Architects (WWISA) (http://www.wwisa.org/)
- International Association of Software Architects (IASA) (http://www.iasahome.org/iasaweb/appmanager/home/
- SoftwareArchitecturePortal.org (http://www.softwarearchitectureportal.org/) — website of IFIP Working Group
- Software Architecture (http://blog.softwarearchitecture.com/) — practical resources for Software Architects
- SoftwareArchitectures.com (http://www.softwarearchitectures.com/) — independent resource of information on
- Microsoft Architecture Journal (http://www.architecturejournal.net/)
- Architectural Patterns (http://www.jools.net/archives/44)
- Software Architecture (http://www.ics.uci.edu/~fielding/pubs/dissertation/software_arch.htm), chapter 1 of Ro
- DiaSpec (http://diaspec.bordeaux.inria.fr/), an approach and tool to generate a distributed framework from a sc
- When Good Architecture Goes Bad (http://www.methodsandtools.com/archive/archive.php?id=85)
- Software Architecture and Related Concerns (http://www.bredemeyer.com/whatis.htm), What is Software Arch
- Handbook of Software Architecture (http://www.handbookofsoftwarearchitecture.com/index.jsp?page=Main)
- The Spiral Architecture Driven Development (http://sadd.codeplex.com) - the SDLC based on Spiral model is t

- Rationale focused software architecture documentation method (http://gupea.ub.gu.se/bitstream/2077/10490/1/

Design

Software Design

The result of the software requirements analysis (SRA) usually is a specification. The design helps us turning this s
kinds of software designs, the IEEE Std 610.12-1990 Standard Glossary of Software Engineering Terminology[1] defi

- Architectural Design: the process of defining a collection of hardware and software components and their interfac
  system.
- Detailed Design: the process of refining and expanding the preliminary design of a system or component to the e
- Functional Design: the process of defining the working relationships among the components of a system.
- Preliminary Design: the process of analyzing design alternatives and defining the architecture, components, inter

Hence software design includes architectural views, but also low-level component and algorithm implementation independent or platform-specific.

Design Considerations

There are many aspects to consider in the design of a piece of software. The importance of each should reflect the go

- **Compatibility** - The software is able to operate with other products that are designed for interoperability wit
  compatible with an older version of itself.
- **Extensibility** - New capabilities can be added to the software without major changes to the underlying archite
- **Fault-tolerance** - The software is resistant to and able to recover from component failure.
- **Maintainability** - The software can be restored to a specified condition within a specified period of time. For
  receive virus definition updates in order to maintain the software's effectiveness.
- **Modularity** - the resulting software comprises well defined, independent components. That leads to better mai
  isolation before being integrated to form a desired software system. This allows division of work in a software de
- **Packaging** - Printed material such as the box and manuals should match the style designated for the target m
  should be visible on the outside of the package. All components required for use should be included in the packa
- **Reliability** - The software is able to perform a required function under stated conditions for a specified period
- **Reusability** - the software is able to add further features and modification with slight or no modification.
- **Robustness** - The software is able to operate under stress or tolerate unpredictable or invalid input. For exam
- **Security** - The software is able to withstand hostile acts and influences.
- **Usability** - The software user interface must be usable for its target user/audience. Default values for the para
  of the users.

Modeling Language

Designers are assisted be the existence of modeling languages. They can be used to express information, knowledge modeling language can be graphical or textual. Examples of graphical modelling languages for software design are:

- Unified Modeling Language (UML) is a general modeling language to describe software both structurally and bel
  Profile (UML).
- Flowchart is a schematic representation of an algorithm or a stepwise process,
- Business Process Modeling Notation (BPMN) is an example of a Process Modeling language.
- Systems Modeling Language (SysML) is a new general-purpose modeling language for systems engineering.

There is quite a few more, but we will concentrate mostly on the UML as we will see in the next chapter.

References

1. http://standards.ieee.org/findstds/standard/610.12-1990.html the IEEE Std 610.12-1990, IEEE standard glossar

2. Software Engineering[8th edition]-lan Sommerville publisher- Pearson

External Links

- IEEE Std 1016-1998 IEEE Recommended Practice for Software Design Descriptions (http://standards.ieee.org/r
- A Software Design Specification Template (http://www.cmcrossroads.com/bradapp/docs/sdd.html)

Design Patterns

Design Patterns

If you remember, software engineers speak a common language called UML. And if we use this analogy of language, instance fairy tales. They are stories about commonly occurring problems in software design and their solutions. And software engineers learn about good design (design patterns) and bad design (anti-patterns).

History

Patterns originated as an architectural concept by Christopher Alexander (1977/79). In 1987, Kent Beck and Ward programming and presented their results at the OOPSLA conference that year.[1][2] In the following years, Beck, Cu

Design patterns gained popularity in computer science after the book *Design Patterns: Elements of Reusable Obj* Four" (Gamma et al.).[3] That same year, the first Pattern Languages of Programming Conference was held an documentation of design patterns.

Definition of a Design Pattern

In software engineering, a design pattern is a general reusable solution to a commonly occurring problem in so transformed directly into code. It is a description or template for how to solve a problem that can be used in ma relationships and interactions between classes or objects, without specifying the final application classes or objects th

Design patterns reside in the domain of modules and interconnections. At a higher level there are architectural p followed by an entire system.[4]

There are many types of design patterns: Structural patterns address concerns related to the high level structu concerns related to the identification of key computations. Algorithm strategy patterns address concerns relat characteristic on a computation platform. Implementation strategy patterns address concerns related to the realiza and the common data structures specific to parallel programming. Execution patterns address concerns related to th executing streams of tasks and building blocks to support the synchronization between tasks.

Examples of Design Patterns

Design patterns are easiest understood when looking at concrete examples. For beginners the following ten patterns of the other patterns mentioned below. The more patterns you know, the better.

Factory Method

The Factory pattern creates an object from a set of similar classes, based on some parameter, usually a string. An e

```
MessageDigest md = MessageDigest.getInstance("SHA-1");
```

If one changes the parameter to "MD5" for instance, one gets an object that calculates the message digest based on changing the algorithm does not require us to re-compile our code. Other examples of this pattern a Class.forName("jdbc.idbDriver"), which admittedly is some very odd syntax, but the idea is the same.

Abstract Factory

Where the Factory pattern only affects one class, the Abstract Factory pattern affects a whole bunch of classes. A w

```
UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
```

This looks quite similar to the Factory pattern, but the difference is that now every Swing class that is being loaded

Singleton

This is one of the most dangerous design patterns, when in doubt don't use it. Its main purpose is to guarantee that a printer manager or a database connection manager. It is useful when access to a limited resource needs to be contr

Iterator

Nowadays, the Iterator pattern is trivial: it allows you to go through a list of objects, starting at the beginning, it end.

Template Method

Also the Template Method pattern is rather simple: as soon as you define an abstract class, that forces its subc Template pattern.

Command

To understand the idea behind the Command pattern consider the following restaurant example: A customer g (command, in this case) and hands it to the cook in the kitchen. In the kitchen the command is executed, and depe

Observer

The Observer pattern is one of the most popular patterns, and it has many variants. Assume you have a table in a of some graph or histogram. If the underlying data changes, not only the table view has to change, but you also e use the Observer pattern: the underlying data is the *observable* and the table view as well as the histogram vie Observer pattern is a button in Swing for instance: here the JButton is the observable, and if something happens listener (the observer) gets notified.

Composite

The Composite pattern is very wide spread. Basically, it is a list that may contain objects, but also lists. A typical directories may contain files, but also may contain other directories. Other examples of the Composite pattern are has users and groups, where groups may contain users, but also other groups.

State

In the State pattern, an internal state of the object influences its behavior. Assume you have some drawing progra Instead of creating different classes for lines, you have one Line class that has an internal state called 'dotted' o straight lines are drawn. This pattern is also implicitly used by Java, when setting the font via 'setFont()' or the col

Proxy

The idea behind the Proxy pattern is that we have some complex object and we need to make it simpler. One typic to give the impression as if the user is dealing with a local object. Another application is when an object would take object may never be needed. In this case a proxy represents the object until it is needed.

Patterns in Practice

Design patterns can speed up the development process by providing tested, proven development paradigms. Effe visible until later in the implementation. Reusing design patterns helps to prevent subtle issues that can cause architects who are familiar with the patterns. In addition to this, patterns allow developers to communicate using w

In order to achieve flexibility, design patterns usually introduce additional levels of indirection, which in some cases

By definition, a pattern must be programmed anew into each application that uses it. Since some authors see th researchers have worked to turn patterns into components. Meyer and Arnout were able to provide full or partial co

Classification and List of Patterns

Design patterns were originally grouped into the categories: creational patterns, structural patterns, and behaviora and consultation. Another classification has also introduced the notion of architectural design pattern that may be Controller pattern. The following patterns are taken from *Design Patterns* [3] and *Code Complete*,[6] unless otherwis

Creational patterns

- **Abstract factory:** Provide an interface for creating families of related or dependent objects without specifyin
- **Builder:** Separate the construction of a complex object from its representation allowing the same construction
- **Factory method:** Define an interface for creating an object, but let subclasses decide which class to instantia
- **Lazy initialization:** Tactic of delaying the creation of an object, the calculation of a value, or some other exp
- **Multiton:** Ensure a class has only named instances, and provide global point of access to them.
- **Object pool:** Avoid expensive acquisition and release of resources by recycling objects that are no longer in us pool patterns.
- **Prototype:** Specify the kinds of objects to create using a prototypical instance, and create new objects by copy
- **Resource acquisition is initialization:** Ensure that resources are properly released by tying them to the l
- **Singleton:** Ensure a class has only one instance, and provide a global point of access to it.

Structural Patterns

- **Adapter or Wrapper:** Convert the interface of a class into another interface clients expect. Adapter lets cla interfaces.
- **Bridge:** Decouple an abstraction from its implementation allowing the two to vary independently.
- **Composite:** Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients tr
- **Decorator:** Attach additional responsibilities to an object dynamically keeping the same interface. Decorators
- **Facade:** Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface
- **Front Controller:** Provide a unified interface to a set of interfaces in a subsystem. Front Controller defines a

- **Flyweight:** Use sharing to support large numbers of fine-grained objects efficiently.
- **Proxy:** Provide a surrogate or placeholder for another object to control access to it.

Behavioral Patterns

- **Blackboard:** Generalized observer, which allows multiple readers and writers. Communicates information syst
- **Chain of responsibility:** Avoid coupling the sender of a request to its receiver by giving more than one obje the request along the chain until an object handles it.
- **Command:** Encapsulate a request as an object, thereby letting you parameterize clients with different request
- **Interpreter:** Given a language, define a representation for its grammar along with an interpreter that uses the
- **Iterator:** Provide a way to access the elements of an aggregate object sequentially without exposing its underly
- **Mediator:** Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling l vary their interaction independently.
- **Memento:** Without violating encapsulation, capture and externalize an object's internal state allowing the obj
- **Null object:** Avoid null references by providing a default object.
- **Observer or Publish/subscribe:** Define a one-to-many dependency between objects where a state change automatically.
- **Servant:** Define common functionality for a group of classes
- **Specification:** Recombinable business logic in a boolean fashion
- **State:** Allow an object to alter its behavior when its internal state changes. The object will appear to change it
- **Strategy:** Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets t
- **Template method:** Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. T without changing the algorithm's structure.
- **Visitor:** Represent an operation to be performed on the elements of an object structure. Visitor lets you define it operates.

Concurrency Patterns

Most of the following concurrency patterns are taken from *POSA2*[8]

- **Active Object:** Decouples method execution from method invocation that reside in their own thread of contr invocation and a scheduler for handling requests.
- **Balking:** Only execute an action on an object when the object is in a particular state.
- **Binding Properties:** Combining multiple observers to force properties in different objects to be synchronized
- **Messaging pattern:** The messaging design pattern (MDP) allows the interchange of information (i.e. messag
- **Double-checked locking:** Reduce the overhead of acquiring a lock by first testing the locking criterion (the lock proceed. Can be unsafe when implemented in some language/hardware combinations. It can therefore somet
- **Event-based asynchronous:** Addresses problems with the Asynchronous pattern that occur in multithread
- **Guarded suspension:** Manages operations that require both a lock to be acquired and a precondition to be s
- **Lock:** One thread puts a "lock" on a resource, preventing other threads from accessing or modifying it.[11][7]
- **Monitor object:** An object whose methods are subject to mutual exclusion, thus preventing multiple objects
- **Reactor:** A reactor object provides an asynchronous interface to resources that must be handled synchronously
- **Read-write lock:** Allows concurrent read access to an object but requires exclusive access for write operation

- **Scheduler:** Explicitly control when threads may execute single-threaded code.
- **Thread pool:** A number of threads are created to perform a number of tasks, which are usually organized in a considered a special case of the object pool pattern.
- **Thread-specific storage:** Static or "global" memory local to a thread.

Data Access Patterns

Another interesting area where patterns have a wide application is the area of *data access patterns*. Clifton Nock [12

- **ORM Patterns:** Domain Object Factory, Object/Relational Map, Update Factory
- **Resource Management Patterns:** Resource Pool, Resource Timer, Retryer, Paging Iterator
- **Cache Patterns:** Cache Accessor, Demand Cache, Primed Cache, Cache Collector, Cache Replicator
- **Concurrency Patterns:** Transaction, Optimistic Lock, Pessimistic Lock

Enterprise Patterns

If you deal with J2EE or with .Net Enterprise applications, the problems that occur and the solutions to them a *J2EE Patterns* [13] lists these patterns:

- **Presentation Tier Patterns:** Intercepting Filter, Front Controller, View Helper, Composite View, Service t
- **Business Tier Patterns:** Business Delegate, Value Object, Session Facade, Composite Entity, Value Object
- **Integration Tier Patterns:** Data Access Object, Service Activator

Real-Time Patterns

Finally, in the area of real-time and embedded software development a vast number of patterns have been iden *Architecture for Real-Time Systems*[16][17] Bruce Powel Douglass lists some very intriguing patterns:

- **Architecture Patterns:** Layered Pattern, Channel Architecture Pattern, Component-Based Architecture, R Microkernel Architecture Pattern, Virtual Machine Pattern
- **Concurrency Patterns:** Message Queuing Pattern, Interrupt Pattern, Guarded Call Pattern, Rendezvous P
- **Memory Patterns:** Static Allocation Pattern, Pool Allocation Pattern, Fixed Sized Buffer Pattern, Smart P Pattern
- **Resource Patterns:** Critical Section Pattern, Priority Inheritance Pattern, Priority Ceiling Pattern, Simulta
- **Distribution Patterns:** Shared Memory Pattern, Remote Method Call Pattern, Observer Pattern, Data Bus
- **Safety and Reliability Patterns:** Monitor-Actuator Pattern, Sanity Check Pattern, Watchdog Pattern, S Homogeneous Redundancy Pattern, Triple Modular Redundancy Pattern, Heterogeneous Redundancy Pattern

Efforts have also been made to codify design patterns in particular domains, including use of existing design patt interface design patterns,[18] information visualization [19], secure design[20], "secure usability"[21], web design Programming Conference proceedings [24] include many examples of domain specific patterns.

Documenting and Describing Patterns

Assume you discovered a new design pattern. Or your friend wants to explain to you this cool pattern she found ir standard format for documenting design patterns. Rather, a variety of different formats have been used by diffe pattern forms have become more well-known than others, and consequently become common starting points f documentation format is the one used by the book *Design Patterns.*[3] It contains the following sections:

- **Pattern Name and Classification:** A descriptive and unique name that helps in identifying and referring
- **Intent:** A description of the goal behind the pattern and the reason for using it.
- **Also Known As:** Other names for the pattern.
- **Motivation (Forces):** A scenario consisting of a problem and a context in which this pattern can be used.
- **Applicability:** Situations in which this pattern is usable; the context for the pattern.

- **Structure:** A graphical representation of the pattern. Class diagrams and Interaction diagrams may be used fo
- **Participants:** A listing of the classes and objects used in the pattern and their roles in the design.
- **Collaboration:** A description of how classes and objects used in the pattern interact with each other.
- **Consequences:** A description of the results, side effects, and trade offs caused by using the pattern.
- **Implementation:** A description of an implementation of the pattern; the solution part of the pattern.
- **Sample Code:** An illustration of how the pattern can be used in a programming language.
- **Known Uses:** Examples of real usages of the pattern.
- **Related Patterns:** Other patterns that have some relationship with the pattern; discussion of the differences

Of particular interest are the Structure, Participants, and Collaboration sections. These sections describe a *design* to their particular designs to solve the recurrent problem described by the design pattern. A micro-architecture relationships. Developers use the design pattern by introducing in their designs this prototypical micro-architec structure and organization similar to the chosen design motif.

References

1. Smith, Reid (October 1987). "Panel on design methodology". *OOPSLA '87 Addendum to the Proceedings*. OOPS *too much programming at, what he termed, 'the high level of wizards.' He pointed out that a written 'pattern lan abstractions. He proposed a 'radical shift in the burden of design and implementation' basing the new methodolo languages and that programming-oriented pattern languages developed at Tektronix has significantly aided their*
2. Beck, Kent; Ward Cunningham (September 1987). "Using Pattern Languages for Object-Oriented Program". *OO* Programming. OOPSLA '87. http://c2.com/doc/oopsla87.html. Retrieved 2006-05-26.
3. Gamma, Erich; Richard Helm, Ralph Johnson, and John Vlissides (1995). *Design Patterns: Elements of Reusabl*
4. Martin, Robert C.. "Design Principles and Design Patterns". http://www.objectmentor.com/resources/articles/P
5. Meyer, Bertrand; Karine Arnout (July 2006). "Componentization: The Visitor Example". *IEEE Computer* (IEEE http://se.ethz.ch/~meyer/publications/computer/visitor.pdf.
6. McConnell, Steve (June 2004). "Design in Construction". *Code Complete* (2nd ed.). Microsoft Press. pp. 104. ISE
7. Fowler, Martin (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley. ISBN 978-0321127426.
8. , unless stated otherwise. Schmidt, Douglas C.; Michael Stal, Hans Rohnert, Frank Buschmann (2000). *Pattern-* *Networked Objects*. John Wiley & Sons. ISBN 0-471-60695-2.
9. http://c2.com/cgi/wiki?BindingProperties
10. Christian Nagel, Bill Evjen, Jay Glynn, Karli Watson, and Morgan Skinner (2008). "Event-based Asynchronous
11. http://c2.com/cgi/wiki?LockPattern
12. Nock, Clifton (2003). *Data Access Patterns: Database Interactions in Object-Oriented Applications*. Addison We
13. Alur, Deepak; John Crupi, Dan Malks (May 2003). *Core J2EE Patterns: Best Practices and Design Strategies (*
14. "STL Design Patterns II,". EventHelix.com Inc.. http://www.eventhelix.com/RealtimeMantra/Patterns/stl_desi
15. "Embedded Design Patterns,". EventHelix.com Inc.. http://www.eventhelix.com/RealtimeMantra/Patterns/. Re
16. Douglass, Bruce Powel (2002). *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*
17. Douglass, Bruce Powel (1999). *Doing Hard Time: Developing Real-Time Systems with UML, Objects,* Framewor
18. Laakso, Sari A. (2003-09-16). "Collection of User Interface Design Patterns". University of Helsinki, Dept. of Cor http://www.cs.helsinki.fi/u/salaakso/patterns/index.html. Retrieved 2008-01-31.
19. Heer, J.; M. Agrawala (2006). "Software Design Patterns for Information Visualization". *IEEE Transactions on* doi:10.1109/TVCG.2006.178. PMID 17080809. http://vis.berkeley.edu/papers/infovis_design_patterns/.
20. Chad Dougherty et al (2009). *Secure Design Patterns*. http://www.cert.org/archive/pdf/09tr010.pdf.
21. Simson L. Garfinkel (2005). *Design Principles and Patterns for Computer Systems That Are Simultaneously Sec*
22. "Yahoo! Design Pattern Library". http://developer.yahoo.com/ypatterns/. Retrieved 2008-01-31.
23. "How to design your Business Model as a Lean Startup?". http://torgronsund.wordpress.com/2010/01/06/lean-s
24. Pattern Languages of Programming, Conference proceedings (annual, 1994—) [3] (http://hillside.net/plop/pastc
25. Gabriel, Dick. "A Pattern Definition". Archived from the original on 2007-02-09. http://web.archive.org/web/20(2007-03-06.
26. Fowler, Martin (2006-08-01). "Writing Software Patterns". http://www.martinfowler.com/articles/writingPatter

Further Reading

Books

- Alexander, Christopher; Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, Shlomo Angel York: Oxford University Press. ISBN 978-0195019193.
- Gamma, Erich; Richard Helm, Ralph Johnson, and John Vlissides (1995). *Design Patterns: Elements of Reusabl*
- Buschmann, Frank; Regine Meunier, Hans Rohnert, Peter Sommerlad (1996). *Pattern-Oriented Software Archite* 471-95869-7.
- Schmidt, Douglas C.; Michael Stal, Hans Rohnert, Frank Buschmann (2000). *Pattern-Oriented Software Archite* Wiley & Sons. ISBN 0-471-60695-2.
- Fowler, Martin (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley. ISBN 978-0321127426.
- Hohpe, Gregor; Bobby Woolf (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messa*
- Freeman, Eric T; Elisabeth Robson, Bert Bates, Kathy Sierra (2004). *Head First Design Patterns*. O'Reilly Med
- Alur, Deepak; John Crupi, Dan Malks (May 2003). *Core J2EE Patterns: Best Practices and Design Strategies (*
- Beck, Kent (October 2007). *Implementation Patterns*. Addison-Wesley. ISBN 978-0321413093.
- Beck, Kent; R. Crocker, G. Meszaros, J.O. Coplien, L. Dominick, F. Paulisch, and J. Vlissides (March 1996). *Pr Engineering*. pp. 25–30.
- Nock, Clifton (2003). *Data Access Patterns: Database Interactions in Object-Oriented Applications*. Addison We
- Borchers, Jan (2001). *A Pattern Approach to Interaction Design*. John Wiley & Sons. ISBN 0-471-49828-9.
- Coplien, James O.; Douglas C. Schmidt (1995). *Pattern Languages of Program Design*. Addison-Wesley. ISBN 0
- Coplien, James O.; John M. Vlissides, and Norman L. Kerth (1996). *Pattern Languages of Program Design 2*. A
- Fowler, Martin (1997). *Analysis Patterns: Reusable Object Models*. Addison-Wesley. ISBN 0-201-89542-0.
- Fowler, Martin (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley. ISBN 978-0321127426.
- Freeman, Eric; Elisabeth Freeman, Kathy Sierra, and Bert Bates (2004). *Head First Design Patterns*. O'Reilly M
- Hohmann, Luke; Martin Fowler and Guy Kawasaki (2003). *Beyond Software Architecture*. Addison-Wesley. ISB
- Alur, Deepak; Elisabeth Freeman, Kathy Sierra, and Bert Bates (2004). *Head First Design Patterns*. O'Reilly M
- Gabriel, Richard (1996) (PDF). *Patterns of Software: Tales From The Software Community*. Oxford University http://www.dreamsongs.com/NewFiles/PatternsOfSoftware.pdf.
- Gamma, Erich; Richard Helm, Ralph Johnson, and John Vlissides (1995). *Design Patterns: Elements of Reusabl*
- Hohpe, Gregor; Bobby Woolf (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messa*
- Holub, Allen (2004). *Holub on Patterns*. Apress. ISBN 1-59059-388-X.
- Kircher, Michael; Markus Völter and Uwe Zdun (2005). *Remoting Patterns: Foundations of Enterprise, Internet* ISBN 0-470-85662-9.
- Larman, Craig (2005). *Applying UML and Patterns*. Prentice Hall. ISBN 0-13-148906-2.
- Liskov, Barbara; John Guttag (2000). *Program Development in Java: Abstraction, Specification, and Object-Ori*
- Manolescu, Dragos; Markus Voelter and James Noble (2006). *Pattern Languages of Program Design 5*. Addison-
- Marinescu, Floyd (2002). *EJB Design Patterns: Advanced Patterns, Processes and Idioms*. John Wiley & Sons.
- Martin, Robert Cecil; Dirk Riehle and Frank Buschmann (1997). *Pattern Languages of Program Design 3*. Addi
- Mattson, Timothy G; Beverly A. Sanders and Berna L. Massingill (2005). *Patterns for Parallel Programming*. A
- Shalloway, Alan; James R. Trott (2001). *Design Patterns Explained, Second Edition: A New Perspective on Obj*
- Vlissides, John M. (1998). *Pattern Hatching: Design Patterns Applied*. Addison-Wesley. ISBN 0-201-43293-5.
- Weir, Charles; James Noble (2000). *Small Memory Software: Patterns for systems with limited memory*. Addison

Web sites

- "History of Patterns". *Portland Pattern Repository*. http://www.c2.com/cgi-bin/wiki?HistoryOfPatterns. Retriev
- "Are Design Patterns Missing Language Features?". Cunningham & Cunningham, Inc.. http://www.c2.com/cgi/ 20.
- "Show Trial of the Gang of Four". Cunningham & Cunningham, Inc.. http://www.c2.com/cgi/wiki?ShowTrialOf

- "Design Patterns in Modern Day Software Factories (WCSF)". XO Software, Ltd. http://www.xosoftware.co.uk/
- "STL Design Patterns II,". EventHelix.com Inc.. http://www.eventhelix.com/RealtimeMantra/Patterns/stl_desi
- "Embedded Design Patterns,". EventHelix.com Inc.. http://www.eventhelix.com/RealtimeMantra/Patterns/. Re
- "Enterprise Integration Patterns". Gregor Hohpe and Bobby Woolf, Addison-Wesley. http://www.enterpriseinteg

External Links

- Directory of websites that provide pattern catalogs (http://hillside.net/patterns/onlinepatterncatalog.htm) at hi
- Ward Cunningham's *Portland Pattern Repository*.
- Messaging Design Pattern (http://jt.dev.java.net/files/documents/5553/150311/designPatterns.pdf) Published i
- Patterns and Anti-Patterns (http://www.dmoz.org/Computers/Programming/Methodologies/Patterns_and_An
- PerfectJPattern Open Source Project (http://perfectjpattern.sourceforge.net) Design Patterns library that aims in Java.
- Lean Startup Business Model Pattern (http://torgronsund.wordpress.com/2010/01/06/lean-startup-business-mo models
- Jt (http://jt.dev.java.net) J2EE Pattern Oriented Framework
- Printable Design Patterns Quick Reference Cards (http://www.mcdonaldland.info/2007/11/28/40/)
- 101 Design Patterns & Tips for Developers (http://sourcemaking.com/design-patterns-and-tips)
- On Patterns and Pattern Languages (http://media.wiley.com/product_data/excerpt/28/04700590/0470059028.p
- Patterns for Scripted Applications (http://www.doc.ic.ac.uk/~np2/patterns/scripting/)
- Design Patterns Reference (http://www.oodesign.com/) at oodesign.com
- Design Patterns for 70% programmers in the world (http://www.slideshare.net/saurabh.net/design-patterns-for-

Anti-Patterns

Anti-Patterns and Code Smells

If design patterns are the good guys, then the anti-patterns are the bad guys. And sometimes a good guy can turn in software engineering.

The "golden hammer" is a favorite notion of this problem: you learned to use a tool in one context (the golden ham complicated tool, all of a sudden you see golden nails everywhere.

A good example is the Singleton pattern: it is so easy that it is the first pattern most beginning software engineers use it at every possible occasion. However, the problem with the Singleton is that it violates *information hiding* engineering, and it should be violated only when there is a really good reason for it. And just having learned about t

In software engineering, an *anti-pattern* is a pattern that may be commonly used but is ineffective and/or counte Koenig,[3] inspired by Gang of Four's book *Design Patterns*, which developed the concept of design patterns in the book *AntiPatterns*,[4] which extended the use of the term beyond the field of software design and into general soci least two key elements present to formally distinguish an actual anti-pattern from a simple bad habit, bad practice,

- Some repeated pattern of action, process or structure that initially appears to be beneficial, but ultimately produ
- A refactored solution exists that is clearly documented, proven in actual practice and repeatable.

By formally describing repeated mistakes, one can recognize the forces that lead to their repetition and learn how ot

Examples of Anti-Patterns

To understand anti-patterns a little better, let us take a look at a few examples. By studying them you may recogn committed yourself at one point in time. Some of these anti-patterns have very funny names.

Singleton Overuse

We have talked about this one: the first pattern you understood immediately, and you used it heavily. But beware don't use it. My experience is that the larger the project, the more Singletons show up.

How do you detect Singletons? This is very easy: look at the class diagram. All classes that have references to ther rid of them, Kerievsky shows you the medicine that cures this disease. [5]

Functional Decomposition

Although very popular once, in a modern object-oriented language there is no more space for functional decomp Usually it indicates old software that was integrated into a new project or migrated.

This anti-pattern reveals itself in three ways: The names of classes sound like function names (e.g. CalculateInteres

This anti-pattern reveals itself in three ways: The names of classes sound like function names (e.g. CalculateInteres
all class attributes are private (which is fine) but they are only used within the class. To detect this anti-pattern y
also lists the functions.

Poltergeist

People like this anti-pattern because of its name. What it is, are classes that briefly appear to only disappear into
limited functionality. Usually they are not needed or can be absorbed in other classes.

Usually one recognizes this anti-pattern by class names that end in '*controller' or '*manager'. Again a tool such as

Often a consequence of "agile" approaches where cogitating is preferred to Design.

Spaghetti

Spaghetti code is like the noodles: it is very long. Although the noodles are delicious, code the longer it gets is not.

SourceMonitor can help you find this pattern, you simply look for methods with many lines of code. Refactoring usu

Blob

A blob is a class with a lot of attributes and methods. Quite often these are not even related. You can detect this s
attributes and methods or many lines of code. Usually splitting this class into several smaller classes will help here.

Copy and Paste

As the name implies, somebody copied some code from some place to another place. It is the simplest way to d
simplest solution is to turn the code into a method instead, or use inheritance.

To detect almost identical code you can use a tool like PMD's Tool Copy/Paste Detector.[7][8]

Lava Flow

What is lava flow? "A lava flow is a moving outpouring of lava, which is created during a non-explosive effusive
rock."[9] In software engineering it means that the code is ancient, nobody has touched it for eons, and nobody has t

You can find these classes by using your source control system. Simply list those classes that have not been checked

Code Smells

Code smells are similar to anti-patterns, but not quite as formal. If code smells, then that smell can be o.k. (like so
Beck introduced the idea in the late 1990s and Martin Fowler made it popular in his book *Refactoring. Improving*
Checkstyle or PMD to find bad smells. Usually refactoring is used to remove the offending odor. Martin Fowler and

Duplicate Code

This smell is very similar to the Copy and Paste anti-pattern. You can use the PMD Tool Copy/Paste Detector [7]

Long Method

Related to the Spaghetti anti-pattern, you can find it using SourceMonitor when sorting classes according to 'A
suspicious.

Indecent Exposure

In the current Victorian age of information hiding, naturally indecent exposure is a bad thing. If a class has too n
indecent exposure. You find this smell by checking for public methods of classes. If a class has more than 50% publi

Lazy Class

Reminds me of the Poltergeist anti-pattern: this is a class that does so little that it has no reason for existence. Try
Sort 'Methods/Class' and look for classes that have fewer than two methods or look for classes with very few lines o

Large Class

A large class is the opposite of a lazy class. You find it similarily, look for classes with too many methods, or too n
or more than 400 statements. Also class with too many attributes could be large classes. Kerievsky shows several po

Really means not coding to code conventions. Look up Meyer, MISRA etc.

Known Anti-Patterns

There are many known anti-patterns. A list and brief description of some is provided for your entertainment.

Organizational anti-patterns

- Analysis paralysis: Devoting disproportionate effort to the analysis phase of a project
- Cash cow: A profitable legacy product that often leads to complacency about new products
- Design by committee: The result of having many contributors to a design, but no unifying vision
- Escalation of commitment: Failing to revoke a decision when it proves wrong
- Management by perkele: Authoritarian style of management with no tolerance of dissent
- Matrix Management: Unfocused organizational structure that results in divided loyalties and lack of direction

- Moral hazard: Insulating a decision-maker from the consequences of his or her decision
- Mushroom management: Keeping employees uninformed and misinformed (kept in the dark and fed manure)
- Stovepipe or Silos: A structure that supports mostly up-down flow of data but inhibits cross organizational comm
- Vendor lock-in: Making a system excessively dependent on an externally supplied component[11]

Project management anti-patterns

- Death march: Everyone knows that the project is going to be a disaster – except the CEO. However, the truth r Zero finally comes ("Big Bang"). Alternative definition: Employees are pressured to work late nights and weeken
- Groupthink: During groupthink, members of the group avoid promoting viewpoints outside the comfort zone of
- Smoke and mirrors: Demonstrating how unimplemented functions will appear
- Software bloat: Allowing successive versions of a system to demand ever more resources
- Waterfall model: An older method of software development that inadequately deals with unanticipated change

Analysis anti-patterns

- Bystander apathy: When a requirement or design decision is wrong, but the people who notice this do nothing b

Software design anti-patterns

- Abstraction inversion: Not exposing implemented functionality required by users, so that they re-implement it u
- Ambiguous viewpoint: Presenting a model (usually Object-oriented analysis and design (OOAD)) without specif
- Big ball of mud: A system with no recognizable structure
- Database-as-IPC: Using a database as the message queue for routine interprocess communication where a much
- Gold plating: Continuing to work on a task or project well past the point at which extra effort is adding value
- Inner-platform effect: A system so customizable as to become a poor replica of the software development platfor
- Input kludge: Failing to specify and implement the handling of possibly invalid input
- Interface bloat: Making an interface so powerful that it is extremely difficult to implement
- Magic pushbutton: Coding implementation logic directly within interface code, without using abstraction
- Race hazard: Failing to see the consequence of different orders of events
- Stovepipe system: A barely maintainable assemblage of ill-related components

Object-oriented design anti-patterns

- Anemic Domain Model: The use of domain model without any business logic. The domain model's objects canno and mutation logic is placed somewhere outside (most likely in multiple places).
- BaseBean: Inheriting functionality from a utility class rather than delegating to it
- Call super: Requiring subclasses to call a superclass's overridden method
- Circle-ellipse problem: Subtyping variable-types on the basis of value-subtypes
- Circular dependency: Introducing unnecessary direct or indirect mutual dependencies between objects or softwar
- Constant interface: Using interfaces to define constants
- God object: Concentrating too many functions in a single part of the design (class)
- Object cesspool: Reusing objects whose state does not conform to the (possibly implicit) contract for re-use
- Object orgy: Failing to properly encapsulate objects permitting unrestricted access to their internals
- Poltergeists: Objects whose sole purpose is to pass information to another object
- Sequential coupling: A class that requires its methods to be called in a particular order
- Yo-yo problem: A structure (e.g., of inheritance) that is hard to understand due to excessive fragmentation

Programming anti-patterns

- Accidental complexity: Introducing unnecessary complexity into a solution
- Action at a distance: Unexpected interaction between widely separated parts of a system
- Blind faith: Lack of checking of (a) the correctness of a bug fix or (b) the result of a subroutine
- Boat anchor: Retaining a part of a system that no longer has any use

- Busy spin: Consuming CPU while waiting for something to happen, usually by repeated checking instead of mes
- Caching failure: Forgetting to reset an error flag when an error has been corrected
- Cargo cult programming: Using patterns and methods without understanding why
- Coding by exception: Adding new code to handle each special case as it is recognized
- Error hiding: Catching an error message before it can be shown to the user and either showing nothing or showi
- Hard code: Embedding assumptions about the environment of a system in its implementation
- Lava flow: Retaining undesirable (redundant or low-quality) code because removing it is too expensive or has un
- Loop-switch sequence: Encoding a set of sequential steps using a switch within a loop statement
- Magic numbers: Including unexplained numbers in algorithms
- Magic strings: Including literal strings in code, for comparisons, as event types etc.
- Soft code: Storing business logic in configuration files rather than source code[14]
- Spaghetti code: Programs whose structure is barely comprehensible, especially because of misuse of code structu

Methodological anti-patterns

- Copy and paste programming: Copying (and modifying) existing code rather than creating generic solutions
- Golden hammer: Assuming that a favorite solution is universally applicable (See: Silver Bullet)
- Improbability factor: Assuming that it is improbable that a known error will occur
- Not Invented Here (NIH) syndrome: The tendency towards *reinventing the wheel* (Failing to adopt an existing, a
- Premature optimization: Coding early-on for perceived efficiency, sacrificing good design, maintainability, and so
- Programming by permutation (or "programming by accident"): Trying to approach a solution by successively mo
- Reinventing the wheel: Failing to adopt an existing, adequate solution
- Reinventing the square wheel: Failing to adopt an existing solution and instead adopting a custom solution whic
- Silver bullet: Assuming that a favorite technical solution can solve a larger process or problem
- Tester Driven Development: Software projects in which new requirements are specified in bug reports

Configuration management anti-patterns

- Dependency hell: Problems with versions of required products
- DLL hell: Inadequate management of dynamic-link libraries (DLLs), specifically on Microsoft Windows
- Extension conflict: Problems with different extensions to pre-Mac OS X versions of the Mac OS attempting to p
- JAR hell: Overutilization of the multiple JAR files, usually causing versioning and location problems because of

References

1. Budgen, D. (2003). *Software design*. Harlow, Eng.: Addison-Wesley. p. 225. ISBN 0-201-72219-4. http://books.g pattern%22+date:1990-2003. "As described in Long (2001), design anti-patterns are 'obvious, but wrong, solutio
2. Scott W. Ambler (1998). *Process patterns: building large-scale systems using object technology*. Cambridge, UK: http://books.google.com/?id=qJJk2yEeoZoC&pg=PA4&dq=%22anti-pattern%22+date:1990-2001. "...common a These approaches are called antipatterns."
3. Koenig, Andrew (March/April 1995). "Patterns and Antipatterns". *Journal of Object-Oriented Programming* **8,** *patterns handbook: techniques, strategies, and applications*. Cambridge, U.K.: Cambridge University Press. p. 38 id=HBAuixGMYWEC&pg=PT1&dq=0-521-64818-1. "Anti-pattern is just like pattern, except that instead of so isn't one."
4. Brown, William J.; Raphael C. Malveau, Hays W. "Skip" McCormick, Thomas J. Mowbray (1998). *AntiPatterns* Wiley & Sons, ltd. ISBN 0471197130.
5. Kerievsky, Joshua (2004). *Refactoring to Patterns*. Addison-Wesley Professional. ISBN 0321213351.
6. http://www.campwoodsw.com/sourcemonitor.html SourceMonitor
7. http://pmd.sourceforge.net/cpd.html PMD
8. http://www.onjava.com/pub/a/onjava/2003/03/12/pmd_cpd.html Detecting Duplicate Code with PMD's CPD
9. http://en.wikipedia.org/wiki/Lava Lava
10. Fowler, Martin (1999). *Refactoring. Improving the Design of Existing Code*. Addison-Wesley. ISBN 0-201-48567-

11. Vendor Lock-In (http://www.antipatterns.com/vendorlockin.htm) at antipatterns.com
12. Lava Flow (http://www.antipatterns.com/lavaflow.htm) at antipatterns.com
13. "Undocumented 'lava flow' antipatterns complicate process". Icmgworld.com. 2002-01-14. http://www.icmgworld
14. Papadimoulis, Alex (2007-04-10). "Soft Coding". Worsethanfailure.com. http://worsethanfailure.com/Articles/So

Further Reading

Books

- Laplante, Phillip A.; Colin J. Neill (2005). *Antipatterns: Identification, Refactoring and Management*. Auerbach
- Brown, William J.; Raphael C. Malveau, Hays W. "Skip" McCormick, Scott W. Thomas, Theresa Hudson (ed). (ISBN 0-471-36366-9.
- Brown, William J.; Raphael C. Malveau, Hays W. "Skip" McCormick, Thomas J. Mowbray (1998). *AntiPatterns* Wiley & Sons, ltd. ISBN 0471197130.
- Kerievsky, Joshua (2004). *Refactoring to Patterns*. Addison-Wesley Professional. ISBN 0321213351.
- Feathers, Michael (2004). *Working Effectively with Legacy Code*. Prentice Hall. ISBN 0131177052.

Web sites

- "The Bad Code Spotter's Guide". Diomidis Spinellis. http://www.informit.com/articles/article.aspx?p=457502. I

External Links

- Anti-pattern (http://c2.com/cgi/wiki?AntiPattern) at WikiWikiWeb
- Anti-patterns catalog (http://c2.com/cgi/wiki?AntiPatternsCatalog)
- AntiPatterns.com (http://www.antipatterns.com) Web site for the AntiPatterns book
- Patterns of Toxic Behavior (http://www.personal.psu.edu/cjn6/Personal/Antipatterns-%20Patterns%20of%20T
- CodeSmell at c2.com (http://c2.com/cgi/wiki?CodeSmell)
- Taxonomy of code smells (http://blog.iandavis.com/2004/11/taxonomy-of-code-smells/)

Implementation

Introduction

Computer programming (often shortened to **programming** or **coding**) is the process of designing, writing of computer programs. This source code is written in a programming language. The purpose of programming is to cr writing source code often requires expertise in many different subjects, including knowledge of the application domai

Definition

Hoc and Nguyen-Xuan define computer programming as "the process of transforming a mental plan in familiar programming is the craft of transforming requirements into something that a computer can execute.

Overview

Within software engineering, programming (the *implementation*) is regarded as one phase in a software development

There is an ongoing debate on the extent to which the writing of programs is an art, a craft or an engineering disci application of all three, with the goal of producing an efficient and evolvable software solution (the criteria for "e many other technical professions in that programmers, in general, do not need to be licensed or pass any standa themselves "programmers" or even "software engineers." However, representing oneself as a "Professional Software many parts of the world. However, because the discipline covers many areas, which may or may not include cri profession as a whole. In most cases, the discipline is self-governed by the entities which require the programming, Air Force use of AdaCore and security clearance).

Another ongoing debate is the extent to which the programming language used in writing computer programs affe that surrounding the Sapir–Whorf hypothesis [3] in linguistics, which postulates that a particular spoken langu language patterns yield different patterns of thought. This idea challenges the possibility of representing the world p any language condition the thoughts of its speaker community.

History

The Antikythera mechanism from ancient Greece was a calculator utilizing gears of various sizes and configuration t in lunar-to-solar calendars, and which is consistent for calculating the dates of the Olympiads.[5] Al-Jazari built pro was the use of pegs and cams placed into a wooden drum at specific locations. which would sequentially trigger leve

device was a small drummer playing various rhythms and drum patterns.[6][7] The Jacquard Loom, which Joseph M holes punched in them. The hole pattern represented the pattern that the loom had to follow in weaving cloth. The cards. Charles Babbage adopted the use of punched cards around 1830 to control his Analytical Engine. The synthes operation and output, along with a way to organize and input instructions in a manner relatively easy for humans t development of computer programming. Development of computer programming accelerated through the Industrial In the late 1880s, Herman Hollerith invented the recording of data on a medium that could then be read by a m above, had been for control, not data. "After some initial trials with paper tape, he settled on punched cards..."[8] T "Hollerith cards" he invented the tabulator, and the keypunch machines. These three inventions were the foun industry. In 1896 he founded the *Tabulating Machine Company* (which later became the core of IBM). The additi Type I Tabulator allowed it to do different jobs without having to be physically rebuilt. By the late 1940s, ther machines, called unit record equipment, to perform data-processing tasks (card reading). Early computer prog complex calculations requested of the newly invented machines.

Data and instructions could be stored on external punched cards, which were kept in order and arranged in program decks.

The invention of the von Neumann architecture allowed computer programs to be stored be painstakingly crafted using the instructions (elementary operations) of the particular m of computer would likely use different instructions (machine language) to do the same ta that let the programmer specify each instruction in a text format, entering abbreviations and specifying addresses in symbolic form (e.g., ADD X, TOTAL). Entering a program in faster, and less prone to human error than using machine language, but because an ass language, any two machines with different instruction sets also have different assembly lan

In 1954, FORTRAN was invented; it was the first high level programming language to have a functional impl language is, in very general terms, any programming language that allows the programmer to write programs in ter level of abstraction "higher" than that of an assembly language.) It allowed programmers to specify calculations b text, or *source*, is converted into machine instructions using a special program called a compiler, which translat FORTRAN stands for "Formula Translation". Many other languages were developed, including some for commerc using punched cards or paper tape. (See computer programming in the punch card era). By the late 1960s, data s programs could be created by typing directly into the computers. Text editors were developed that allowed changes (Usually, an error in punching a card meant that the card had to be discarded and a new one punched to replace it.

As time has progressed, computers have made giant leaps in the area of processing power. This has brought ab underlying hardware. Although these high-level languages usually incur greater overhead, the increase in speed c practical than in the past. These increasingly abstracted languages typically are easier to learn and allow the pr source code. However, high-level languages are still impractical for a few programs, such as those where low-level ha

Throughout the second half of the twentieth century, programming was an attractive career in most developed c offshore outsourcing (importing software and services from other countries, usually at a lower wage), making progr increasing economic opportunities in less developed areas. It is unclear how far this trend will continue and how dee

Modern programming

Quality requirements

Whatever the approach to software development may be, the final program must satisfy some fundamental properti

- **Efficiency/performance**: the amount of system resources a program consumes (processor time, memory spa even user interaction): the less, the better. This also includes correct disposal of some resources, such as cleaning
- **Reliability**: how often the results of a program are correct. This depends on conceptual correctness of algorith resource management (e.g., buffer overflows and race conditions) and logic errors (such as division by zero or off
- **Robustness**: how well a program anticipates problems not due to programmer error. This includes situations s resources such as memory, operating system services and network connections, and user error.
- **Usability**: the ergonomics of a program: the ease with which a person can use the program for its intended pur make or break its success even regardless of other issues. This involves a wide range of textual, graphical and so cohesiveness and completeness of a program's user interface.
- **Portability**: the range of computer hardware and operating system platforms on which the source code of a pr differences in the programming facilities provided by the different platforms, including hardware and operating s system, and availability of platform specific compilers (and sometimes libraries) for the language of the source c
- **Maintainability**: the ease with which a program can be modified by its present or future developers in order adapt it to new environments. Good practices during initial development make the difference in this regard. This significantly affect the fate of a program over the long term.

Algorithmic complexity

The academic field and the engineering practice of computer programming are both largely concerned with discove problem. For this purpose, algorithms are classified into *orders* using so-called Big O notation, *O(n)*, which expresse of the size of an input. Expert programmers are familiar with a variety of well-established algorithms and their resp best suited to the circumstances.

Methodologies

The first step in most formal software development projects is requirements analysis, followed by testing to determi There exist a lot of differing approaches for each of those tasks. One approach popular for requirements analysis software development where the various stages of formal software development are more integrated together into approaches to the Software development process

Popular modeling techniques include Object-Oriented Analysis and Design (OOAD) and Model-Driven Architecture both the OOAD and MDA.

A similar technique used for database design is Entity-Relationship Modeling (ER Modeling).

Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and l

Measuring language usage

It is very difficult to determine what are the most popular of modern programming languages. Some languages ar strong in the corporate data center, often on large mainframes, FORTRAN in engineering applications, scripting l some languages are regularly used to write many different kinds of applications.

Methods of measuring programming language popularity include: counting the number of job advertisements that that are sold (this overestimates the importance of newer languages), and estimates of the number of existing lines c of business languages such as COBOL).

Debugging

Debugging is a very important task in the software development process, because an incorrect program can have s languages are more prone to some kinds of faults because their specification does not require compilers to perform a a static analysis tool can help detect some possible problems.

Debugging is often done with IDEs like Eclipse, Kdevelop, NetBeans, and Visual Studio. Standalone debuggers like less of a visual environment, usually using a command line.

Programming languages

Different programming languages support different styles of programming (called *programming paradigms*). The c considerations, such as company policy, suitability to task, availability of third-party packages, or individual pref best suited for the task at hand will be selected. Trade-offs from this ideal involve finding enough programmers w availability of compilers for that language, and the efficiency with which programs written in a given language spectrum from "low-level" to "high-level"; "low-level" languages are typically more machine-oriented and faster to more abstract and easier to use but execute less quickly.

Allen Downey, in his book *How To Think Like A Computer Scientist*, writes:

> The details look different in different languages, but a few basic instructions appear in just about every language

- **input**: Get data from the keyboard, a file, or some other device.
- **output**: Display data on the screen or send data to a file or other device.
- **arithmetic**: Perform basic arithmetical operations like addition and multiplication.
- **conditional execution**: Check for certain conditions and execute the appropriate sequence of statements.
- **repetition**: Perform some action repeatedly, usually with some variation.

Many computer languages provide a mechanism to call functions provided by libraries. Provided the functions in passing arguments), then these functions may be written in any other language.

Programmers

Computer programmers are those who write computer software. Their jobs usually involve:

- Coding
- Compilation
- Debugging
- Documentation
- Integration
- Maintenance
- Requirements analysis
- Software architecture
- Software testing
- Specification

References

1. Hoc, J.-M. and Nguyen-Xuan, A. Language semantics, mental models and analogy. J.-M. Hoc et al., Eds. Psych through Brad A. Myers , John F. Pane , Andy Ko, Natural programming languages and environments, Commun 45/1015864.1015888)
2. Paul Graham (2003). *Hackers and Painters*. http://www.paulgraham.com/hp.html. Retrieved 2006-08-22.
3. Kenneth E. Iverson, the originator of the APL programming language, believed that the Sapir–Whorf hypothesis hypothesis by name). His Turing award lecture, "Notation as a tool of thought", was devoted to this theme, argu algorithms. Iverson K.E.,"Notation as a tool of thought (http://elliscave.com/APL_J/tool.pdf)", *Communicatio*
4. "Ancient Greek Computer's Inner Workings Deciphered (http://news.nationalgeographic.com/news/2006/11/061 2006.
5. Freeth, Tony; Jones, Alexander; Steele, John M.; Bitsakis, Yanis (July 31, 2008). "Calendars with Olympiad disp (7204): 614–617. doi:10.1038/nature07130. PMID 18668103. http://www.nature.com/nature/journal/v454/n7204
6. A 13th Century Programmable Robot (http://www.shef.ac.uk/marcoms/eview/articles58/robot.html), Universit
7. Fowler, Charles B. (October 1967). "The Museum of Music: A History of Mechanical Instruments". *Music Educa* doi:10.2307/3391092. http://jstor.org/stable/3391092
8. "Columbia University Computing History - Herman Hollerith". Columbia.edu. http://www.columbia.edu/acis/hi

9. 12:10 p.m. ET (2007-03-20). "Fortran creator John Backus dies - Tech and gadgets- msnbc.com". MSNBC. http:
10. "CSC-302 99S : Class 02: A Brief History of Programming Languages". Math.grin.edu. http://www.math.grin.ed 2010-04-25.
11. Survey of Job advertisements mentioning a given language (http://www.computerweekly.com/Articles/2007/09/ ob.htm)>

Further reading

- Weinberg, Gerald M., *The Psychology of Computer Programming*, New York: Van Nostrand Reinhold, 1971

External links

- How to Think Like a Computer Scientist (http://openbookproject.net/thinkCSpy) - by Jeffrey Elkner, Allen B.

Code Convention

Coding conventions are a set of guidelines for a specific programming language that recommend programming in this language. These conventions usually cover file organization, indentation, comments, declarations, st programming principles, programming rules of thumb, etc. Software programmers are highly recommended to follc and make software maintenance easier. Coding conventions are only applicable to the human maintainers and pe documented set of rules that an entire team or company follows, or may be as informal as the habitual coding pract As a result, not following some or all of the rules has no impact on the executable programs created from the source

Software maintenance

Reducing the cost of software maintenance is the most often cited reason for following coding conventions. In their Sun Microsystems provides the following rationale:[1]

> Code conventions are important to programmers for a number of reasons:
>
> - 80% of the lifetime cost of a piece of software goes to maintenance.
> - Hardly any software is maintained for its whole life by the original author.
> - Code conventions improve the readability of the software, allowing engineers to understand new code more
> - If you ship your source code as a product, you need to make sure it is as well packaged and clean as any o

Quality

Software peer review frequently involves reading source code. This type of peer review is primarily a defect detectic read the source file before the code is submitted for review. Code that is written using consistent guidelines is easier of the defect detection process.

Even for the original author, consistently coded software eases maintainability. There is no guarantee that an indi code was written in a certain way long after the code was originally written. Coding conventions can help. Consiste to understand the software.

Refactoring

Refactoring refers to a software maintenance activity where source code is modified to improve readability or impro with a team's stated coding standards after its initial release. Any change that does not alter the behavior of the sc changing variable names, renaming methods, moving methods or whole classes and breaking large methods (or funct

Agile software development methodologies plan for regular (or even continuous) refactoring making it an integral pa

Task automation

Coding conventions allow to have simple scripts or programs whose job is to process source code for some purpos count the software size (Source lines of code) to track current project progress or establish a baseline for future proje

Consistent coding standards can, in turn, make the measurements more consistent. Special tags within source examples are javadoc and doxygen. The tools specifies the use of a set of tags, but their use within a project is deter

Coding conventions simplify writing new software whose job is to process existing software. Use of static code anal class of development tools stems from increased maturity and sophistication of the practitioners themselves (and th languages themselves.

Language factors

All software practitioners must grapple with the problems of organizing and managing very many detailed instruct task for which it was written. For all but the smallest software projects, source code (instructions) are partitioned i for programmers to collect closely related functions (behaviors) in the same file and to collect related files into programming (such as found in FORTRAN) towards more object-oriented constructs (such as found in C++), it be file (the 'one class per file' convention).[3][4] Java has gone one step further - the Java compiler returns an error if it

A convention in one language may be a requirement in another. Language conventions also affect individual sou unique. The rules a compiler applies to the source creates implicit standards. For example, Python code is (indentation) is actually significant to the interpreter. Python does not use the brace syntax Perl uses to delimit fu

uses a brace syntax similar to Perl or C/C++ to delimit functions, does not allow the following, which seems fairly

```
set i 0
while {$i < 10}
{
   puts "$i squared = [expr $i*$i]"
   incr i
}
```

The reason is that in Tcl, curly braces are not used only to delimit functions as in C or Java. More generally, curly Tcl, the *word* **while** takes two arguments, a *condition* and an *action*. In the example above, **while** is missing character to delimit the end of a command).

Common conventions

As mentioned above, common coding conventions may cover the following areas:

- Comment conventions
- Indent style conventions
- Naming conventions
- Programming practices
- Programming principles
- Programming rules of thumb
- Programming style conventions

Examples

Only one statement should occur per line

For example, in Java this would involve having statements written like this:

```
a++;
b = a;
```

But not like this:

```
a++; b = a;
```

Boolean values in decision structures

Some programmers suggest that coding where the result of a decision is merely the computation of a Boolean value, the computation itself, like this:

```
return (hours < 24) && (minutes < 60) && (seconds < 60);
```

The difference is entirely stylistic, because optimizing compilers may produce identical object code for both forms. I and maintain.

Arguments in favor of the longer form include: it is then possible to set a per-line breakpoint on one branch of tl refactoring the return line, which would increase the chances of bugs being introduced; the longer form would alwa still in scope.

Left-hand comparisons

In languages which use one symbol (typically a single equals sign, (=)) for assignment and another (typically two PHP, Perl numeric context, and most languages in the last 15 years), and where assignments may be made witl comparison style: to place constants or expressions to the left in any comparison. [9] [10]

Here are both left and right-hand comparison styles, applied to a line of Perl code. In both cases, this compares the in the subsequent block.

```
if ( $a == 42 ) { ... }  # A right-hand comparison checking if $a equals 42.
if ( 42 == $a ) { ... }  # Recast, using the left-hand comparison style.
```

The difference occurs when a developer accidentally types = instead of ==:

```
if ( $a = 42 ) { ... }  # Inadvertent assignment which is often hard to debug
if ( 42 = $a ) { ... }  # Compile time error indicates source of problem
```

The first (right-hand) line now contains a potentially subtle flaw: rather than the previous behaviour, it now sets t
block. As this is syntactically legitimate, the error may go unnoticed by the programmer, and the software may ship

The second (left-hand) line contains a semantic error, as numeric values cannot be assigned to. This will result in
error cannot go unnoticed by the programmer.

Some languages have built-in protections against inadvertent assignment. Java and C#, for example, do not suppor

The risk can also be mitigated by use of static code analysis tools that can detect this issue.

Looping and control structures

The use of logical control structures for looping adds to good programming style as well. It helps someone readi
imperative programming languages). For example, in pseudocode:

```
i = 0

while i < 5
  print i * 2
  i = i + 1
end while

print "Ended loop"
```

The above snippet obeys the naming and indentation style guidelines, but the following use of the "for" construct m

```
for i = 0, i < 5, i=i+1
  print i * 2

print "Ended loop"
```

In many languages, the often used "for each element in a range" pattern can be shortened to:

```
for i = 0 to 5
  print i * 2

print "Ended loop"
```

In programming languages that allow curly brackets, it has become common for style documents to require th
constructs.

```
for (i = 0 to 5) {
  print i * 2;
}
```

```
print "Ended loop";
```

This prevents program-flow bugs which can be time-consuming to track down, such as where a terminating semicolc

```
for (i = 0; i < 5; ++i);
   printf("%d\n", i*2);    /* The incorrect indentation hides the fact
                           that this line is not part of the loop body. */

printf("Ended loop");
```

...or where another line is added before the first:

```
for (i = 0; i < 5; ++i)
   fprintf(logfile, "loop reached %d\n", i);
   printf("%d\n", i*2);    /* The incorrect indentation hides the fact
                           that this line is not part of the loop body. */

printf("Ended loop");
```

Lists

Where items in a list are placed on separate lines, it is sometimes considered good practice to add the item-separa languages where doing so is supported by the syntax (e.g., C, Java)

```
const char *array[] = {
   "item1",
   "item2",
   "item3",  /* still has the comma after it */
};
```

This prevents syntax errors or subtle string-concatenation bugs when the list items are re-ordered or more items separator on the line which was previously last in the list. However, this technique can result in a syntax error (o support trailing commas, not all list-like syntactical constructs in those languages may support it.

References

1. "Code Conventions for the Java Programming Language : Why Have Code Conventions". Sun Microsystems, Inc http://java.sun.com/docs/codeconv/html/CodeConventions.doc.html#16712.
2. Jeffries, Ron (2001-11-08). "What is Extreme Programming? : Design Improvement". XP Magazine. http://www
3. Hoff, Todd (2007-01-09). "C++ Coding Standard : Naming Class Files". http://www.possibility.com/Cpp/CppC
4. FIFE coding standards (http://wiki.fifengine.de/index.php?title=Coding_standards)
5. van Rossum, Guido; Fred L. Drake, Jr., editor (2006-09-19). "Python Tutorial : First Steps Towards Programmi http://docs.python.org/tut/node5.html#SECTION005200000000000000000.
6. Raymond, Eric (2000-05-01). "Why Python?". Linux Journal. http://www.linuxjournal.com/article/3882.
7. Tcl Developer Xchange. "Summary of Tcl language syntax". ActiveState. http://www.tcl.tk/man/tcl8.4/TclCmd
8. Staplin, George Peter (2006-07-16). "Why can I not start a new line before a brace group". 'the Tcler's Wiki'. ht
9. Sklar, David; Adam Trachtenberg (2003). *PHP Cookbook*. O'Reilly., recipe 5.1 "Avoiding == Versus = Confusic
10. "C Programming FAQs: Frequently Asked Questions". Addison-Wesley, 1995. Nov. 2010. http://c-faq.com/style

External links

Coding conventions for languages

- ActionScript: Flex SDK coding conventions and best practices (http://opensource.adobe.com/wiki/display/flexs
- Ada: Ada 95 Quality and Style Guide: Guidelines for Professional Programmers (http://www.adaic.com/docs/9
- Ada: Guide for the use of the Ada programming language in high integrity systems (http://www.dit.upm.es/ork
- Ada: NASA Flight Software Branch — Ada Coding Standard (http://software.gsfc.nasa.gov/AssetsApproved/P.
- Ada: European Space Agency's Ada Coding Standard (ftp://ftp.estec.esa.nl/pub/wm/wme/bssc/bssc983.pdf) (B
- C: Ganssle Group's Firmware Development Standard (http://www.ganssle.com/fsm.pdf)
- C: Netrino Embedded C Coding Standard (http://www.netrino.com/Coding-Standard)
- C: Micrium C Coding Standard (http://micrium.com/download/an2000.pdf)
- C++: Quantum Leaps C/C++ Coding Standard (http://www.state-machine.com/doc/AN_QL_Coding_Stand
- C++: GeoSoft's C++ Programming Style Guidelines (http://geosoft.no/development/cppstyle.html)
- C++: Google's C++ Style Guide (http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml)
- C#: Design Guidelines for Developing Class Libraries (http://msdn.microsoft.com/en-us/library/ms229042(VS.8
- C#: Microsoft (http://blogs.msdn.com/brada/articles/361363.aspx), Philips Healthcare (http://www.tiobe.com/
- D: The D Style (http://www.digitalmars.com/d/1.0/dstyle.html)
- Erlang: Erlang Programming Rules and Conventions (http://www.erlang.se/doc/programming_rules.shtml)
- Flex: Code conventions for the Flex SDK (http://opensource.adobe.com/wiki/display/flexsdk/Coding+Conventi
- GML: Game Maker Language (http://yoyogames.com/)
- Java: Ambysoft's Coding Standards for Java (http://www.ambysoft.com/essays/javaCodingStandards.html)
- Java: Code Conventions for the Java Programming Language (http://java.sun.com/docs/codeconv/)
- Java: GeoSoft's Java Programming Style Guidelines (http://geosoft.no/development/javastyle.html)
- Java: Java Coding Standards (http://www.dmoz.org//Computers/Programming/Languages/Java/Coding_Stan
- Lisp: Riastradh's Lisp Style Rules (http://mumble.net/~campbell/scheme/style.txt)
- Mono: Programming style for Mono (http://www.mono-project.com/Coding_Guidelines)
- Object Pascal: Object Pascal Style Guide (http://bdn.borland.com/article/10280)
- Perl: Perl Style Guide (http://perldoc.perl.org/perlstyle.html)
- PHP::PEAR: PHP::PEAR Coding Standards (http://pear.php.net/manual/en/standards.php)
- Python: Style Guide for Python Code (http://www.python.org/peps/pep-0008.html)
- Ruby: The Unofficial Ruby Usage Guide (http://www.caliban.org/ruby/rubyguide.shtml)
- Ruby: Good API Design (http://rpa-base.rubyforge.org/wiki/wiki.cgi?GoodAPIDesign)

Coding conventions for projects

- Apache Developers' C Language Style Guide (http://httpd.apache.org/dev/styleguide.html)
- Drupal PHP Coding Standards (http://drupal.org/coding-standards)
- Linux Kernel Coding Style (http://lxr.linux.no/source/Documentation/CodingStyle) (or Documentation/Coding
- ModuLiq Zero Indent Coding Style (http://moduliq.org/documentation/moduliq_zero_indent_coding_style.ht
- Mozilla Coding Style Guide (http://www.mozilla.org/hacking/mozilla-style-guide.html)
- Road Intranet's C++ Guidelines (http://www.qhull.org/road/road-faq/xml/cpp-guideline.xml)
- The NetBSD source code style guide (ftp://ftp.netbsd.org/pub/NetBSD/NetBSD-current/src/share/misc/style)
- "GNAT Coding Style: A Guide for GNAT Developers". *GCC online documentation*. Free Software Foundation. l (http://gcc.gnu.org/onlinedocs/gnat-style.pdf))

Good Coding

Introduction to Software Engineering/Implementation/Good Coding

Documentation

Software documentation or **source code documentation** is written text that accompanies computer soft different things to people in different roles.

Involvement of people in software life

Documentation is an important part of software engineering. Types of documentation include:

1. Requirements - Statements that identify attributes, capabilities, characteristics, or qualities of a system. This is
2. Architecture/Design - Overview of softwares. Includes relations to an environment and construction principles to
3. Technical - Documentation of code, algorithms, interfaces, and APIs.
4. End User - Manuals for the end-user, system administrators and support staff.
5. Marketing - How to market the product and analysis of the market demand.

Requirements documentation

Requirements documentation is the description of what a particular software does or shall do. It is used throughou also used as an agreement or as the foundation for agreement on what the software shall do. Requirements are prod end users, customers, product managers, project managers, sales, marketing, software architects, usability enginee requirements documentation has many different purposes.

Requirements come in a variety of styles, notations and formality. Requirements can be goal-like (e.g., *distributed clicking a configuration file and select the 'build' function*), and anything in between. They can be specified as stat formulas, and as a combination of them all.

The variation and complexity of requirements documentation makes it a proven challenge. Requirements may be im what kind of documentation is needed and how much can be left to the architecture and design documentation, ar variety of people that shall read and use the documentation. Thus, requirements documentation is often incon software changes become more difficult—and therefore more error prone (decreased software quality) and time-consu

The need for requirements documentation is typically related to the complexity of the product, the impact of the complex or developed by many people (e.g., mobile phone software), requirements can help to better communicate impact on human life (e.g., nuclear power systems, medical equipment), more formal requirements documentation two (e.g., very small mobile phone applications developed specifically for a certain campaign) very little requirement later built upon, requirements documentation is very helpful when managing the change of the software and verifyin

Traditionally, requirements are specified in requirements documents (e.g. using word processing applications an changing nature of requirements documentation (and software documentation in general), database-centric systems a

Architecture/Design documentation

Architecture documentation is a special breed of design document. In a way, architecture documents are third deriv documents being first). Very little in the architecture documents is specific to the code itself. These documents d particular routine exists in the form that it does, but instead merely lays out the general requirements that would m short on details but thick on explanation. It may suggest approaches for lower level design, but leave the actual exp

Another breed of design docs is the comparison document, or trade study. This would often take the form of a *u* alternate approaches. It could be at the user interface, code, design, or even architectural level. It will outline wha pros and cons of each. A good trade study document is heavy on research, expresses its idea clearly (without relying impartial. It should honestly and clearly explain the costs of whatever solution it offers as best. The objective particular point of view. It is perfectly acceptable to state no conclusion, or to conclude that none of the alternative be approached as a scientific endeavor, not as a marketing technique.

A very important part of the design document in enterprise software development is the Database Design Do Elements. The DDD includes the formal information that the people who interact with the database need. The p players within the scene. The potential users are:

- Database Designer
- Database Developer
- Database Administrator
- Application Designer
- Application Developer

When talking about Relational Database Systems, the document should include following parts:

- Entity - Relationship Schema, including following information and their clear definitions:
  - Entity Sets and their attributes
  - Relationships and their attributes
  - Candidate keys for each entity set
  - Attribute and Tuple based constraints
- Relational Schema, including following information:
  - Tables, Attributes, and their properties
  - Views
  - Constraints such as primary keys, foreign keys,
  - Cardinality of referential constraints
  - Cascading Policy for referential constraints

- Primary keys

It is very important to include all information that is to be used by all actors in the scene. It is also very important

Technical documentation

This is what most programmers mean when using the term *software documentation*. When creating software, code various aspects of its intended operation. It is important for the code documents to be thorough, but not so ve overview documentation are found specific to the software application or software product being documented by A also the end customers or clients using this software application. Today, we see lot of high end applications in security, industry automation and a variety of other domains. Technical documentation has become important wi may change over a period of time with architecture changes. Hence, technical documentation has gained lot of impo

Often, tools such as Doxygen, NDoc, javadoc, EiffelStudio, Sandcastle, ROBODoc, POD, TwinText, or Universal extract the comments and software contracts, where available, from the source code and create reference manuals in into a *reference guide* style, allowing a programmer to quickly look up an arbitrary function or class.

The idea of auto-generating documentation is attractive to programmers for various reasons. For example, bec comments), the programmer can write it while referring to the code, and use the same tools used to create the sou the documentation up-to-date.

Of course, a downside is that only programmers can edit this kind of documentation, and it depends on them to documents nightly). Some would characterize this as a pro rather than a con.

Donald Knuth has insisted on the fact that documentation can be a very difficult afterthought process and has ad the source code and extracted by automatic means.

Elucidative Programming is the result of practical applications of Literate Programming in real programming documentation be stored separately. This paradigm was inspired by the same experimental findings that produced create and access information that is not going to be part of the source file itself. Such annotations are usually porting, where third party source code is analysed in a functional way. Annotations can therefore help the documentation system would hinder progress. Kelp (http://kelp.sf.net/) stores annotations in separate files, linking

User documentation

Unlike code documents, user documents are usually far more diverse with respect to the source code of the program,

In the case of a software library, the code documents and user documents could be effectively equivalent and are wo

Typically, the user documentation describes each feature of the program, and assists the user in realizing these fea troubleshooting assistance. It is very important for user documents to not be confusing, and for them to be up to da is very important for them to have a thorough index. Consistency and simplicity are also very valuable. User do software will do. API Writers are very well accomplished towards writing good user documents as they would be we See also Technical Writing.

There are three broad ways in which user documentation can be organized.

1. **Tutorial:** A tutorial approach is considered the most useful for a new user, in which they are guided through e
2. **Thematic:** A thematic approach, where chapters or sections concentrate on one particular area of interest, is o convey their ideas through a knowledge based article to facilitating the user needs. This approach is usually prac user population is largely correlated with the troubleshooting demands [2], [3].
3. **List or Reference:** The final type of organizing principle is one in which commands or tasks are simply listed This latter approach is of greater use to advanced users who know exactly what sort of information they are lool

A common complaint among users regarding software documentation is that only one of these three approaches w provided software documentation for personal computers to online help that give only reference information on co experienced users get the most out of a program is left to private publishers, who are often given significant assistan

Marketing documentation

For many applications it is necessary to have some promotional materials to encourage casual observers to spend three purposes:-

1. To excite the potential user about the product and instill in them a desire for becoming more involved with it.
2. To inform them about what exactly the product does, so that their expectations are in line with what they will l
3. To explain the position of this product with respect to other alternatives.

One good marketing technique is to provide clear and memorable *catch phrases* that exemplify the point we wish anything else provided by the manufacturer.

Notes

1. Woelz, Carlos. "The KDE Documentation Primer". http://i18n.kde.org/docs/doc-primer/index.html. Retrieved 1
2. Microsoft. "Knowledge Base Articles for Driver Development". http://www.microsoft.com/whdc/driver/kernel/k
3. Prekaski, Todd. "Building web and Adobe AIR applications from a shared Flex code base". http://www.adobe.c 2009.

External links

- kelp (http://kelp.sf.net/) - a source code annotation framework for architectural, design and technical document
- ISO documentation standards committee (http://isotc.iso.org/livelink/livelink?func=ll&objId=8914719&objActi
  Standardization committee which develops user documentation standards.

Testing

Introduction

Software testing is an investigation conducted to provide stakeholders with information about the quality of objective, independent view of the software to allow the business to appreciate and understand the risks of softwa process of executing a program or application with the intent of finding software bugs.

Software testing can also be stated as the process of validating and verifying that a software program/application/p

1. meets the business and technical requirements that guided its design and development;
2. works as expected; and
3. can be implemented with the same characteristics.

Software testing, depending on the testing method employed, can be implemented at any time in the development have been defined and the coding process has been completed. As such, the methodology of the test is governed by t

Different software development models will focus the test effort at different points in the development process. development and place an increased portion of the testing in the hands of the developer, before it reaches a formal occurs after the requirements have been defined and the coding process has been completed.

Overview

Testing can never completely identify all the defects within software. Instead, it furnishes a *criticism* or *comparison* principles or mechanisms by which someone might recognize a problem. These oracles may include (but are not lin of the same product, inferences about intended or expected purpose, user or customer expectations, relevant standar

Every software product has a target audience. For example, the audience for video game software is completely dif or otherwise invests in a software product, it can assess whether the software product will be acceptable to its **Software testing** is the process of attempting to make this assessment.

A study conducted by NIST in 2002 reports that software bugs cost the U.S. economy $59.5 billion annually. More performed.[3]

History

The separation of debugging from testing was initially introduced by Glenford J. Myers in 1979.[4] Although his a bug"[4][5]) it illustrated the desire of the software engineering community to separate fundamental development act William C. Hetzel classified in 1988 the phases and goals in software testing in the following stages:[6]

- Until 1956 - Debugging oriented[7]
- 1957–1978 - Demonstration oriented[8]
- 1979–1982 - Destruction oriented[9]
- 1983–1987 - Evaluation oriented[10]
- 1988–2000 - Prevention oriented[11]

Software testing topics

Scope

A primary purpose of testing is to detect software failures so that defects may be discovered and corrected. This is properly under all conditions but can only establish that it does not function properly under specific conditions.[12] as execution of that code in various environments and conditions as well as examining the aspects of code: does it culture of software development, a testing organization may be separate from the development team. There are va testing may be used to correct the process by which software is developed.[13]

Functional vs non-functional testing

Functional testing refers to activities that verify a specific action or function of the code. These are usually found methodologies work from use cases or user stories. Functional tests tend to answer the question of "can the user do t

Non-functional testing refers to aspects of the software that may not be related to a specific function or user actio such questions as "how many people can log in at once".

Defects and failures

Not all software defects are caused by coding errors. One common source of expensive defects is caused by requ
omission by the program designer.[14] A common source of requirements gaps is non-functional requirements such
security.

Software faults occur through the following processes. A programmer makes an error (mistake), which results in a d
in certain situations the system will produce wrong results, causing a failure.[15] Not all defects will necessarily re
failures. A defect can turn into a failure when the environment is changed. Examples of these changes in envir
alterations in source data or interacting with different software.[15] A single defect may result in a wide range of fail

Finding faults early

It is commonly believed that the earlier a defect is found the cheaper it is to fix it.[16] The following table shows th
example, if a problem in the requirements is found only post-release, then it would cost 10–100 times more to fix tha

Cost to fix a defect		Time detected			
		Requirements	**Architecture**	**Construction**	**System tes**
Time introduced	**Requirements**	1×	3×	5–10×	10×
	Architecture	-	1×	10×	15×
	Construction	-	-	1×	10×

Compatibility

A common cause of software failure (real or perceived) is a lack of compatibility with other application software,
environments that differ greatly from the original (such as a terminal or GUI application intended to be run on th
render in a web browser). For example, in the case of a lack of backward compatibility, this can occur because the
target environment, which not all users may be running. This results in the unintended consequence that the latest
on older hardware that earlier versions of the target environment was capable of using. Sometimes such issues can b
separate program module or library.

Input combinations and preconditions

A very fundamental problem with software testing is that testing under *all* combinations of inputs and precondition
means that the number of defects in a software product can be very large and defects that occur infrequently are dif
quality (how it is supposed to *be* versus what it is supposed to *do*)—usability, scalability, performance, compati
sufficient value to one person may be intolerable to another.

Static vs. dynamic testing

There are many approaches to software testing. Reviews, walkthroughs, or inspections are considered as static testin
cases is referred to as dynamic testing. Static testing can be (and unfortunately in practice often is) omitted. Dynam
(which is generally considered the beginning of the testing stage). Dynamic testing may begin before the program is
discrete functions). Typical techniques for this are either using stubs/drivers or execution from a debugger envir
tested to a large extent interactively ("on the fly"), with results displayed immediately after each calculation or text

Software verification and validation

Software testing is used in association with verification and validation:[19]

- Verification: Have we built the software right? (i.e., does it match the specification).
- Validation: Have we built the right software? (i.e., is this what the customer wants).

The terms verification and validation are commonly used interchangeably in the industry; it is also common to see
Glossary of Software Engineering Terminology:

> Verification is the process of evaluating a system or component to determine whether the products of a given de
> phase.
>
> Validation is the process of evaluating a system or component during or at the end of the development process t

The software testing team

Software testing can be done by software testers. Until the 1980s the term "software tester" was used generally, bu
and the different goals in software testing,[20] different roles have been established: *manager*, *test lead*, *test designer*

Software quality assurance (SQA)

Though controversial, software testing is a part of the software quality assurance (SQA) process.[12] In SQA, so
development process rather than just the artefacts such as documentation, code and systems. They examine and
faults that end up in the delivered software: the so-called *defect rate*.

What constitutes an "acceptable defect rate" depends on the nature of the software; A flight simulator video gam
airplane.

Although there are close links with SQA, testing departments often exist independently, and there may be no SQA

Software testing is a task intended to detect defects in software by contrasting a computer program's expected re
(quality assurance) is the implementation of policies and procedures intended to prevent defects from occurring in th

Testing methods

The box approach

Software testing methods are traditionally divided into white- and black-box testing. These two approaches are usec
test cases.

White box testing

White box testing is when the tester has access to the internal data structures and algorithms including the cod

Types of white box testing

The following types of white box testing exist:

- API testing (application programming interface) - testing of the application using public and private APIs
- Code coverage - creating tests to satisfy some criteria of code coverage (e.g., the test designer can create test
- Fault injection methods - improving the coverage of a test by introducing faults to test code paths
- Mutation testing methods
- Static testing - White box testing includes all static testing

Test coverage

White box testing methods can also be used to evaluate the completeness of a test suite that was created with b
parts of a system that are rarely tested and ensures that the most important function points have been tested.[2

Two common forms of code coverage are:

- *Function coverage*, which reports on functions executed
- *Statement coverage*, which reports on the number of lines executed to complete the test

They both return a code coverage metric, measured as a percentage.

Black box testing

Black box testing treats the software as a "black box"—without any knowledge of internal implementation. Black
analysis, all-pairs testing, fuzz testing, model-based testing, exploratory testing and specification-based testing.

Specification-based testing: Specification-based testing aims to test the functionality of software according
and only sees the output from, the test object. This level of testing usually requires thorough test cases to be pr
the output value (or behavior), either "is" or "is not" the same as the expected value specified in the test case.

Specification-based testing is necessary, but it is insufficient to guard against certain risks.[23]

Advantages and disadvantages: The black box tester has no "bonds" with the code, and a tester's percep
and you shall receive," black box testers find bugs where programmers do not. On the other hand, black box tes
flashlight," because the tester doesn't know how the software being tested was actually constructed. As a result,
something that could have been tested by only one test case, and/or (2) some parts of the back-end are not test

Therefore, black box testing has the advantage of "an unaffiliated opinion", on the one hand, and the disadvantage c

Grey box testing

Grey box testing is the combination of black box testing and white box testing. **Grey box testing** (American sp
structures and algorithms for purposes of designing the test cases, but testing at the user, or black-box level. Man
because the input and output are clearly outside of the "black-box" that we are calling the system under test. This c
between two modules of code written by two different developers, where only the interfaces are exposed for test. Ho
would not normally be able to change the data outside of the system under test. Grey box testing may also include
messages.

Testing levels

Tests are frequently grouped by where they are added in the software development process, or by the level of specifi

Unit testing

Unit testing refers to tests that verify the functionality of a specific section of code, usually at the function level. the minimal unit tests include the constructors and destructors.[25]

These type of tests are usually written by developers as they work on code (white-box style), to ensure that the spe tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a pie software uses work independently of each other.

Unit testing is also called *component testing*.

Integration testing

Integration testing is any type of software testing that seeks to verify the interfaces between components ag iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interfac

Integration testing works to expose defects in the interfaces and interaction between integrated components (corresponding to elements of the architectural design are integrated and tested until the software works as a system.

System testing

System testing tests a completely integrated system to verify that it meets its requirements.[27]

System integration testing

System integration testing verifies that a system is integrated to any external or third-party systems defined in the s

Regression testing

Regression testing focuses on finding defects after a major code change has occurred. Specifically, it seeks to regressions occur whenever software functionality that was previously working correctly stops working as intended changes, when the newly developed part of the software collides with the previously existing code. Common me checking whether previously fixed faults have re-emerged. The depth of testing depends on the phase in the release for changes added late in the release or deemed to be risky, to very shallow, consisting of positive tests on each featu

Acceptance testing

Acceptance testing can mean one of two things:

1. A smoke test is used as an acceptance test prior to introducing a new build to the main testing process, i.e. befo
2. Acceptance testing performed by the customer, often in their lab environment on their own hardware, is known as part of the hand-off process between any two phases of development.[*citation needed*]

Alpha testing

Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team software as a form of internal acceptance testing, before the software goes to beta testing.[28]

Beta testing

Beta testing comes after alpha testing and can be considered a form of external user acceptance testing. Versions of outside of the programming team. The software is released to groups of people so that further testing can ensure available to the open public to increase the feedback field to a maximal number of future users.[*citation needed*]

Non-functional testing

Special methods exist to test non-functional aspects of software. In contrast to functional testing, which establish expected behavior defined in the design requirements), non-functional testing verifies that the software functions pro injection, in the form of fuzzing, is an example of non-functional testing. Non-functional testing, especially for softw invalid or unexpected inputs, thereby establishing the robustness of input validation routines as well as error-handl from the software fault injection page; there are also numerous open-source and free software tools available that pe

Software performance testing and load testing

Performance testing is executed to determine how fast a system or sub-system performs under a particular workloa system, such as scalability, reliability and resource usage. Load testing is primarily concerned with testing that quantities of data or a large number of users. This is generally referred to as software scalability. The related load referred to as *endurance testing*.

Volume testing is a way to test functionality. *Stress testing* is a way to test reliability. *Load testing* is a way to test testing are. The terms load testing, performance testing, reliability testing, and volume testing, are often used interc

Stability testing

Stability testing checks to see if the software can continuously function well in or above an acceptable period. This a endurance) testing.

Usability testing

Usability testing is needed to check if the user interface is easy to use and understand.It approach towards the use o

Security testing

Security testing is essential for software that processes confidential data to prevent system intrusion by hackers.

Internationalization and localization

The general ability of software to be internationalized and localized can be automatically tested without actual tra
still works, even after it has been translated into a new language or adapted for a new culture (such as different curr

Actual translation to human languages must be tested, too. Possible localization failures include:

- Software is often localized by translating a list of strings out of context, and the translator may choose the wron
- If several people translate strings, technical terminology may become inconsistent.
- Literal word-for-word translations may sound inappropriate, artificial or too technical in the target language.
- Untranslated messages in the original language may be left hard coded in the source code.
- Some messages may be created automatically in run time and the resulting string may be ungrammatical, functi
- Software may use a keyboard shortcut which has no function on the source language's keyboard layout, but is us
- Software may lack support for the character encoding of the target language.
- Fonts and font sizes which are appropriate in the source language, may be inappropriate in the target language; small.
- A string in the target language may be longer than the software can handle. This may make the string partly in
- Software may lack proper support for reading or writing bi-directional text.
- Software may display images with text that wasn't localized.
- Localized operating systems may have differently-named system configuration files and environment variables an

To avoid these and other localization problems, a tester who knows the target language must run the program w
readable, translated correctly in context and don't cause failures.

Destructive testing

Destructive testing attempts to cause the software or a sub-system to fail, in order to test its robustness.

The testing process

Traditional CMMI or waterfall development model

A common practice of software testing is that testing is performed by an independent group of testers after the fu
practice often results in the testing phase being used as a project buffer to compensate for project delays, thereby co

Another practice is to start software testing at the same moment the project starts and it is a continuous process un

Agile or Extreme development model

In counterpoint, some emerging software disciplines such as extreme programming and the agile software developme
this process, unit tests are written first, by the software engineers (often with pair programming in the extreme pro
expected to. Then as code is written it passes incrementally larger portions of the test suites. The test suites ar
discovered, and they are integrated with any regression tests that are developed. Unit tests are maintained along wi
build process (with inherently interactive tests being relegated to a partially manual build acceptance process). Th
where software updates can be published to the public frequently. [33] [34]

A sample testing cycle

Although variations exist between organizations, there is a typical cycle for testing.[35] The sample below is commo

- **Requirements analysis**: Testing should begin in the requirements phase of the software development life cy
  determining what aspects of a design are testable and with what parameters those tests work.
- **Test planning**: Test strategy, test plan, testbed creation. Since many activities will be carried out during test
- **Test development**: Test procedures, test scenarios, test cases, test datasets, test scripts to use in testing soft
- **Test execution**: Testers execute the software based on the plans and test documents then report any errors fo
- **Test reporting**: Once testing is completed, testers generate metrics and make final reports on their test effort
- **Test result analysis**: Or Defect Analysis, is done by the development team usually along with the client, in
  software working properly) or deferred to be dealt with later.
- **Defect Retesting**: Once a defect has been dealt with by the development team, it is retested by the testing t
- **Regression testing**: It is common to have a small test program built of a subset of tests, for each integration
  delivery has not ruined anything, and that the software product as a whole is still working correctly.
- **Test Closure**: Once the test meets the exit criteria, the activities such as capturing the key outputs, lessons le

used as a reference for future projects.

Automated testing

Many programming groups are relying more and more on automated testing, especially groups that use test-dr
continuous integration software will run tests automatically every time code is checked into a version control system

While automation cannot reproduce everything that a human can do (and all the ways they think of doing it), it c
developed test suite of testing scripts in order to be truly useful.

Testing tools

Program testing and fault detection can be aided significantly by testing tools and debuggers. Testing/debug tools i

- Program monitors, permitting full or partial monitoring of program code including:
  - Instruction set simulator, permitting complete instruction level monitoring and trace facilities
  - Program animation, permitting step-by-step execution and conditional breakpoint at source level or in mach
  - Code coverage reports
- Formatted dump or symbolic debugging, tools allowing inspection of program variables on error or at chosen poi
- Automated functional GUI testing tools are used to repeat system-level tests through the GUI
- Benchmarks, allowing run-time performance comparisons to be made
- Performance analysis (or profiling tools) that can help to highlight hot spots and resource usage

Some of these features may be incorporated into an Integrated Development Environment (IDE).

- A regression testing technique is to have a standard set of tests, which cover existing functionality that result in
  data, where there should not be differences, using a tool like diffkit. Differences detected indicate unexpected fun

Measurement in software testing

Usually, quality is constrained to such topics as correctness, completeness, security,[*citation needed*] but can also in
ISO/IEC 9126, such as capability, reliability, efficiency, portability, maintainability, compatibility, and usability.

There are a number of frequently-used software measures, often called *metrics*, which are used to assist in determini

Testing artifacts

Software testing process can produce several artifacts.

Test plan

A test specification is called a test plan. The developers are well aware what test plans will be executed and this
idea is to make them more cautious when developing their code or making additional changes. Some companies

Traceability matrix

A traceability matrix is a table that correlates requirements or design documents to test documents. It is used t
the test results are correct.

Test case

A test case normally consists of a unique identifier, requirement references from a design specification, precondit
output, expected result, and actual result. Clinically defined a test case is an input and an expected result.[36] T
whereas other test cases described in more detail the input scenario and what results might be expected. It can
separate test procedure that can be exercised against multiple test cases, as a matter of economy) but with one
ID, test step, or order of execution number, related requirement(s), depth, test category, author, and check boxe
test cases may also contain prerequisite states or steps, and descriptions. A test case should also contain a place
document, spreadsheet, database, or other common repository. In a database system, you may also be able to se
configuration was used to generate those results. These past results would usually be stored in a separate table.

Test script

The test script is the combination of a test case, test procedure, and test data. Initially the term was derived fro
Today, test scripts can be manual, automated, or a combination of both.

Test suite

The most common term for a collection of test cases is a test suite. The test suite often also contains more detai
contains a section where the tester identifies the system configuration used during testing. A group of test cases
following tests.

Test data

In most cases, multiple sets of values or data are used to test the same functionality of a particular feature. All t
in separate files and stored as test data. It is also useful to provide this data to the client and with the product o

Test harness

The software, tools, samples of data input and output, and configurations are all referred to collectively as a test

Certifications

Several certification programs exist to support the professional aspirations of software testers and quality assura
applicant to demonstrate the ability to test software. No certification is based on a widely accepted body of knowl
certification.[37] Certification itself cannot measure an individual's productivity, their skill, or practical knowle
tester.[38]

Software testing certification types

- *Exam-based*: Formalized exams, which need to be passed; can also be learned by self-study [e.g., for ISTQB o
- *Education-based*: Instructor-led sessions, where each course has to be passed [e.g., International Institute for

Testing certifications

- Certified Associate in Software Testing (CAST) offered by the Quality Assurance Institute (QAI)[40]
- CATe offered by the *International Institute for Software Testing*[41]
- Certified Manager in Software Testing (CMST) offered by the Quality Assurance Institute (QAI)[40]
- Certified Software Tester (CSTE) offered by the Quality Assurance Institute (QAI)[40]
- Certified Software Test Professional (CSTP) offered by the *International Institute for Software Testing*[41]
- CSTP (TM) (Australian Version) offered by *K. J. Ross & Associates*[42]
- ISEB offered by the Information Systems Examinations Board
- ISTQB Certified Tester, Foundation Level (CTFL) offered by the International Software Testing Qualificati
- ISTQB Certified Tester, Advanced Level (CTAL) offered by the International Software Testing Qualificatio
- TMPF TMap Next Foundation offered by the *Examination Institute for Information Science*[45]
- TMPA TMap Next Advanced offered by the *Examination Institute for Information Science*[45]

Quality assurance certifications

- CMSQ offered by the *Quality Assurance Institute* (QAI).[40]
- CSQA offered by the *Quality Assurance Institute* (QAI)[40]
- CSQE offered by the American Society for Quality (ASQ)[46]
- CQIA offered by the American Society for Quality (ASQ)[46]

Controversy

Some of the major software testing controversies include:

What constitutes responsible software testing?

Members of the "context-driven" school of testing[47] believe that there are no "best practices" of testing, but rat
testing practices to suit each unique situation.[48]

Agile vs. traditional

Should testers learn to work under conditions of uncertainty and constant change or should they aim at process
popularity since 2006 mainly in commercial circles,[49][50] whereas government and military[51] software provid
the Waterfall model).[*citation needed*]

Exploratory test vs. scripted[52]

Should tests be designed at the same time as they are executed or should they be designed beforehand?

Manual testing vs. automated

Some writers believe that test automation is so expensive relative to its value that it should be used sparingly.[5
should write unit-tests of the XUnit type before coding the functionality. The tests then can be considered as a

Software design vs. software implementation

Should testing be carried out only at the end or throughout the whole process?

Who watches the watchmen?

The idea is that any form of observation is also an interaction—the act of testing can also affect that which is b

References

1. Exploratory Testing (http://www.kaner.com/pdfs/ETatQAI.pdf), Cem Kaner, Florida Institute of Technology, *Conference*, Orlando, FL, November 2006
2. Leitner, A., Ciupa, I., Oriol, M., Meyer, B., Fiva, A., "Contract Driven Development = Test Driven Developmer ns/cdd_leitner_esec_fse_2007.pdf), Proceedings of ESEC/FSE'07: European Software Engineering Conference a Engineering 2007, (Dubrovnik, Croatia), September 2007
3. Software errors cost U.S. economy $59.5 billion annually (http://www.abeacha.com/NIST_press_release_bugs_
4. Myers, Glenford J. (1979). *The Art of Software Testing*. John Wiley and Sons. ISBN 0-471-04328-1.
5. Company, People's Computer (1987). "Dr. Dobb's journal of software tools for the professional programmer". *Dr* (M&T Pub) **12** (1-6): 116. http://books.google.com/?id=7RoIAAAAIAAJ.
6. Gelperin, D.; B. Hetzel (1988). "The Growth of Software Testing". *CACM* **31** (6). ISSN 0001-0782.
7. *until 1956 it was the debugging oriented period, when testing was often associated to debugging: there was no cle* (1988). "The Growth of Software Testing". *CACM* **31** (6). ISSN 0001-0782.
8. *From 1957–1978 there was the demonstration oriented period where debugging and testing was distinguished nov* Gelperin, D.; B. Hetzel (1988). "The Growth of Software Testing". *CACM* **31** (6). ISSN 0001-0782.
9. *The time between 1979–1982 is announced as the destruction oriented period, where the goal was to find errors.* *CACM* **31** (6). ISSN 0001-0782.
10. *1983–1987 is classified as the evaluation oriented period: intention here is that during the software lifecycle a pr* Hetzel (1988). "The Growth of Software Testing". *CACM* **31** (6). ISSN 0001-0782.
11. *From 1988 on it was seen as prevention oriented period where tests were to demonstrate that software satisfies* Hetzel (1988). "The Growth of Software Testing". *CACM* **31** (6). ISSN 0001-0782.
12. Kaner, Cem; Falk, Jack and Nguyen, Hung Quoc (1999). *Testing Computer Software, 2nd Ed.*. New York, et al: **Invalid <ref> tag; name "Kaner1" defined multiple times with different content Invalid < different content**
13. Kolawa, Adam; Huizinga, Dorota (2007). *Automated Defect Prevention: Best Practices in Software Managemen* http://www.wiley.com/WileyCDA/WileyTitle/productCd-0470042125.html.
14. Kolawa, Adam; Huizinga, Dorota (2007). *Automated Defect Prevention: Best Practices in Software Managemen* http://www.wiley.com/WileyCDA/WileyTitle/productCd-0470042125.html.
15. Section 1.1.2, Certified Tester Foundation Level Syllabus (http://www.istqb.org/downloads/syllabi/SyllabusFou
16. Kaner, Cem; James Bach, Bret Pettichord (2001). *Lessons Learned in Software Testing: A Context-Driven Appr*
17. McConnell, Steve (2004). *Code Complete* (2nd ed.). Microsoft Press. pp. 960. ISBN 0-7356-1967-0.
18. Principle 2, Section 1.3, Certified Tester Foundation Level Syllabus (http://www.bcs.org/upload/pdf/istqbsyll.p
19. Tran, Eushiuan (1999). "Verification/Validation/Certification". in Koopman, P.. *Topics in Dependable Embedde* http://www.ece.cmu.edu/~koopman/des_s99/verification/index.html. Retrieved 2008-01-13.
20. see D. Gelperin and W.C. Hetzel
21. Introduction (http://www.bullseye.com/coverage.html#intro), Code Coverage Analysis, Steve Cornett
22. Laycock, G. T. (1993) (PostScript). *The Theory and Practice of Specification Based Software Testing*. Dept of C http://www.mcs.le.ac.uk/people/gtl1/thesis.ps.gz. Retrieved 2008-02-13.
23. Bach, James (June 1999). "Risk and Requirements-Based Testing" (PDF). *Computer* **32** (6): 113–114. http://wv 2008-08-19.
24. Savenkov, Roman (2008). *How to Become a Software Tester*. Roman Savenkov Consulting. p. 159. ISBN 978-0-(
25. Binder, Robert V. (1999). *Testing Object-Oriented Systems: Objects, Patterns, and Tools*. Addison-Wesley Profe
26. Beizer, Boris (1990). *Software Testing Techniques* (Second ed.). New York: Van Nostrand Reinhold. pp. 21,430.
27. IEEE (1990). *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*. Nev

28. van Veenendaal, Erik. "Standard glossary of terms used in Software Testing". http://www.astqb.org/educational
29. Globalization Step-by-Step: The World-Ready Approach to Testing. Microsoft Developer Network (http://msdn
30. e)Testing Phase in Software Testing:- (http://www.etestinghub.com/testing_lifecycles.php#2)
31. Myers, Glenford J. (1979). *The Art of Software Testing*. John Wiley and Sons. pp. 145–146. ISBN 0-471-04328-1
32. Dustin, Elfriede (2002). *Effective Software Testing*. Addison Wesley. p. 3. ISBN 0-20179-429-2.
33. Marchenko, Artem (November 16, 2007). "XP Practice: Continuous Integration". http://agilesoftwaredevelopmer
34. Gurses, Levent (February 19, 2007). "Agile 101: What is Continuous Integration?". http://www.jacoozi.com/blo
35. Pan, Jiantao (Spring 1999). "Software Testing (18-849b Dependable Embedded Systems)". *Topics in Dependable* Carnegie Mellon University. http://www.ece.cmu.edu/~koopman/des_s99/sw_testing/.
36. IEEE (1998). *IEEE standard for software test documentation*. New York: IEEE. ISBN 0-7381-1443-X.
37. Kaner, Cem (2001). "NSF grant proposal to "lay a foundation for significant improvements in the quality of acac http://www.testingeducation.org/general/nsf_grant.pdf.
38. Kaner, Cem (2003). "Measuring the Effectiveness of Software Testers" (pdf). http://www.testingeducation.org/a
39. Black, Rex (December 2008). *Advanced Software Testing- Vol. 2: Guide to the ISTQB Advanced Certification a* ISBN 1933952369.
40. Quality Assurance Institute (http://www.qaiglobalinstitute.com/)
41. International Institute for Software Testing (http://www.testinginstitute.com/)
42. K. J. Ross & Associates (http://www.kjross.com.au/cstp/)
43. "ISTQB". http://www.istqb.org/.
44. "ISTQB in the U.S.". http://www.astqb.org/.
45. EXIN: Examination Institute for Information Science (http://www.exin-exams.com)
46. American Society for Quality (http://www.asq.org/)
47. context-driven-testing.com (http://www.context-driven-testing.com)
48. Article on taking agile traits without the agile method. (http://www.technicat.com/writing/process.html)
49. “We’re all part of the story” (http://stpcollaborative.com/knowledge/272-were-all-part-of-the-story) by David S
50. IEEE article about differences in adoption of agile trends between experienced managers vs. young students of tl n.jsp?url=/iel5/10705/33795/01609838.pdf?temp=x). See also Agile adoption study from 2007 (http://www.aml
51. Willison, John S. (April 2004). "Agile Software Development for an Agile Force". *CrossTalk* (STSC) (April 2004) http://web.archive.org/web/20051029135922/http://www.stsc.hill.af.mil/crosstalk/2004/04/0404willison.html.
52. IEEE article on Exploratory vs. Non Exploratory testing (http://ieeexplore.ieee.org/iel5/10351/32923/01541817.
53. An example is Mark Fewster, Dorothy Graham: *Software Test Automation*. Addison Wesley, 1999, ISBN 0-201-3
54. Microsoft Development Network Discussion on exactly this topic (http://channel9.msdn.com/forums/Coffeehous

External links

- Software testing tools and products (http://www.dmoz.org/Computers/Programming/Software_Testing/Produc
- "Software that makes Software better" Economist.com (http://www.economist.com/science/tq/displaystory.cfm?
- Automated software testing metrics including manual testing metrics (http://www.innovativedefense.com/img/U

Unit Tests

In computer programming, **unit testing** is a method by which individual units of source code are tested to dete application. In procedural programming a unit may be an individual function or procedure. Unit tests are created by

Ideally, each test case is independent from the others: substitutes like method stubs, mock objects,[1] fakes and test are typically written and run by software developers to ensure that code meets its design and behaves as intended. to being formalized as part of build automation.

Benefits

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct.[2] A satisfy. As a result, it affords several benefits. Unit tests find problems early in the development cycle.

Facilitates change

Unit testing allows the programmer to refactor code at a later date, and make sure the module still works correctly functions and methods so that whenever a change causes a fault, it can be quickly identified and fixed.

Readily-available unit tests make it easy for the programmer to check whether a piece of code is still working proper

In continuous unit testing environments, through the inherent practice of sustained maintenance, unit tests will co in the face of any change. Depending upon established development practices and unit test coverage, up-to-the-seco separately.

Simplifies integration

Unit testing may reduce uncertainty in the units themselves and can be used in a bottom-up testing style approach parts, integration testing becomes much easier.

An elaborate hierarchy of unit tests does not equal integration testing. Integration with peripheral units should Integration testing typically still relies heavily on humans testing manually; high-level or global-scope testing can and cheaper.[*citation needed*]

Documentation

Unit testing provides a sort of living documentation of the system. Developers looking to learn what functionality is a basic understanding of the unit's API.

Unit test cases embody characteristics that are critical to the success of the unit. These characteristics can indicat that are to be trapped by the unit. A unit test case, in and of itself, documents these critical characteristics, alth code to document the product in development.

By contrast, ordinary narrative documentation is more susceptible to drifting from the implementation of the prog relaxed practices in keeping documents up-to-date).

Design

When software is developed using a test-driven approach, the unit test may take the place of formal design. Each u observable behaviour. The following Java example will help illustrate this point.

Here is a test class that specifies a number of elements of the implementation. First, that there must be an int constructor called AdderImpl. It goes on to assert that the Adder interface should have a method called add, with the behaviour of this method for a small range of values.

```
public class TestAdder {
   public void testSum() {
      Adder adder = new AdderImpl();
      assert(adder.add(1, 1) == 2);
      assert(adder.add(1, 2) == 3);
      assert(adder.add(2, 2) == 4);
      assert(adder.add(0, 0) == 0);
      assert(adder.add(-1, -2) == -3);
      assert(adder.add(-1, 1) == 0);
      assert(adder.add(1234, 988) == 2222);
   }
}
```

In this case the unit test, having been written first, acts as a design document specifying the form and behaviour of the programmer. Following the "do the simplest thing that could possibly work" practice, the easiest solution that w

```
interface Adder {
   int add(int a, int b);
}
class AdderImpl implements Adder {
   int add(int a, int b) {
      return a + b;
   }
}
```

Unlike other diagram-based design methods, using a unit-test as a design has one significant advantage. The implementation adheres to the design. With the unit-test design method, the tests will never pass if the developer d

It is true that unit testing lacks some of the accessibility of a diagram, but UML diagrams are now easily generated to IDEs). Free tools, like those based on the xUnit framework, outsource to another system the graphical rendering

Separation of interface from implementation

Because some classes may have references to other classes, testing a class can frequently spill over into testing
database: in order to test the class, the tester often writes code that interacts with the database. This is a mist
boundary, and especially should not cross such process/network boundaries because this can introduce unaccep
boundaries turns unit tests into integration tests, and when test cases fail, makes it less clear which component is ca

Instead, the software developer should create an abstract interface around the database queries, and then impl
necessary attachment from the code (temporarily reducing the net effective coupling), the independent unit can be
results in a higher quality unit that is also more maintainable.

Unit testing limitations

Testing cannot be expected to catch every error in the program: it is impossible to evaluate every execution path
Additionally, unit testing by definition only tests the functionality of the units themselves. Therefore, it will not c
performed across multiple units, or non-functional test areas such as performance). Unit testing should be done
software testing, unit tests can only show the presence of errors; they cannot show the absence of errors.

Software testing is a combinatorial problem. For example, every boolean decision statement requires at least two t
As a result, for every line of code written, programmers often need 3 to 5 lines of test code.[3] This obviously tak
many problems that cannot easily be tested at all – for example those that are nondeterministic or involve multip
least as buggy as the code it is testing. Fred Brooks in The Mythical Man-Month quotes: *never take two chronor*
contradict, how do you know which one is correct?

To obtain the intended benefits from unit testing, rigorous discipline is needed throughout the software developmen
have been performed, but also of all changes that have been made to the source code of this or any other unit in th
of the unit fails a particular test that it had previously passed, the version-control software can provide a list of th
that time.

It is also essential to implement a sustainable process for ensuring that test case failures are reviewed daily an
ingrained into the team's workflow, the application will evolve out of sync with the unit test suite, increasing false p

Applications

Extreme Programming

Unit testing is the cornerstone of Extreme Programming, which relies on an automated unit testing framework.
xUnit, or created within the development group.

Extreme Programming uses the creation of unit tests for test-driven development. The developer writes a unit test
because either the requirement isn't implemented yet, or because it intentionally exposes a defect in the existing co
with other tests, pass.

Most code in a system is unit tested, but not necessarily all paths through the code. Extreme Programming ma
traditional "test every execution path" method. This leads developers to develop fewer tests than classical methods
methods have rarely ever been followed methodically enough for all execution paths to have been thoroughly testec
is rarely exhaustive (because it is often too expensive and time-consuming to be economically viable) and provides g

Crucially, the test code is considered a first class project artifact in that it is maintained at the same quality as the
unit testing code to the code repository in conjunction with the code it tests. Extreme Programming's thorough uni
confident code development and refactoring, simplified code integration, accurate documentation, and more modular
test.

Techniques

Unit testing is commonly automated, but may still be performed manually. The IEEE does not favor one over the
instructional document. Nevertheless, the objective in unit testing is to isolate a unit and validate its correctness. A
listed in this article. Conversely, if not planned carefully, a careless manual unit test case may execute as an in
preclude the achievement of most if not all of the goals established for unit testing.

To fully realize the effect of isolation while using an automated approach, the unit or code body under test is exe
words, it is executed outside of the product or calling context for which it was originally created. Testing in such
being tested and other units or data spaces in the product. These dependencies can then be eliminated.

Using an automation framework, the developer codes criteria into the test to verify the unit's correctness. During te
frameworks will also automatically flag these failed test cases and report them in a summary. Depending upon the s

As a consequence, unit testing is traditionally a motivator for programmers to create decoupled and cohesive code
Design patterns, unit testing, and refactoring often work together so that the best solution may emerge.

Unit testing frameworks

Unit testing frameworks are most often third-party products that are not distributed as part of the compiler suite.
a wide variety of languages. Examples of testing frameworks include open source solutions such as the varic
proprietary/commercial solutions such as TBrun, Testwell CTA++ and VectorCAST/C++.

It is generally possible to perform unit testing without the support of a specific framework by writing client code th
or other control flow mechanisms to signal failure. Unit testing without a framework is valuable in that there is a ba
hardly better than having none at all, whereas once a framework is in place, adding unit tests becomes relatively ea
or must be hand-coded.

Language-level unit testing support

Some programming languages support unit testing directly. Their grammar allows the direct declaration of unit
Additionally, the boolean conditions of the unit tests can be expressed in the same syntax as boolean expressio
statements.

Languages that directly support unit testing include:

- Cobra
- D

Notes

1. Fowler, Martin (2007-01-02). "Mocks aren't Stubs". http://martinfowler.com/articles/mocksArentStubs.html. Re
2. Kolawa, Adam; Huizinga, Dorota (2007). *Automated Defect Prevention: Best Practices in Software Managemen* http://www.wiley.com/WileyCDA/WileyTitle/productCd-0470042125.html.
3. Cramblitt, Bob (2007-09-20). "Alberto Savoia sings the praises of software testing". http://searchsoftwarequality Retrieved 2007-11-29.
4. daVeiga, Nada (2008-02-06). "Change Code Without Fear: Utilize a regression safety net". http://www.ddj.com/
5. IEEE Standards Board, "IEEE Standard for Software Unit Testing: An American National Standard, ANSI/IEE EE%201008.pdf) in *IEEE Standards: Software Engineering, Volume Two: Process Standards; 1999 Edition; pub* Software Engineering Technical Committee of the IEEE Computer Society.
6. Bullseye Testing Technology (2006–2008). "Intermediate Coverage Goals". http://www.bullseye.com/coverage.ht

External links

- The evolution of Unit Testing Syntax and Semantics (http://weblogs.asp.net/rosherove/archive/2008/01/17/the
- Unit Testing Guidelines from GeoSoft (http://geosoft.no/development/unittesting.html)
- Test Driven Development (Ward Cunningham's Wiki) (http://c2.com/cgi/wiki?TestDrivenDevelopment)
- Unit Testing 101 for the Non-Programmer (http://www.saravananansubramanian.com/Saravanan/Articles_On_S s.html)
- Step-by-Step Guide to JPA-Enabled Unit Testing (Java EE) (http://www.sizovpoint.com/2010/01/step-by-step

Profiling

In software engineering, **program profiling**, **software profiling** or simply **profiling**, a form of dynamic prog
a program's behavior using information gathered as the program executes. The usual purpose of this analysis is
overall speed, decrease its memory requirement or sometimes both.

- A **(code) profiler** is a **performance analysis** tool that, most commonly, measures only the frequency and (e.g. memory profilers) in addition to more comprehensive profilers, capable of gathering extensive performance
- An instruction set simulator which is also — by necessity — a profiler, can measure the totality of a program's

Gathering program events

Profilers use a wide variety of techniques to collect data, including hardware interrupts, code instrumentation, counters. The usage of profilers is 'called out' in the performance engineering process.

Use of profilers

Program analysis tools are extremely important for understanding program behavior. Computer architects need such Software writers need tools to analyze their programs and identify critical sections of code. Compiler writers often prediction algorithm is performing... (ATOM, PLDI, '94)

The output of a profiler may be:-

- A statistical *summary* of the events observed (a **profile**)

  Summary profile information is often shown annotated against the source code statements where the events occ program.

```
/* ----------- source------------------------ count */
0001            IF X = "A"                  0055
0002              THEN DO
0003                ADD 1 to XCOUNT           0032
0004              ELSE
```

```
0005        IF X = "B"                  0055
```

- A stream of recorded events (a **trace**)

  For sequential programs, a summary profile is usually sufficient, but performance problems in parallel programs
  time relationship of events, thus requiring a full trace to get an understanding of what is happening.
  The size of a (full) trace is linear to the program's instruction path length, making it somewhat impractical. A t
  terminated at another point to limit the output.

- An ongoing interaction with the hypervisor (continuous or periodic monitoring via on-screen display for instance

  This provides the opportunity to switch a trace on or off at any desired point during execution in addition to vi
  provides the opportunity to suspend asynchronous processes at critical points to examine interactions with othe

History

Performance analysis tools existed on IBM/360 and IBM/370 platforms from the early 1970s, usually based on ti
timer intervals to detect "hot spots" in executing code. This was an early example of sampling (see below). In
performance monitoring features.

Profiler-driven program analysis on Unix dates back to at least 1979, when Unix systems included a basic tool "pro
used. In 1982, gprof extended the concept to a complete call graph analysis [1]

In 1994, Amitabh Srivastava and Alan Eustace of Digital Equipment Corporation published a paper describing
profiler. That is, at compile time, it inserts code into the program to be analyzed. That inserted code outputs an
known as "instrumentation".

In 2004, both the gprof and ATOM papers appeared on the list of the 50 most influential PLDI papers of all time.[3

Profiler types based on output

Flat profiler

Flat profilers compute the average call times, from the calls, and do not break down the call times based on the call

Call-graph profiler

Call graph profilers show the call times, and frequencies of the functions, and also the call-chains involved based on

Methods of data gathering

Event-based profilers

The programming languages listed here have event-based profilers:

- Java: the JVMTI (JVM Tools Interface) API, formerly JVMPI (JVM Profiling Interface), provides hooks to pro
  leave.
- .NET: Can attach a profiling agent as a COM server to the CLR. Like Java, the runtime then provides various
  leave, object creation, etc. Particularly powerful in that the profiling agent can rewrite the target application's b
- Python: Python profiling includes the profile module, hotshot (which is call-graph based), and using the 'sys.setp
  python_{call,return,exception}.
- Ruby: Ruby also uses a similar interface like Python for profiling. Flat-profiler in profile.rb, module, and ruby-pr

Statistical profilers

Some profilers operate by sampling. A sampling profiler probes the target program's program counter at regular int
less numerically accurate and specific, but allow the target program to run at near full speed.

The resulting data are not exact, but a statistical approximation. *The actual amount of error is usually more than
the expected error in it is the square-root of n sampling periods.* [4]

In practice, sampling profilers can often provide a more accurate picture of the target program's execution than ot
thus don't have as many side effects (such as on memory caches or instruction decoding pipelines). Also since the
would otherwise be hidden. They are also relatively immune to over-evaluating the cost of small, frequently calle
spent in user mode versus interruptible kernel mode such as system call processing.

Still, kernel code to handle the interrupts entails a minor loss of CPU cycles, diverted cache usage, and is unable t
(microsecond-range activity).

Dedicated hardware can go beyond this: some recent MIPS processors JTAG interface have a PCSAMPLE register,

Some of the most commonly used statistical profilers are AMD CodeAnalyst, Apple Inc. Shark, gprof, Intel VTune a

Instrumenting profilers

Some profilers **instrument** the target program with additional instructions to collect the required information.

Instrumenting the program can cause changes in the performance of the program, potentially causing inaccurate res program execution, typically always slowing it. However, instrumentation can be very specific and be carefully con depends on the placement of instrumentation points and the mechanism used to capture the trace. Hardware suppo on just one machine instruction. The impact of instrumentation can often be deducted (i.e. eliminated by subtractio

gprof is an example of a profiler that uses both instrumentation and sampling. Instrumentation is used to gather ca sampling.

Instrumentation

- **Manual**: Performed by the programmer, e.g. by adding instructions to explicitly calculate runtimes, simply cou Response Measurement standard.
- **Automatic source level**: instrumentation added to the source code by an automatic tool according to an ins
- **Compiler assisted**: Example: "gcc -pg ..." for gprof, "quantify g++ ..." for Quantify
- **Binary translation**: The tool adds instrumentation to a compiled binary. Example: ATOM
- **Runtime instrumentation**: Directly before execution the code is instrumented. The program run is fully su
- **Runtime injection**: More lightweight than runtime instrumentation. Code is modified at runtime to have jur

Interpreter instrumentation

- **Interpreter debug** options can enable the collection of performance metrics as the interpreter encounters eac three examples that usually have complete control over execution of the target code, thus enabling extremely cor

Hypervisor/Simulator

- **Hypervisor**: Data are collected by running the (usually) unmodified program under a hypervisor. Example: SI
- **Simulator** and **Hypervisor**: Data collected interactively and selectively by running the unmodified program Interactive test/debug) and IBM OLIVER (CICS interactive test/debug).

References

1. *gprof: a Call Graph Execution Profiler* (http://docs.freebsd.org/44doc/psd/18.gprof/paper.pdf)
2. Atom: A system for building customized program analysis tools, Amitabh Srivastava and Alan Eustace, 1994 (ht (download (http://www.ece.cmu.edu/~ece548/tools/atom/man/wrl_94_2.pdf))
3. 20 Years of PLDI (1979 - 1999): A Selection, Kathryn S. McKinley, Editor (http://www.cs.utexas.edu/users/mc
4. Statistical Inaccuracy of gprof Output (http://lgl.epfl.ch/teaching/case_tools/doc/gprof/gprof_12.html)

- Dunlavey, “Performance tuning with instruction-level cost derived from call-stack sampling”, ACM SIGPLAN N
- Dunlavey, “Performance Tuning: Slugging It Out!”, Dr. Dobb's Journal, Vol 18, #12, November 1993, pp 18–26

External links

- Article "Need for speed — Eliminating performance bottlenecks (http://www.ibm.com/developerworks/rational/ applications using IBM Rational Application Developer.
- Profiling Runtime Generated and Interpreted Code using the VTune™ Performance Analyzer (http://software.i ng.pdf)

Test-driven Development

Test-driven development (**TDD**) is a software development process that relies on the repetition of a very sho case that defines a desired improvement or new function, then produces code to pass that test and finally refactors having developed or 'rediscovered' the technique, stated in 2003 that TDD encourages simple designs and inspires co

Test-driven development is related to the test-first programming concepts of extreme programming, begun in 19 right.[3]

Programmers also apply the concept to improving and debugging legacy code developed with older techniques.[4]

Requirements

Test-driven development requires developers to create automated unit tests that define code requirements (immedia either true or false. Passing the tests confirms correct behavior as developers evolve and refactor the code. D automatically run sets of test cases.

Test-driven development cycle

The following sequence is based on the book *Test-Driven Development by Example*[1].

Add a test

In **test-driven development**, each new feature begins with writing a test. This test must inevitably fail becau fail, then either the proposed "new" feature already exists or the test is defective.) To write a test, the developer m developer can accomplish this through use cases and user stories that cover the requirements and exception conditi This is a differentiating feature of test-driven development versus writing unit tests *after* the code is written: it m subtle but important difference.

Run all tests and see if the new one fails

This validates that the test harness is working correctly and that the new test does not mistakenly pass without rec it rules out the possibility that the new test will always pass, and therefore be worthless. The new test should also not entirely guarantee) that it is testing the right thing, and will pass only in intended cases.

Write some code

The next step is to write some code that will cause the test to pass. The new code written at this stage will not be acceptable because later steps will improve and hone it.

It is important that the code written is *only* designed to pass the test; no further (and therefore untested) functional

Run the automated tests and see them succeed

If all test cases now pass, the programmer can be confident that the code meets all the tested requirements. This is

Refactor code

Now the code can be cleaned up as necessary. By re-running the test cases, the developer can be confident that cod removing duplication is an important aspect of any software design. In this case, however, it also applies to removir example magic numbers or strings that were repeated in both, in order to make the test pass in step 3.

Repeat

Starting with another new test, the cycle is then repeated to push forward the functionality. The size of the steps run. If new code does not rapidly satisfy a new test, or other tests fail unexpectedly, the programmer should undo helps by providing revertible checkpoints. When using external libraries it is important not to make increments t unless there is some reason to believe that the library is buggy or is not sufficiently feature-complete to serve all the

Development style

There are various aspects to using test-driven development, for example the principles of "keep it simple, stupid" (only the code necessary to pass tests, designs can be cleaner and clearer than is often achieved by other methods.[principle "Fake it till you make it".

To achieve some advanced design concept (such as a design pattern), tests are written that will generate that desig all required tests. This can be unsettling at first but it allows the developer to focus only on what is important.

Write the tests first. The tests should be written before the functionality that is being tested. This has bee written for testability, as the developers must consider how to test the application from the outset, rather than wo written. When writing feature-first code, there is a tendency by developers and the development organisations to pu

First fail the test cases. The idea is to ensure that the test really works and can catch an error. Once this is coined the "test-driven development mantra", known as red/green/refactor where red means *fail* and green is *pass*.

Test-driven development constantly repeats the steps of adding test cases that fail, passing them, and refacto programmer's mental model of the code, boosts confidence and increases productivity.

Advanced practices of test-driven development can lead to Acceptance Test-driven development (ATDD) where the which then drive the traditional unit test-driven development (UTDD) process.[5] This process ensures the custom their requirements. With ATDD, the development team now has a specific target to satisfy, the acceptance tests wants from that user story.

Benefits

A 2005 study found that using TDD meant writing more tests and, in turn, programmers that wrote more tests ten more direct correlation between TDD and productivity were inconclusive.[7]

Programmers using pure TDD on new ("greenfield") projects report they only rarely feel the need to invoke a debu unexpectedly, reverting the code to the last version that passed all tests may often be more productive than debuggi

Test-driven development offers more than just simple validation of correctness, but can also drive the design of a functionality will be used by clients (in the first case, the test cases). So, the programmer is concerned with the Design by Contract as it approaches code through test cases rather than through mathematical assertions or precon

Test-driven development offers the ability to take small steps when required. It allows a programmer to focus on
cases and error handling are not considered initially, and tests to create these extraneous circumstances are implen
written code is covered by at least one test. This gives the programming team, and subsequent users, a greater level

While it is true that more code is required with TDD than without TDD because of the unit test code, total code
to limit the number of defects in the code. The early and frequent nature of the testing helps to catch defects early
expensive problems. Eliminating defects early in the process usually avoids lengthy and tedious debugging later in tl

TDD can lead to more modularized, flexible, and extensible code. This effect often comes about because the metl
small units that can be written and tested independently and integrated together later. This leads to smaller, mor
mock object design pattern also contributes to the overall modularization of the code because this pattern requires t
mock versions for unit testing and "real" versions for deployment.

Because no more code is written than necessary to pass a failing test case, automated tests tend to cover every code
to an existing if statement, the developer would first have to write a failing test case that motivates the branch.
thorough: they will detect any unexpected changes in the code's behaviour. This detects problems that can arise w
functionality.

Vulnerabilities

- Test-driven development is difficult to use in situations where full functional tests are required to determine succ
  work with databases, and some that depend on specific network configurations. TDD encourages developers to p
  the logic that is in testable library code, using fakes and mocks to represent the outside world.
- Management support is essential. Without the entire organization believing that test-driven development is goin
  tests is wasted.[10]
- Unit tests created in a test-driven development environment are typically created by the developer who will also
  same blind spots with the code: If, for example, a developer does not realize that certain input parameters must
  input parameters. If the developer misinterprets the requirements specification for the module being developed, b
- The high number of passing unit tests may bring a false sense of security, resulting in fewer additional software t
- The tests themselves become part of the maintenance overhead of a project. Badly written tests, for example one
  to failure, are expensive to maintain. There is a risk that tests that regularly generate false failures will be ignore
  possible to write tests for low and easy maintenance, for example by the reuse of error strings, and this should b
- The level of coverage and testing detail achieved during repeated TDD cycles cannot easily be re-created at a lat
  time goes by. If a poor architecture, a poor design or a poor testing strategy leads to a late change that makes d
  fixed. Merely deleting, disabling or rashly altering them can lead to undetectable holes in the test coverage.

Code Visibility

Test suite code clearly has to be able to access the code it is testing. On the other hand normal design criteria su
should not be compromised. Therefore unit test code for TDD is usually written within the same project or module a

In object oriented design this still does not provide access to private data and methods. Therefore, extra work may b
use reflection to access fields that are marked private.[11] Alternatively, an inner class can be used to hold the uni
attributes. In the .NET Framework and some other programming languages, partial classes may be used to expose p

It is important that such testing hacks do not remain in the production code. In C and other languages, compile
additional classes and indeed all other test-related code to prevent them being compiled into the released code. T
which is unit tested. The regular running of fewer but more comprehensive, end-to-end, integration tests on the fina
code exists that subtly relies on aspects of the test harness.

There is some debate among practitioners of TDD, documented in their blogs and other writings, as to whether it
argue that it should be sufficient to test any class through its public interface as the private members are a mere i
without breaking numbers of tests. Others say that crucial aspects of functionality may be implemented in privat
public interface only obscures the issue: unit testing is about testing the smallest unit of functionality possible.[12][1

Fakes, mocks and integration tests

Unit tests are so named because they each test *one unit* of code. A complex module may have a thousand unit test
process boundaries in a program, let alone network connections. Doing so introduces delays that make tests run slo
dependencies on external modules or data also turns *unit tests* into *integration tests*. If one module misbehaves in
look for the cause of the failure.

When code under development relies on a database, a web service, or any other external process or service, enforcin
design more modular, more testable and more reusable code.[14] Two steps are necessary:

1. Whenever external access is going to be needed in the final design, an interface should be defined that describes
   for a discussion of the benefits of doing this regardless of TDD.
2. The interface should be implemented in two ways, one of which really accesses the external process, and the oth
   a message such as “Person object saved” to a trace log, against which a test assertion can be run to verify correc
   assertions that can make the test fail, for example, if the person's name and other data are not as expected. Fak

store or user, can help the test process by always returning the same, realistic data that tests can rely upon. The routines can be developed and reliably tested. Fake services other than data stores may also be useful in TDD: F random number services may always return 1. Fake or mock implementations are examples of dependency inject

A corollary of such dependency injection is that the actual database or other external-access code is never tested by tests are needed that instantiate the test-driven code with the "real" implementations of the interfaces discussed a really integration tests. There will be fewer of them, and they need to be run less often than the unit tests. They can xUnit.

Integration tests that alter any persistent store or database should always be designed carefully with consideration o This is often achieved using some combination of the following techniques:

- The TearDown method, which is integral to many test frameworks.
- try...catch...finally exception handling structures where available.
- Database transactions where a transaction atomically includes perhaps a write, a read and a matching delete ope
- Taking a "snapshot" of the database before running any tests and rolling back to the snapshot after each test ru continuous integration system such as CruiseControl.
- Initialising the database to a clean state *before* tests, rather than cleaning up *after* them. This may be relevant w deleting the final state of the database before detailed diagnosis can be performed.

Frameworks such as Moq, jMock, NMock, EasyMock, Typemock, jMockit, Unitils, Mockito, Mockachino, PowerM complex mock objects easier.

References

1. Beck, K. Test-Driven Development by Example, Addison Wesley, 2003
2. Lee Copeland (December 2001). "Extreme Programming". Computerworld. http://www.computerworld.com/soft January 11, 2011.
3. Newkirk, JW and Vorontsov, AA. *Test-Driven Development in Microsoft .NET*, Microsoft Press, 2004.
4. Feathers, M. Working Effectively with Legacy Code, Prentice Hall, 2004
5. Koskela, L. "Test Driven: TDD and Acceptance TDD for Java Developers", Manning Publications, 2007
6. Erdogmus, Hakan; Morisio, Torchiano. "On the Effectiveness of Test-first Approach to Programming". Proceedir 2005. (NRC 47445). http://iit-iti.nrc-cnrc.gc.ca/publications/nrc-47445_e.html. Retrieved 2008-01-14. "We found students who wrote more tests tended to be more productive."
7. Proffitt, Jacob. "TDD Proven Effective! Or is it?". http://theruntime.com/blogs/jacob/archive/2008/01/22/tdd- relationship to quality is problematic at best. Its relationship to productivity is more interesting. I hope there's a very well to me. There is an undeniable correlation between productivity and the number of tests, but that corre outlier compared to roughly half of the TDD group being outside the 95% band)."
8. Llopis, Noel (20 February 2005). "Stepping Through the Looking Glass: Test-Driven Game Development (Part 1 http://www.gamesfromwithin.com/articles/0502/000073.html. Retrieved 2007-11-01. "Comparing [TDD] to the r checking and debugger stepping with code that verifies that your program does exactly what you intended it to
9. Müller, Matthias M.; Padberg, Frank. "About the Return on Investment of Test-Driven Development" (PDF). U http://www.ipd.uka.de/mitarbeiter/muellerm/publications/edser03.pdf. Retrieved 2007-11-01.
10. Loughran, Steve (November 6th, 2006). "Testing" (PDF). HP Laboratories. http://people.apache.org/~stevel/sli
11. Burton, Ross (11/12/2003). "Subverting Java Access Protection for Unit Testing". O'Reilly Media, Inc.. http://v 2009-08-12.
12. Newkirk, James (7 June 2004). "Testing Private Methods/Member Variables - Should you or shouldn't you". Mi http://blogs.msdn.com/jamesnewkirk/archive/2004/06/07/150361.aspx. Retrieved 2009-08-12.
13. Stall, Tim (1 Mar 2005). "How to Test Private and Protected methods in .NET". CodeProject. http://www.code
14. Fowler, Martin (1999). *Refactoring - Improving the design of existing code.* Boston: Addison Wesley Longman, I

External links

- [10] (http://c2.com/cgi/wiki?TestDrivenDevelopment%7CTestDrivenDevelopment) on WikiWikiWeb
- Test or spec? Test and spec? Test from spec! (http://www.eiffel.com/general/monthly_column/2004/september
- Microsoft Visual Studio Team Test from a TDD approach (http://msdn.microsoft.com/en-us/library/ms379625(
- Write Maintainable Unit Tests That Will Save You Time And Tears (http://msdn.microsoft.com/en-us/magazi
- Improving Application Quality Using Test-Driven Development (TDD) (http://www.methodsandtools.com/arch

Refactoring

Code refactoring is "a disciplined way to restructure code",[1] undertaken in order to improve some of the *non-f* series of "refactorings", each of which is a (usually) tiny change in a computer program's source code that does not readability and reduced complexity to improve the maintainability of the source code, as well as a more expressive i

> *By continuously improving the design of code, we make it easier and easier to work with. This is in sharp contr attention paid to expediently adding new features. If you get into the hygienic habit of refactoring continuously,*

Refactoring does not take place in a vacuum, but typically the refactoring process takes place in a context of adding

- "... refactoring and adding new functionality are two different but complementary tasks" -- Scott Ambler

Overview

Refactoring is usually motivated by noticing a code smell.[3] For example the method at hand may be very long, or such problems can be addressed by *refactoring* the source code, or transforming it into a new form that behaves the one or more smaller subroutines. Or for duplicate routines, remove the duplication and utilize one shared function technical debt.

There are two general categories of benefits to the activity of refactoring.

1. Maintainability. It is easier to fix bugs because the source code is easy to read and the intent of its author is eas routines into a set of individually concise, well-named, single-purpose methods. It might be achieved by moving a comments.
2. Extensibility. It is easier to extend the capabilities of the application if it uses recognizable design patterns, and

Before refactoring a section of code, a solid set of automatic unit tests is needed. The tests should demonstrate in a The process is then an iterative cycle of making a small program transformation, testing it to ensure correctness, an undo your last small change and try again in a different way. Through many small steps the program moves fi programming and other agile methodologies describe this activity as an integral part of the software development cy

List of refactoring techniques

Here are some examples of code refactorings; some of these may only apply to certain languages or language typ Fowler's Refactoring Website.[5]

- Techniques that allow for more abstraction
  - Encapsulate Field – force code to access the field with getter and setter methods
  - Generalize Type – create more general types to allow for more code sharing
  - Replace type-checking code with State/Strategy[6]
  - Replace conditional with polymorphism[7]
- Techniques for breaking code apart into more logical pieces
  - Extract Method, to turn part of a larger method into a new method. By breaking down code in smaller piece
  - Extract Class moves part of the code from an existing class into a new class.
- Techniques for improving names and location of code
  - Move Method or Move Field – move to a more appropriate Class or source file
  - Rename Method or Rename Field – changing the name into a new one that better reveals its purpose
  - Pull Up – in OOP, move to a superclass
  - Push Down – in OOP, move to a subclass

Hardware refactoring

While the term *refactoring* originally referred exclusively to refactoring of software code, in recent years code writt The term *hardware refactoring* is used as a shorthand term for refactoring of code in hardware description language hardware engineers,[8] hardware refactoring is to be considered a separate field from traditional code refactoring.

Automated refactoring of analog hardware descriptions (in VHDL-AMS) has been proposed by Zeng and Huss.[9] hardware design. The non-functional measurement that improves is that refactored code can be processed by standa HDLs, albeit manual refactoring, has also been investigated by Synopsys fellow Mike Keating.[10][11] His target designers' productivity.

In the summer of 2008, there was an intense discussion about refactoring of VHDL code on the news://comp.lang refactoring performed by one engineer, and the question to whether or not automated tools for such refactoring exist

As of late 2009, Sigasi is offering automated tool support for VHDL refactoring.[13]

History

In the past refactoring was avoided in development processes. One example of this is that CVS (created in 1984) do

Although refactoring code has been done informally for years, William Opdyke's 1992 Ph.D. dissertation[14] is the fi theory and machinery have long been available as program transformation systems. All of these resources provide a description of how to apply the method and indicators for when you should (or should not) apply the method.

Martin Fowler's book *Refactoring: Improving the Design of Existing Code*[3] is the canonical reference.

The first known use of the term "refactoring" in the published literature was in a September, 1990 article by Wil published in 1992, also used this term.[15]

The term "factoring" has been used in the Forth community since at least the early 1980s[*citation needed*]. Chap subject.

In extreme programming, the Extract Method refactoring technique has essentially the same meaning as factoring i maintained functions.

Automated code refactoring

Many software editors and IDEs have automated refactoring support. Here is a list of a few of these editors, or so-ca

- IntelliJ IDEA (for Java)
- Eclipse's Java Development Toolkit (JDT)
- NetBeans (for Java)
  - and RefactoringNG (http://kenai.com/projects/refactoringng/), a Netbeans module for refactoring where yo
- Embarcadero Delphi
- Visual Studio (for .NET)
- JustCode (addon for Visual Studio)
- ReSharper (addon for Visual Studio)
- Coderush (addon for Visual Studio)
- Visual Assist (addon for Visual Studio with refactoring support for VB, VB.NET. C# and C++)
- DMS Software Reengineering Toolkit (Implements large-scale refactoring for C, C++, C#, COBOL, Java, PHP
- Photran a Fortran plugin for the Eclipse IDE
- SharpSort addin for Visual Studio 2008
- Sigasi Studio - standalone or plugin software for VHDL and System Verilog
- XCode
- Smalltalk Refactoring Browser (for Smalltalk)
- Simplifide (for Verilog, VHDL and SystemVerilog)
- Tidier (for Erlang)

References

1. Scott Ambler
2. Kerievsky, Joshua (2004). *Refactoring to Patterns*. Addison Wesley.
3. Fowler, Martin (1999). *Refactoring: Improving the design of existing code*. Addison Wesley.
4. Martin, Robert (2009). *Clean Code*. Prentice Hall.
5. Refactoring techniques in Fowler's refactoring Website (http://www.refactoring.com/catalog/index.html)
6. Replace type-checking code with State/Strategy (http://www.refactoring.com/catalog/replaceTypeCodeWithSta
7. Replace conditional with polymorphism (http://www.refactoring.com/catalog/replaceConditionalWithPolymorpl
8. Hardware description languages and programming languages
9. Kaiping Zeng, Sorin A. Huss, "Architecture refinements by code refactoring of behavioral VHDL-AMS models". I
10. M. Keating :"Complexity, Abstraction, and the Challenges of Designing Complex Systems", in DAC'08 tutorial [

a Verification Gap: C++ to RTL for Practical Design"

1. M. Keating, P. Bricaud: *Reuse Methodology Manual for System-on-a-Chip Designs*, Kluwer Academic Publishers
2. http://newsgroups.derkeiler.com/Archive/Comp/comp.lang.vhdl/2008-06/msg00173.html
3. www.eetimes.com/news/latest/showArticle.jhtml?articleID=222001855 (http://www.eetimes.com/news/latest/sh
4. Opdyke, William F (June 1992) (compressed Postscript). *Refactoring Object-Oriented Frameworks*. Ph.D. thesis ftp://st.cs.uiuc.edu/pub/papers/refactoring/opdyke-thesis.ps.Z. Retrieved 2008-02-12.
5. Martin Fowler, "MF Bliki: EtymologyOfRefactoring" (http://martinfowler.com/bliki/EtymologyOfRefactoring.ht
6. Opdyke, William F.; Johnson, Ralph E. (September 1990). "Refactoring: An Aid in Designing Application Frame *Symposium on Object Oriented Programming Emphasizing Practical Applications (SOOPPA)*. ACM.

Further reading

- Fowler, Martin (1999). *Refactoring. Improving the Design of Existing Code*. Addison-Wesley. ISBN 0-201-48567-
- Wake, William C. (2003). *Refactoring Workbook*. Addison-Wesley. ISBN 0-321-10929-5.
- Mens, Tom and Tourwé, Tom (2004) *A Survey of Software Refactoring* (http://doi.ieeecomputersociety.org/10.1 February 2004 (vol. 30 no. 2), pp. 126-139
- Feathers, Michael C (2004). *Working Effectively with Legacy Code*. Prentice Hall. ISBN 0-13-117705-2.
- Kerievsky, Joshua (2004). *Refactoring To Patterns*. Addison-Wesley. ISBN 0-321-21335-1.
- Arsenovski, Danijel (2008). *Professional Refactoring in Visual Basic*. Wrox. ISBN 0-47-017979-1.
- Arsenovski, Danijel (2009). *Professional Refactoring in C# and ASP.NET*. Wrox. ISBN 978-0470434529.
- Ritchie, Peter (2010). *Refactoring with Visual Studio 2010*. Packt. ISBN 978-1849680103.

External links

- What Is Refactoring? (http://c2.com/cgi/wiki?WhatIsRefactoring) (c2.com article)
- Martin Fowler's homepage about refactoring (http://www.refactoring.com/)
- Aspect-Oriented Refactoring (http://www.theserverside.com/articles/article.tss?l=AspectOrientedRefactoringPa
- A Survey of Software Refactoring (http://csdl.computer.org/comp/trans/ts/2004/02/e2toc.htm) by Tom Mens a
- Refactoring (http://www.dmoz.org/Computers/Programming/Methodologies/Refactoring/) at DMOZ
- Refactoring Java Code (http://www.methodsandtools.com/archive/archive.php?id=4)
- Refactoring To Patterns Catalog (http://industriallogic.com/xp/refactoring/catalog.html)
- Extract Boolean Variable from Conditional (http://www.industriallogic.com/papers/extractboolean.html) (a refa
- Test-Driven Development With Refactoring (http://www.testingtv.com/2009/09/24/test-driven-development-wi
- Revisiting Fowler's Video Store: Refactoring Code, Refining Abstractions (http://blog.symprise.net/2009/04/rev

Software Quality

Introduction

In the context of software engineering, **software quality** measures how well software is designed (*quality of d conformance*),[1] although there are several different definitions. It is often described as the 'fitness for purpose' of a

Whereas *quality of conformance* is concerned with implementation (see Software Quality Assurance), *quality of de* worthwhile product.[2]

Definition

One of the challenges of software quality is that "everyone feels they understand it".[3]

Software quality may be defined as conformance to explicitly stated functional and performance requirements, ex that are expected of all professionally developed software.

The three key points in this definition:

1. Software requirements are the foundations from which quality is measured.

   > Lack of conformance to requirement is lack of quality.

2. Specified standards define a set of development criteria that guide the manager is software engineering.

> If criteria are not followed lack of quality will usually result.

3. A set of implicit requirements often goes unmentioned, for example ease of use, maintainability etc.

> If software confirms to its explicit requirement but fails to meet implicit requirements, software quality is s

A definition in Steve McConnell's *Code Complete* divides software into two pieces: **internal** and **external quali** a product that face its users, where internal quality characteristics are those that do not.[4]

Another definition by Dr. Tom DeMarco says "a product's quality is a function of how much it changes the w satisfaction is more important than anything in determining software quality.[1]

Another definition, coined by Gerald Weinberg in *Quality Software Management: Systems Thinking*, is "Quality is subjective - different people will experience the quality of the same software very differently. One strength of this "Who are the people we want to value our software?" and "What will be valuable to *them*?"

History

Software product quality

- Correctness
- Product quality
  - conformance to requirements or program specification; related to Reliability
- Scalability
- Completeness
- Absence of bugs
- Fault-tolerance
  - Extensibility
  - Maintainability
- Documentation

The Consortium for IT Software Quality (CISQ) was launched in 2009 to standardize the measurement of softwa executives from Global 2000 IT organizations, system integrators, outsourcers, and package vendors to jointly a quality and to promote a market-based ecosystem to support its deployment.

Source code quality

A computer has no concept of "well-written" source code. However, from a human point of view source code can be its behavior. Many source code programming style guides, which often stress readability and usually languag maintenance. Some of the issues that affect code quality include:

- Readability
- Ease of maintenance, testing, debugging, fixing, modification and portability
- Low complexity
- Low resource consumption: memory, CPU
- Number of compilation or lint warnings
- Robust input validation and error handling, established by software fault injection

Methods to improve the quality:

- Refactoring
- Code Inspection or software review
- Documenting code

Software reliability

Software **reliability** is an important facet of software quality. It is defined as "the probability of failure-free ope time".[6]

One of reliability's distinguishing characteristics is that it is objective, measurable, and can be estimated, where especially important in the discipline of Software Quality Assurance. These measured criteria are typically called sof

History

With software embedded into many devices today, software failure has caused more than inconvenience. Software poorly designed user interfaces to direct programming errors. An example of a programming error that lead to multi .edu/papers/therac.pdf) (PDF). This has resulted in requirements for development of some types software. In th Federal Aviation Administration (FAA) have requirements for software development.

Goal of reliability

The need for a means to objectively determine software reliability comes from the desire to apply the techniques o desire is a result of the common observation, by both lay-persons and specialists, that computer software does no undesirable behaviour, up to and including outright failure, with consequences for the data which is processed, the materials which those machines might negatively affect. The more critical the application of the software to eco important is the need to assess the software's reliability.

Regardless of the criticality of any single software application, it is also more and more frequently observed that through the technology we use. It is only expected that this infiltration will continue, along with an accompanying As software becomes more and more crucial to the operation of the systems on which we depend, the argument g dependability. In other words, the software should behave in the way it is intended, or even better, in the way it sh

Challenge of reliability

The circular logic of the preceding sentence is not accidental—it is meant to illustrate a fundamental problem i determining, in advance, exactly how the software is intended to operate. The problem seems to stem from a co software in some sense takes on a role which would otherwise be filled by a human being. This is a problem on tw could never perform, especially at the high level of reliability that is often expected from software in comparison t mental capabilities of humans which separate them from mere mechanisms: qualities such as adaptability, general common sense.

Nevertheless, most software programs could safely be considered to have a particular, even singular purpose. If completely defined, it should present a means for at least considering objectively whether the software is, in fact, running the software in a given environment, with given data. Unfortunately, it is still not known whether it is poss outcome of the entire set of possible environment and input data to a given program, without which it is probably i

However, various attempts are in the works to attempt to rein in the vastness of the space of software's enviro descriptions of programs. Such attempts to improve software reliability can be applied at different stages of a progr include: requirements, design, programming, testing, and runtime evaluation. The study of theoretical software re mathematical field of computer science which is an outgrowth of language and automata theory.

Reliability in program development

Requirements

A program cannot be expected to work as desired if the developers of the program do not, in fact, know the progran desired behaviour in parallel with development, in sufficient detail. What level of detail is considered sufficient impractical, if not actually impossible. This is because the desired behaviour tends to change as the possible ra accurately, failed attempts, to achieve it.

Whether a program's desired behaviour can be successfully specified in advance is a moot point if the behaviour can process of creating requirements for new software projects. In situ with the formalization effort is an attempt to hel software projects without sufficient knowledge of what computer software is in fact capable. Communicating this k programmers cannot always know in advance what is actually possible for software in advance of trying.

Design

While requirements are meant to specify what a program should do, design is meant, at least at a high level, to questioned by some, but those who look to formalize the process of ensuring reliability often offer good software de design usually involves the use of more abstract and general means of specifying the parts of the software and wh down into many smaller programs, such that those smaller pieces together do the work of the whole program.

The purposes of high-level design are as follows. It separates what are considered to be problems of architecture, or which solve problems of actual data processing. It applies additional constraints to the development process by nar hoped—removing variables which could increase the likelihood of programming errors. It provides a program tem different teams of developers working on disparate parts, such that they can know in advance how each of their perhaps most controversially, it specifies the program independently of the implementation language or languages, otherwise creep into the design, perhaps unwittingly on the part of programmer-designers.

Programming

The history of computer programming language development can often be best understood in the light of atten becomes more difficult to understand in proportion (perhaps exponentially) to the size of the programs. (Another w way of getting the computer to do more and more of the work, but this may be a different way of saying the sa functionality is a sure way to fail to detect errors in the program, and thus the use of better languages should, conv

Improvements in languages tend to provide incrementally what software design has attempted to do in one fell s inventions as statement, sub-routine, file, class, template, library, component and more have allowed the arrangeme hierarchies and modules, which provide structure at different granularities, so that from any point of view the progr

In addition, improvements in languages have enabled more exact control over the shape and use of data elements, to a very fine degree, including how and when they are accessed, and even the state of the data before and after it is

Software Build and Deployment

Many programming languages such as C and Java require the program "source code" to be translated in to a for program called a compiler. Additional operations may be involved to associate, bind, link or package files toget application. The totality of the compiling and assembly process is generically called "building" the software.

The software build is critical to software quality because if any of the generated files are incorrect the softwar inadvertently used, then testing can lead to false results.

Software builds are typically done in work area unrelated to the runtime area, such as the application server. F
software build products to the runtime area. The deployment procedure may also involve technical parameters, wh
For example, a Java application server may have options for parent-first or parent-last class loading. Using the i
application server.

The technical activities supporting software quality including build, deployment, change control and reporting are
software tools have arisen to help meet the challenges of configuration management including file control tools and h

Testing

Software testing, when done correctly, can increase overall software *quality of conformance* by testing that th
limited to:

1. Unit Testing
2. Functional Testing
3. Regression Testing
4. Performance Testing
5. Failover Testing
6. Usability Testing

A number of agile methodologies use testing early in the development cycle to ensure quality in their products. For
before the code they will test, is used in Extreme Programming to ensure quality.

Runtime

runtime reliability determinations are similar to tests, but go beyond simple confirmation of behaviour to the eval
code or particular hardware configurations.

Software quality factors

A software quality factor is a non-functional requirement for a software program which is not called up by the
enhances the quality of the software program. Note that none of these factors are binary; that is, they are not "eith
one seeks to maximize in one's software to optimize its quality. So rather than asking whether a software product "h

Some software quality factors are listed here:

Understandability

Clarity of purpose. This goes further than just a statement of purpose; all of the design and user documentation
obviously subjective in that the user context must be taken into account: for instance, if the software product is
to the layman.

Completeness

Presence of all constituent parts, with each part fully developed. This means that if the code calls a subroutine f
that library and all required parameters must be passed. All required input data must also be available.

Conciseness

Minimization of excessive or redundant information or processing. This is important where memory capacity is l
to a minimum. It can be improved by replacing repeated functionality by one subroutine or function which achi

Portability

Ability to be run well and easily on multiple computer configurations. Portability can mean both between differ
between different operating systems—such as running on both Mac OS X and GNU/Linux.

Consistency

Uniformity in notation, symbology, appearance, and terminology within itself.

Maintainability

Propensity to facilitate updates to satisfy new requirements. Thus the software product that is maintainable sho
capacity for memory, storage and processor utilization and other resources.

Testability

Disposition to support acceptance criteria and evaluation of performance. Such a characteristic must be built-in
complex design leads to poor testability.

Usability

Convenience and practicality of use. This is affected by such things as the human-computer interface. The comp

(UI), which for best usability is usually graphical (i.e. a GUI).

Reliability

Ability to be expected to perform its intended functions satisfactorily. This implies a time factor in that a reliab
encompasses environmental considerations in that the product is required to perform correctly in whatever cond

Efficiency

Fulfillment of purpose without waste of resources, such as memory, space and processor utilization, network ban

Security

Ability to protect data against unauthorized access and to withstand malicious or inadvertent interference with
such as authentication, access control and encryption, security also implies resilience in the face of malicious, int

Measurement of software quality factors

There are varied perspectives within the field on measurement. There are a great many measures that are valued b
by others. Some believe that quantitative measures of software quality are essential. Others believe that context
qualitative measures. Several leaders in the field of software testing have written about the difficulty of measuring w

One example of a popular metric is the number of faults encountered in the software. Software that contains few
contains many faults. Questions that can help determine the usefulness of this metric in a particular context include

1. What constitutes "many faults?" Does this differ depending upon the purpose of the software (e.g., blogging soft
and complexity of the software?
2. Does this account for the importance of the bugs (and the importance to the stakeholders of the people those bu
or the incidence of users it affects? If so, how? And if not, how does one know that 100 faults discovered is bette
3. If the count of faults being discovered is shrinking, how do I know what that means? For example, does that me
this is a smaller/less ambitious change than before? Or that fewer tester-hours have gone into the project than b
before? Or that the team has discovered that fewer faults reported is in their interest?

This last question points to an especially difficult one to manage. All software quality metrics are in some sense n
discovers that they will benefit from a drop in the number of reported bugs, there is a strong tendency for the tea
circumvent the bug tracking system, or that four or five bugs get lumped into one bug report, or that testers lear
mean to measure, without creating incentives for software programmers and testers to consciously or unconsciously

Software quality factors cannot be measured because of their vague definitions. It is necessary to find measureme
requirements. For example, reliability is a software quality factor, but cannot be evaluated in its own right. Ho
measured. Some such attributes are mean time to failure, rate of failure occurrence, and availability of the system.
statements in a program.

A scheme that could be used for evaluating software quality factors is given below. For every characteristic, there a
of scoring formula could be developed based on the answers to these questions, from which a measurement of the ch

Understandability

Are variable names descriptive of the physical or functional property represented? Do uniquely recognisable fun
deviations from forward logical flow adequately commented? Are all elements of an array functionally related?....

Completeness

Are all necessary components available? Does any process fail for lack of resources or programming? Are all pote
handling?

Conciseness

Is all code reachable? Is any code redundant? How many statements within loops could be placed outside the loop, t

Portability

Does the program depend upon system or library routines unique to a particular installation? Have machine-de
internal bit representation of alphanumeric or special characters been avoided? How much effort would be req
environment to another?

Consistency

Is one variable name used to represent different logical or physical entities in the program? Does the program c
constant? Are functionally similar arithmetic expressions similarly constructed? Is a consistent scheme used fo
elements?

Maintainability

Has some memory capacity been reserved for future expansion? Is the design cohesive—i.e., does each module h
change in data structures (object-oriented designs are more likely to allow for this)? If the code is procedure-based
the main program, or just a module?

Testability

Are complex structures employed in the code? Does the detailed design contain clear pseudo-code? Is the pseudo-c
concurrent designs, are schemes available for providing adequate test cases?

Usability
Is a GUI used? Is there adequate on-line help? Is a user manual provided? Are meaningful error messages provided?

Reliability
Are loop indexes range-tested? Is input data checked for range errors? Is divide-by-zero avoided? Is exception h
intended functions correctly in a specified period of time under stated operation conditions, but there could also be a

Efficiency
Have functions been optimized for speed? Have repeatedly used blocks of code been formed into subroutines? Has th

Security
Does the software protect itself and its data against unauthorized access and use? Does it allow its operator to enfor
correctly implemented? Can the software withstand attacks that can be anticipated in its intended environment?

User's perspective

In addition to the technical qualities of software, the end user's experience also determines the quality of software.
the usability of a given software product. Some important questions to be asked are:

- Is the user interface intuitive (self-explanatory/self-documenting)?
- Is it easy to perform simple operations?
- Is it feasible to perform complex operations?
- Does the software give sensible error messages?
- Do widgets behave as expected?
- Is the software well documented?
- Is the user interface responsive or too slow?

Also, the availability of (free or paid) support may factor into the usability of the software.

References

Notes

1. Pressman 2005, p. 746
2. Pressman 2005, p. 388
3. Crosby, P., *Quality is Free*, McGraw-Hill, 1979
4. McConnell 1993, p. 558
5. DeMarco, T., *Management Can Make Quality (Im)possible*, Cutter IT Summit, Boston, April 1999
6. J.D. Musa, A. Iannino, and K. Okumoto, *Engineering and Managing Software with Reliability Measures*, McGra
7. Pressman 2005, p. 762
8. ISTQB (http://istqbexamcertification.com/what-is-a-software-testing/) - What is software testing?
9. Cem Kaner http://www.kaner.com/pdfs/metrics2004.pdf
0. Douglass Hoffman http://www.softwarequalitymethods.com/Papers/DarkMets%20Paper.pdf

Bibliography

- McConnell, Steve (1993), *Code Complete* (First ed.), Microsoft Press
- Pressman, Scott (2005), *Software Engineering: A Practitioner's Approach* (Sixth, International ed.), McGraw-Hi

Further reading

- International Organization for Standardization. *Software Engineering—Product Quality—Part 1: Quality Model*.
- Diomidis Spinellis. *Code Quality: The Open Source Perspective (http://www.spinellis.gr/codequality)*. Addison W
- Ho-Won Jung, Seung-Gweon Kim, and Chang-Sin Chung. Measuring software product quality: A survey of ISO/
  9). *IEEE Software*, 21(5):10–13, September/October 2004.
- Stephen H. Kan. *Metrics and Models in Software Quality Engineering*. Addison-Wesley, Boston, MA, second edi
- Robert L. Glass. *Building Quality Software*. Prentice Hall, Upper Saddle River, NJ, 1992.
- Roland Petrasch, "The Definition of, Software Quality': A Practical Approach (http://web.archive.org/web/200

)", ISSRE, 1999

External links

- Linux: Fewer Bugs Than Rivals (http://www.wired.com/software/coolapps/news/2004/12/66022) Wired Magaz

Static Analysis

Static program analysis is the analysis of computer software that is performed without actually executing programs is known as dynamic analysis). In most cases the analysis is performed on some version of the source c usually applied to the analysis performed by an automated tool, with human analysis being called program understa

The sophistication of the analysis performed by tools varies from those that only consider the behavior of individua code of a program in their analysis. Uses of the information obtained from the analysis vary from highlightir mathematically prove properties about a given program (e.g., its behavior matches that of its specification).

It can be argued that software metrics and reverse engineering are forms of static analysis.

A growing commercial use of static analysis is in the verification of properties of software used in safety-critical cc medical software is increasing in sophistication and complexity, and the U.S. Food and Drug Administration (FDA the quality of software[1].

Formal methods

Formal methods is the term applied to the analysis of software (and hardware) whose results are obtained purely techniques used include denotational semantics, axiomatic semantics, operational semantics, and abstract interpreta

It has been proven that, barring some hypothesis that the state space of programs is finite, finding all possible run- the final result of a program, is undecidable: there is no mechanical method that can always answer truthfully whet dates from the works of Church, Kurt Gödel and Turing in the 1930s (see the halting problem and Rice's theore attempt to give useful approximate solutions.

Some of the implementation techniques of formal static analysis include:

- Model checking considers systems that have finite state or may be reduced to finite state by abstraction;
- Data-flow analysis is a lattice-based technique for gathering information about the possible set of values;
- Abstract interpretation models the effect that every statement has on the state of an abstract machine (i.e., it 'e statement and declaration). This abstract machine over-approximates the behaviours of the system: the abstract *incompleteness* (not every property true of the original system is true of the abstract system). If properly done, t abstract system can be mapped to a true property of the original system)[2]. The Frama-c framework and Polysp
- Use of assertions in program code as first suggested by Hoare logic. There is tool support for some programming and the Java Modeling Language — JML — using ESC/Java and ESC/Java2, ANSI/ISO C Specification Langu

References

1. FDA (2010-09-08). "Infusion Pump Software Safety Research at FDA". Food and Drug Administration. http://www.fda.gov/MedicalDevices/ProductsandMedicalProcedures/GeneralHospitalDevicesandSupplies/Infusi
2. Jones, Paul (2010-02-09). "A Formal Methods-based verification approach to medical device software analysis". E http://embeddeddsp.embedded.com/design/opensource/222700533. Retrieved 2010-09-09.

Bibliography

- Syllabus and readings (http://www.stanford.edu/class/cs295/) for Alex Aiken (http://theory.stanford.edu/~aike
- Nathaniel Ayewah, David Hovemeyer, J. David Morgenthaler, John Penix, William Pugh, "Using Static Analysi 9/MS.2008.130)," IEEE Software, vol. 25, no. 5, pp. 22-29, Sep./Oct. 2008, doi:10.1109/MS.2008.130
- Brian Chess, Jacob West (Fortify Software) (2007). *Secure Programming with Static Analysis*. Addison-Wesley.
- Adam Kolawa (Parasoft), Static Analysis Best Practices (http://www.parasoft.com/jsp/redirector.jsp/WWH_C
- *Improving Software Security with Precise Static and Runtime Analysis (http://research.microsoft.com/en-us/un* "Static Techniques for Security," Stanford doctoral thesis, 2006.
- Flemming Nielson, Hanne R. Nielson, Chris Hankin (1999, corrected 2004). *Principles of Program Analysis*. Spri
- "Abstract interpretation and static analysis," (http://santos.cis.ksu.edu/schmidt/Escuela03/home.html) Interna Schmidt (http://people.cis.ksu.edu/~schmidt/)

External links

- The SAMATE Project (http://samate.nist.gov), a resource for Automated Static Analysis tools

- Integrate static analysis into a software development process (http://www.embedded.com/shared/printableArtic
- Code Quality Improvement - Coding standards conformance checking (DDJ) (http://www.ddj.com/dept/debug/
- Episode 59: Static Code Analysis (http://www.se-radio.net/index.php?post_id=220531) Interview (Podcast) at
- Implementing Automated Governance for Coding Standards (http://www.infoq.com/articles/governance-coding-
  the build process

Metrics

A **software metric** is a measure of some property of a piece of software or its specifications. Since quantitative n
computer science practitioners and theoreticians to bring similar approaches to software development. The goal is
may have numerous valuable applications in schedule and budget planning, cost estimation, quality assurance testi
personnel task assignments.

Common software measurements

Common software measurements include:

- Balanced scorecard
- Bugs per line of code
- COCOMO
- Code coverage
- Cohesion
- Comment density[1]
- Connascent software components
- Coupling
- Cyclomatic complexity
- Function point analysis
- Halstead Complexity
- Instruction path length
- Number of classes and interfaces
- Number of lines of code
- Number of lines of customer requirements
- Program execution time
- Program load time
- Binary file|Program size (binary)
- Robert Cecil Martin's software package metrics
- Weighted Micro Function Points

Limitations

As software development is a complex process, with high variance on both methodologies and objectives, it is
determine a valid and concurrent measurement metric, especially when making such a prediction prior to the detail
metrics matter, and what they mean.[2][3] The practical utility of *software* measurements has thus been limited to n

- Schedule
- Size/Complexity
- Cost
- Quality

Common goal of measurement may target one or more of the above aspects, or the balance between them as indicat

Acceptance and Public Opinion

Some software development practitioners point out that simplistic measurements can cause more harm than good.
software development process.[2] Impact of measurement on programmers psychology have raised concerns for h
attempts to cheat the metrics, while others find it to have positive impact on developers value towards their ow
definition of many measurement methodologies are imprecise, and consequently it is often unclear how tools for c
imperfect quantification is better than none ("You can't control what you can't measure.")[7]. Evidence shows that
military, NASA[8], IT consultants, academic institutions[9], and commercial and academic development estimation s

References

1. "Descriptive Information (DI) Metric Thresholds". *Land Software Engineering Centre*. http://www.lsec.dnd.ca/q October 2010.
2. Binstock, Andrew. "Integration Watch: Using metrics effectively". *SD Times*. BZ Media. http://www.sdtimes.co
3. Kolawa, Adam. "When, Why, and How: Code Analysis". *The Code Project*. http://www.codeproject.com/KB/in
4. Kaner, Dr. Cem, *Software Engineer Metrics: What do they measure and how do we know?*, http://citeseerx.ist.p
5. ProjectCodeMeter (2010) "ProjectCodeMeter Users Manual" page 65 [5] (http://www.projectcodemeter.com/cost
6. Lincke, Rüdiger; Lundberg, Jonas; Löwe, Welf (2008), "Comparing software metrics tools", *International Sympos* http://www.arisa.se/files/LLL-08.pdf
7. DeMarco, Tom. *Controlling Software Projects: Management, Measurement and Estimation*. ISBN 0-13-171711-1
8. NASA Metrics Planning and Reporting Working Group (MPARWG) [6] (https://esdswg.eosdis.nasa.gov/wg/mp
9. USC Center for Systems and Software Engineering [7] (http://sunset.usc.edu/csse/research/COCOMOII/cocomo

External links

- "Minimal Essential Software Quality Metrics" (http://archive.is/20121216051506/qualinfra.blogspot.com/2010/0 of essential metrics for a successful product delivery.
- Definitions of software metrics in .NET (http://www.ndepend.com/Metrics.aspx)
- International Function Point Users Group (http://www.ifpug.org)
- What is FPA (http://www.nesma.org/section/fpa/) at Nesma website
- Estimating With Use Case Points (http://www.methodsandtools.com/archive/archive.php?id=25) by Mike Coh with UML, using use cases.
- OO & Agile Metrics Resources (http://www.parlezuml.com/metrics/index.htm) - includes workshop material on
- Further defines the term Software Metrics with examples. (http://www.sqa.net/softwarequalitymetrics.html)
- Software Engineering Metrics: What do they measure and how do we know (http://www.kaner.com/pdfs/metric metrics

Software Package Metrics

This article describes various **software package metrics**. They have been mentioned by Robert Cecil Martin book (2002).

The term *software package*, as it is used here, refers to a group of related classes (in the field of object-oriented prog

- **Number of Classes and Interfaces:** The number of concrete and abstract classes (and interfaces) in the p
- **Afferent Couplings (Ca):** The number of other packages that depend upon classes within the package is a
- **Efferent Couplings (Ce):** The number of other packages that the classes in the package depend upon is an
- **Abstractness (A):** The ratio of the number of abstract classes (and interfaces) in the analyzed package to th metric is 0 to 1, with A=0 indicating a completely concrete package and A=1 indicating a completely abstract p
- **Instability (I):** The ratio of efferent coupling (Ce) to total coupling (Ce + Ca) such that I = Ce / (Ce + Ca) range for this metric is 0 to 1, with I=0 indicating a completely stable package and I=1 indicating a completely
- **Distance from the Main Sequence (D):** The perpendicular distance of a package from the idealized line between abstractness and stability. A package squarely on the main sequence is optimally balanced with respect abstract and stable (x=0, y=1) or completely concrete and instable (x=1, y=0). The range for this metric is 0 t sequence and D=1 indicating a package that is as far from the main sequence as possible.
- **Package Dependency Cycles:** Package dependency cycles are reported along with the hierarchical paths o

References

- Robert Cecil Martin (2002). *Agile Software Development: Principles, Patterns and Practices*. Pearson Education

External links

- OO Metrics (http://www.parlezuml.com/metrics/OO%20Design%20Principles%20&%20Metrics.pdf) tutorial ex
- JHawk (http://www.virtualmachinery.com/jhawkprod.htm) - Java Metrics tool, All the most important code m
- Lattix (http://www.lattix.com/products) - Architecture tool that supports a variety of architecture metrics incl

- NDepend (http://www.ndepend.com/) - .NET application that supports the package dependency metrics.
- CppDepend (http://www.cppdepend.com/) - C++ Metrics tool that supports all the most important code metri
- JDepend (http://clarkware.com/software/JDepend.html) - Java application that supports the package depender
- STAN (http://www.stan4j.com/) - Structure Analysis for Java. Eclipse integrated and standalone visual depend
- SourceMonitor (http://www.campwoodsw.com/sourcemonitor.html) - Something for C++, C, C#, VB.NET, Ja
- PHP Depend (http://pdepend.org/) - PHP version of JDepend that supports the package dependency metrics.

Visualization

Software visualization[1] is the static or animated 2-D or 3-D[2] visual representation of information abo behavior.[6]

Typically, the information used for visualization is software metric data from measurement activities or from rev quality assurance but can be used to manually discover anomalies similar to the process of visual data mining.[7]

The objectives of software visualizations are to support the understanding of software systems (i.e., its structure) a well as the analysis of software systems and their anomalies (e.g., by showing classes with high coupling).

Types

Single component

Tool for software visualization might be used to visualize source code and quality defects during software developm and visualization of quality defects in object-oriented software systems and services. Designed as a plugin for an ID a class and its methods with other classes in the software system and mark potential quality defects to warn the d the software system.

Whole (sub-)systems

Other more powerful tools are used to visualize a whole system or subsystem to explore the architecture or to apply

References

1. (Diehl, 2002; Diehl, 2007; Knight, 2002)
2. (Marcus et al., 2003; Wettel et al., 2007)
3. (Staples & Bieman, 1999)
4. (Lanza, 2004)
5. (Girba et al., 2005, Lopez et al., 2004; Van Rysselberghe et al., 2004)
6. (Kuhn et al., 2006, Stasko et al., 1997)
7. (Keim, 2002; Soukup, 2002).

Further reading

- Diehl, S. (2002). *Software Visualization*. International Seminar. Revised Papers (LNCS Vol. 2269), Dagstuhl Cas
- Diehl, S. (2007). *Software Visualization — Visualizing the Structure, Behaviour, and Evolution of Software*. Spri
- Gîrba, T., Kuhn, A., Seeberger, M., and Ducasse, S., "How Developers Drive Software Evolution," Proceedings c 2005), IEEE Computer Society Press, 2005, pp. 113–122. PDF (http://www.iam.unibe.ch/~scg/Archive/Papers/
- Keim, D. A. (2002). *Information visualization and visual data mining*. IEEE Transactions on Visualization and C refs.
- Knight, C. (2002). *System and Software Visualization*. In *Handbook of software engineering & knowledge engine* Publishing Company.
- Kuhn, A., and Greevy, O., "Exploiting the Analogy Between Traces and Signal Processing," Proceedings IEEE I Computer Society Press, Los Alamitos CA, September 2006. PDF (http://www.iam.unibe.ch/~scg/Archive/Pap
- Lanza, M. (2004). *CodeCrawler — polymetric views in action*. Proceedings. 19th International Conference on Au Alamitos, CA, USA: IEEE Comput. Soc, 2004, p 394 5.
- Lopez, F. L., Robles, G., & Gonzalez, B. J. M. (2004). *Applying social network analysis to the information in C* Repositories (MSR 2004)" W17S Workshop 26th International Conference on Software Engineering, Edinburgh,
- Marcus, A., Feng, L., & Maletic, J. I. (2003). *3D representations for software visualization*. Paper presented at t San Diego, California.
- Soukup, T. (2002). *Visual data mining : techniques and tools for data visualization and mining*. New York: Chic
- Staples, M. L., & Bieman, J. M. (1999). *3-D Visualization of Software Structure*. In *Advances in Computers* (Vo

- Stasko, J. T., Brown, M. H., & Price, B. A. (1997). *Software Visualization*: MIT Press.
- Van Rysselberghe, F. (2004). *Studying Software Evolution Information By Visualizing the Change History*. Proce 328–337, IEEE Computer Society Press, 2004
- Wettel, R., and Lanza, M., *Visualizing Software Systems as Cities*. In Proceedings of VISSOFT 2007 (4th IEEE and Analysis), pp. 92 – 99, IEEE Computer Society Press, 2007.

External links

- EPDV (http://code.google.com/p/epdv/) Eclipse Project Dependencies Viewer
- SoftVis (http://www.softvis.org) is the second meeting in a planned series of biennial conferences.
- The Program Visualization Workshops (http://www.algoanim.net/pvw2006/) aim to bring together researchers visualizations or animations as well as educators who use or evaluate visualization or animations in their teachin
- CppDepend (http://www.cppdepend.com/) - useful C++ tool to visualize dependencies.

Code Review

Code review is systematic examination (often as peer review) of computer source code. It is intended to find a both the overall quality of software and the developers' skills. Reviews are done in various forms such as pair progra

Introduction

Code reviews can often find and remove common vulnerabilities such as format string exploits, race conditions, m Online software repositories based on Subversion (with Redmine or Trac), Mercurial, Git or others allow groups of i collaborative code review can facilitate the code review process.

Automated code reviewing software lessens the task of reviewing large chunks of code on the developer by systemati

Capers Jones' ongoing analysis of over 12,000 software development projects showed that the latent defect disc inspection, the figure is less than 50%.[*citation needed*] The latent defect discovery rate for most forms of testing is a

Typical code review rates are about 150 lines of code per hour. Inspecting and reviewing more than a few hund embedded software) may be too fast to find errors. [3] Industry data indicate that code review can accomplish at mo

Types

Code review practices fall into three main categories: pair programming, formal code review and lightweight code re

Formal code review, such as a Fagan inspection, involves a careful and detailed process with multiple participants review, in which software developers attend a series of meetings and review code line by line, usually using printed have been proven effective at finding defects in the code under review.

Lightweight code review typically requires less overhead than formal code inspections, though it can be equally effe conducted as part of the normal development process:

- Over-the-shoulder – One developer looks over the author's shoulder as the latter walks through the code.
- Email pass-around – Source code management system emails code to reviewers automatically after checkin is ma
- Pair Programming – Two authors develop code together at the same workstation, such is common in Extreme P
- Tool-assisted code review – Authors and reviewers use specialized tools designed for peer code review.

Some of these may also be labeled a "Walkthrough" (informal) or "Critique" (fast and informal).

Many teams that eschew traditional, formal code review use one of the above forms of lightweight review as part o in the book *Best Kept Secrets of Peer Code Review* found that lightweight reviews uncovered as many bugs as form

Criticism

Historically, formal code reviews have required a considerable investment in preparation for the review event and ex

Some believe that skillful, disciplined use of a number of other development practices can result in similarly high argue, layering additional XP practices, such as refactoring and test-driven development will result in latent de without the investment.[*citation needed*]

Use of code analysis tools can support this activity. Especially tools that work in the IDE as they provide direct feed

References

1. Kolawa, Adam; Huizinga, Dorota (2007). *Automated Defect Prevention: Best Practices in Software Managemen* http://www.wiley.com/WileyCDA/WileyTitle/productCd-0470042125.html.
2. Jones, Capers; Christof, Ebert (April 2009). "Embedded Software: Facts, Figures, and Future". IEEE Computer Retrieved 2010-10-05.

3. Ganssle, Jack (February 2010). "A Guide to Code Inspections". The Ganssle Group. http://www.ganssle.com/ins
4. Jones, Capers (June 2008). "Measuring Defect Potentials and Defect Removal Efficiency". Crosstalk, The Journa http://www.stsc.hill.af.mil/crosstalk/2008/06/0806jones.html. Retrieved 2010-10-05.

- Jason Cohen (2006). *Best Kept Secrets of Peer Code Review (Modern Approach. Practical Advice.)*. Smartbears

External links

- *Security Code Review FAQs* (http://www.ouncelabs.com/resources/code-review-faq.asp)
- Security code review guidelines (http://www.homeport.org/~adam/review.html)
- Lightweight Tool Support for Effective Code Reviews (http://web.archive.org/web/20080720093900/http://www white paper
- Code Review Best Practices (http://www.parasoft.com/jsp/printables/When_Why_How_Code_Review.pdf?pa
- Best Practices for Peer Code Review (http://web.archive.org/web/20070929033247/http://smartbear.com/docs/
- Code review case study (http://web.archive.org/web/20070328001806/http://smartbearsoftware.com/docs/book
- "A Guide to Code Inspections" (Jack G. Ganssle) (http://www.ganssle.com/inspections.pdf)
- Article Four Ways to a Practical Code Review (http://www.methodsandtools.com/archive/archive.php?id=66)

Code Inspection

Inspection in software engineering, refers to peer review of any work product by trained individuals who look referred to as a Fagan inspection after Michael Fagan, the creator of a very popular software inspection process.

Introduction

An inspection is one of the most common sorts of review practices found in software projects. The goal of the insp and approve it for use in the project. Commonly inspected work products include software requirements specificatio and a team is gathered for an inspection meeting to review the work product. A moderator is chosen to moderate th product and noting each defect. **The goal of the inspection is to identify defects**. In an inspection, a d approving it. For example, if the team is inspecting a software requirements specification, each defect will be text in

The process

The inspection process was developed by Michael Fagan in the mid-1970s and it has later been extended and modifi

The process should have entry criteria that determine if the inspection process is ready to begin. This prevents un criteria might be a checklist including items such as "The document has been spell-checked".

The stages in the inspections process are: Planning, Overview meeting, Preparation, Inspection meeting, Rework a might be iterated.

- **Planning:** The inspection is planned by the moderator.
- **Overview meeting:** The author describes the background of the work product.
- **Preparation:** Each inspector examines the work product to identify possible defects.
- **Inspection meeting:** During this meeting the reader reads through the work product, part by part and the in
- **Rework:** The author makes changes to the work product according to the action plans from the inspection mee
- **Follow-up:** The changes by the author are checked to make sure everything is correct.

The process is ended by the moderator when it satisfies some predefined exit criteria.

Inspection roles

During an inspection the following roles are used.

- **Author:** The person who created the work product being inspected.
- **Moderator:** This is the leader of the inspection. The moderator plans the inspection and coordinates it.
- **Reader:** The person reading through the documents, one item at a time. The other inspectors then point out d
- **Recorder/Scribe:** The person that documents the defects that are found during the inspection.
- **Inspector:** The person that examines the work product to identify possible defects.

Related inspection types

Code review

A code review can be done as a special kind of inspection in which the team examines a sample of code and fixes a not properly implement its requirements, which does not function as the programmer intended, or which is not readable or its performance could be improved). In addition to helping teams find and fix bugs, code reviews are use

for helping junior developers learn new programming techniques.

Peer Reviews

Peer Reviews are considered an industry best-practice for detecting software defects early and learning about softw
software inspections and are integral to software product engineering activities. A collection of coordinated knowl
Reviews. The elements of Peer Reviews include the structured review process, standard of excellence product checkli

Software inspections are the most rigorous form of Peer Reviews and fully utilize these elements in detecting defects
the producer to obtain the deepest understanding of an artifact and reaching a consensus among participants. Mea
investment obtained through accelerated learning and early defect detection. For best results, Peer Reviews are roll
policy and procedure, training practitioners and managers, defining measurements and populating a database structu

External links

- Review and inspection practices (http://www.stellman-greene.com/reviews)
- Article Software Inspections (http://www.methodsandtools.com/archive/archive.php?id=29) by Ron Radice
- Comparison of different inspection and review techniques (http://www.the-software-experts.de/e_dta-sw-test-ins

Deployment & Maintenance

Introduction

Software deployment is all of the activities that make a software system available for use.

The general deployment process consists of several interrelated activities with possible transitions between them. T
both. Because every software system is unique, the precise processes or procedures within each activity can hardly
process that has to be customized according to specific requirements or characteristics. A brief description of each a

Deployment activities

Release

The release activity follows from the completed development process. It includes all the operations to prepare a
must determine the resources required to operate at the customer site and collect information for carrying out su

Install and activate

Activation is the activity of starting up the executable component of software. For simple system, it involves est
should make all the supporting systems ready to use.

In larger software deployments, the working copy of the software might be installed on a production server in a
be installed in a test environment, development environment and disaster recovery environment.

Deactivate

Deactivation is the inverse of activation, and refers to shutting down any executing components of a system. De
a software system may need to be deactivated before an update can be performed. The practice of removing infr
application retirement or application decommissioning.

Adapt

The adaptation activity is also a process to modify a software system that has been previously installed. It differ
as changing the environment of customer site, while updating is mostly started from remote software producer.

Update

The update process replaces an earlier version of all or part of a software system with a newer release.

Built-In

Mechanisms for installing updates are built into some software systems. Automation of these update processes ra
Internet Security is an example of a system with a semi-automatic method for retrieving and installing updates f
Other software products provide query mechanisms for determining when updates are available.

Version tracking

Version tracking systems help the user find and install updates to software systems installed on PCs and local n

- Web based version tracking systems notify the user when updates are available for software systems installe
versions on a user's computer and then queries its database to see if any updates are available.
- Local version tracking system notifies the user when updates are available for software systems installed on a
information for each software package installed on a local system. One click of a button launches a browser w
filling of the user name and password for sites that require a login.
- Browser based version tracking systems notify the user when updates are available for software packages inst
extension which helps the user find the current version number of any program listed on the web.

Uninstall

Uninstallation is the inverse of installation. It is the removal of a system that is no longer required. It also involv the uninstalled system's files and dependencies.

Retire

Ultimately, a software system is marked as obsolete and support by the producers is withdrawn. It is the end of

Deployment roles

The complexity and variability of software products has necessitated the creation of specialized roles for coordinatin user is frequently also the "software deployer" when they install the software package on their machine. For enterpr involved typically change as the application progresses from test (pre-production) to production environment applications are:

- Pre-production environments
  - Application developers: see Software development process
  - Build and release engineers: see Release engineering
  - Release managers: see Release management
  - Deployment coordinators: see DevOps
- Production environments
  - System administrator
  - Database administrator
  - Release coordinators: see DevOps
  - Operations project managers: see Information Technology Infrastructure Library

Examples

- FAI OpenSource Software Linux
- M23 OpenSource Software Linux
- Open PC Server Integration (opsi) OpenSource Software Windows
- RPM with YUM OpenSource Software Linux
- MS SCCM Microsoft Windows
- HP OpenView (Hewlett-Packard)
- Tivoli Provisioning Manager and IBM Tivoli Intelligent Orchestrator
- DX-Union (Materna)
- Novell ZENworks (Novell) Zero Effort Networks
- Garibaldi (Software) (INOSOFT AG)
- Client Management Suite (Baramundi Software AG, Augsburg)
- Blackberry MDS Suite Research In Motion (RIM)
- Intellisync Mobile Suite Nokia
- Mobile Device Manager 2008 Microsoft
- ubi-Suite ubitexx
- Java Web Start

References

External links

- Standardization efforts
  - Solution Installation Schema Submission request to W3C (http://www.w3.org/Submission/2004/04/)
  - OASIS Solution Deployment Descriptor TC (http://www.oasis-open.org/committees/sdd/charter.php)
  - OMG Specification for Deployment and Configuration of Component-based Distributed Applications (http:/
  - JSR 88: Java EE Application Deployment (http://jcp.org/en/jsr/detail?id=88)
- Articles
  - The Future of Software Delivery (https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?lang=en

a=dtl-2108wp5) - free developerWorks whitepaper

- Carzaniga A., Fuggetta A., Hall R. S., Van Der Hoek A., Heimbigner D., Wolf A. L. — A Characterization F CU-CS-857-98, Dept. of Computer Science, University of Colorado, April 1998. http://serl.cs.colorado.edu/~

- Resources
  - Microsoft's resource page on Client Deployment (http://technet.microsoft.com/en-us/windows/default.aspx)

Maintenance

Software maintenance in software engineering is the modification of a software product after delivery to correc

A common perception of maintenance is that it is merely fixing bugs. However, studies and surveys over the years used for non-corrective actions (Pigosky 1997). This perception is perpetuated by users submitting problem reports t

Software maintenance and evolution of systems was first addressed by Meir M. Lehman in 1969. Over a period Evolution (Lehman 1997). Key findings of his research include that maintenance is really evolutionary developm happens to systems (and software) over time. Lehman demonstrated that systems continue to evolve over time. As refactoring is taken to reduce the complexity.

The key software maintenance issues are both managerial and technical. Key management issues are: alignment v estimating costs. Key technical issues are: limited understanding, impact analysis, testing, maintainability measurem

Software maintenance processes

This section describes the six software maintenance processes as:

1. The implementation processes contains software preparation and transition activities, such as the conception and problems identified during development, and the follow-up on product configuration management.
2. The problem and modification analysis process, which is executed once the application has become the responsib analyze each request, confirm it (by reproducing the situation) and check its validity, investigate it and propose finally, obtain all the required authorizations to apply the modifications.
3. The process considering the implementation of the modification itself.
4. The process acceptance of the modification, by confirming the modified work with the individual who submitted
5. The migration process (platform migration, for example) is exceptional, and is not part of daily maintenance tas change in functionality, this process will be used and a maintenance project team is likely to be assigned to this
6. Finally, the last maintenance process, also an event which does not occur on a daily basis, is the retirement of a

There are a number of processes, activities and practices that are unique to maintainers, for example:

- Transition: a controlled and coordinated sequence of activities during which a system is transferred progressively
- Service Level Agreements (SLAs) and specialized (domain-specific) maintenance contracts negotiated by maintai
- Modification Request and Problem Report Help Desk: a problem-handling process used by maintainers to priorit
- Modification Request acceptance/rejection: modification request work over a certain size/effort/complexity may

Categories of maintenance in ISO/IEC 14764

E.B. Swanson (http://www.anderson.ucla.edu/x1960.xml) initially identified three categories of maintenance: corre ISO/IEC 14764 presents:

- Corrective maintenance: Reactive modification of a software product performed after delivery to correct discover
- Adaptive maintenance: Modification of a software product performed after delivery to keep a software product u
- Perfective maintenance: Modification of a software product after delivery to improve performance or maintainab
- Preventive maintenance: Modification of a software product after delivery to detect and correct latent faults in t

There is also a notion of pre-delivery/pre-release maintenance which is all the good things you do to lower the tota standards that includes software maintainability goals. The management of coupling and cohesion of the software. and JA1006 for example). Note also that some academic institutions are carrying out research to quantify the cos design documents and system/software comprehension training and resources (multiply costs by approx. 1.5-2.0 whe

References

1. ISO/IEC 14764:2006 Software Engineering — Software Life Cycle Processes — Maintenance (http://www.iso.or 64)
2. E. Burt Swanson, The dimensions of maintenance. Proceedings of the 2nd international conference on Software e citation.cfm?id=359522)

Further reading

- April, Alain; Abran, Alain (2008). *Software Maintenance Management*. New York: Wiley-IEEE. ISBN 978-0470-
- Gopalaswamy Ramesh; Ramesh Bhattiprolu (2006). *Software maintenance : effective practices for geographically* ISBN 9780070483453.
- Grubb, Penny; Takang, Armstrong (2003). *Software Maintenance*. New Jersey: World Scientific Publishing. ISB
- Lehman, M.M.; Belady, L.A. (1985). *Program evolution : processes of software change*. London: Academic Press
- Page-Jones, Meilir (1980). *The Practical Guide to Structured Systems Design*. New York: Yourdon Press. ISBN

External links

- [12] (http://www.software-continuity.com) Software Continuity : a rating methodology for software sustainabilit
- Journal of Software Maintenance (http://www3.interscience.wiley.com/cgi-bin/jhome/5391/)
- Software Maintenance Maturity Model (http://www.s3m.ca)

Evolution

Software evolution is the term used in software engineering (specifically software maintenance) to refer to the various reasons.

General introduction

Fred Brooks, in his key book *The Mythical Man-Month*,[1] states that over 90% of the costs of a typical system ar will inevitably be maintained.

In fact, Agile methods stem from maintenance like activities in and around web based technologies, where the bulk

Software maintenance address bug fixes and minor enhancements and software evolution focus on adaptation and m

Types of software maintenance

E.B. Swanson initially identified three categories of maintenance: corrective, adaptive, and perfective. Four categor These have since been updated and normalized internationally in the ISO/IEC 14764:2006:[3]

- *Corrective maintenance*: Reactive modification of a software product performed after delivery to correct discover
- *Adaptive maintenance*: Modification of a software product performed after delivery to keep a software product u
- *Perfective maintenance*: Modification of a software product after delivery to improve performance or maintainab
- *Preventive maintenance*: Modification of a software product after delivery to detect and correct latent faults in t

All of the preceding take place when there is a known requirement for change.

Although these categories were supplemented by many authors like Warren et al. (1999)[*citation needed*] and standard has kept the basic four categories.

More recently the description of software maintenance and evolution has been done using ontologies (Kitchemham 2003,[*citation needed*] Dias (2003),[*citation needed*] and Ruiz (2004)),[*citation needed*] which enrich the description o

Lehman's Laws of Software Evolution

Prof. Meir M. Lehman, who worked at Imperial College London from 1972 to 2002, and his colleagues have identi behaviours (or observations) are known as *Lehman's Laws*, and there are eight of them:

1. Continuing Change
2. Increasing Complexity
3. Large Program Evolution
4. Invariant Work-Rate
5. Conservation of Familiarity
6. Continuing Growth
7. Declining Quality
8. Feedback System

It is worth mentioning that the laws are believed to apply mainly to monolithic, proprietary software. For exam software development appear to challenge some of the laws[*citation needed*].

The laws predict that change is inevitable and not a consequence of bad programming and that there are lim implementing changes and new functionality.

Maturity Models specific to software evolution have been developed to help improve processes to ensure continuous

The "global process" that is made by the many stakeholders (e.g. developers, users, their managers) has many f structure and other characteristics of the global system. Process simulation techniques, such as system dynamics can

Software evolution is not likely to be Darwinian, Lamarckian or Baldwinian, but an important phenomenon on its o and economy, the successful evolution of software is becoming increasingly critical. This is an important topic of res

The evolution of software, because of its rapid path in comparison to other man-made entities, was seen by Lehman

References

1. Fred Brooks, *The Mythical Man-Month.* Addison-Wesley, 1975 & 1995. ISBN 0-201-00650-2 & ISBN 0-201-8359
2. Lientz, B.P. and Swanson, E.B., *Software Maintenance Management, A Study Of The Maintenance Of Compute* Wesley, Reading MA, 1980. ISBN 0201042053
3. ISO/IEC 14764:2006, 2006.

Project Management

Introduction

Software project management is the art and science of planning and leading software projects[1]. It is a sub-c monitored and controlled.

History

The history of software project management is closely related to the history of software. Software was developed fo oriented programming began to become popular in the 1960's, making *repeatable solutions* possible for the software component-based software engineering. Companies quickly understood the relative ease of use that software program quickly in the 1970's and 1980's. To manage new development efforts, companies applied proven project managem when confusion occurred in the gray zone between the user specifications and the delivered software. To be able to matching user requirements to delivered products, in a method known now as the waterfall model. Since then, following are the most common causes:[2]

1. Unrealistic or unarticulated project goals
2. Inaccurate estimates of needed resources
3. Badly defined system requirements
4. Poor reporting of the project's status
5. Unmanaged risks
6. Poor communication among customers, developers, and users
7. Use of immature technology
8. Inability to handle the project's complexity
9. Sloppy development practices
10. Poor project management
11. Stakeholder politics
12. Commercial pressures

The first three items in the list above show the difficulties articulating the needs of the client in such a way that project management tools are useful and often necessary, but the true art in software project management is app Without a method, tools are worthless. Since the 1960's, several proprietary software project management methods computer consulting firms have also developed similar methods for their clients. Today software project managemer waterfall model to a more cyclic project delivery model that imitates a Software release life cycle.

Software development process

A software development process is concerned primarily with the production aspect of software development, as oppo primarily for supporting the management of software development, and are generally skewed toward addressing bu similar way to general project management processes. Examples are:

- Risk management is the process of measuring or assessing risk and then developing strategies to manage the risk another party, avoiding the risk, reducing the negative effect of the risk, and accepting some or all of the conseq management begins with the business case for starting the project, which includes a cost-benefit analysis as well plan.
- A subset of risk management that is gaining more and more attention is "Opportunity Management", which mea positive, rather than a negative impact. Though theoretically handled in the same way, using the term "opportu team focused on possible positive outcomes of any given risk register in their projects, such as spin-off projects, v

- Requirements management is the process of identifying, eliciting, documenting, analyzing, tracing, prioritizing ar communicating to relevant stakeholders. New or altered computer system[1] Requirements management, which i engineering process; whereby business analysts or software developers identify the needs or requirements of a clie design a solution.
- Change management is the process of identifying, documenting, analyzing, prioritizing and agreeing on changes t communicating to relevant stakeholders. Change impact analysis of new or altered scope, which includes Require engineering process; whereby business analysts or software developers identify the altered needs or requirements position to re-design or modify a solution. Theoretically, each change can impact the timeline and budget of a so analysis before approval.
- Software configuration management is the process of identifying, and documenting the scope itself, which is the s enabling communication of these to relevant stakeholders. In general, the processes employed include version cor agreements.
- Release management is the process of identifying, documenting, prioritizing and agreeing on releases of software relevant stakeholders. Most software projects have access to three software environments to which software can where distributed teams need to integrate their work before release to users, there will often be more environmer before release to User acceptance testing (UAT).
- A subset of release management that is gaining more and more attention is Data Management, as obviously the only in the software environment called "production". In order to test their work, programmers must therefore al versions of a production system were once used for this purpose, but as companies rely more and more on outsid released to development teams. In complex environments, data-sets may be created that are then migrated acros overall software release schedule.

Project planning, monitoring and control

The purpose of project planning is to identify the scope of the project, estimate the work involved, and create a pro software to be developed. The project plan is then developed to describe the tasks that will lead to completion.

The purpose of project monitoring and control is to keep the team and management up to date on the project's p can take action to correct the problem. Project monitoring and control involves status meetings to gather status fr keep the products up to date.

Issue

In computing, the term **issue** is a unit of work to accomplish an improvement in a system. An issue could be a b word "issue" is popularly misused in lieu of "problem." This usage is probably related.[*citation needed*]

For example, OpenOffice.org used to call their modified version of BugZilla IssueZilla. As of September 2010, they c

Problems occur from time to time and fixing them in a timely fashion is essential to achieve correctness of a system

Severity levels

Issues are often categorized in terms of **severity levels**. Different companies have different definitions of severities

- Critical
- High - The bug or issue affects a crucial part of a system, and must be fixed in order for it to resume normal ope
- Medium - The bug or issue affects a minor part of a system, but has some impact on its operation. This severity affected.
- Low - The bug or issue affects a minor part of a system, and has very little impact on its operation. This severit with lower importance) is affected.
- Cosmetic - The system works correctly, but the appearance does not match the expected one. For example: wro sizes, typos, etc. This is the lowest priority issue.

In many software companies, issues are often investigated by Quality Assurance Analysts when they verify a s responsible for resolving them. They can also be assigned by system users during the User Acceptance Testing (UAT

Issues are commonly communicated using Issue or Defect Tracking Systems. In some other cases, emails or instant r

Philosophy

As a subdiscipline of project management, some regard the management of software development akin to the ma management skills, but no programming skills. John C. Reynolds rebuts this view, and argues that software deve program to the managing editor of a newspaper who cannot write.[3]

External links

- Resources on Software Project Management from Steve McConnell: http://www.construx.com/Page.aspx?nid=2
- Resources on Software Project Management from Dan Galorath: http://www.galorath.com/wp/category/project

References

1. Stellman, Andrew; Greene, Jennifer (2005). *Applied Software Project Management*. O'Reilly Media. ISBN 978-0-
2. IEEE (http://spectrum.ieee.org/computing/software/why-software-fails/5) magazine article "Why Software Fail
3. John C. Reynolds, *Some thoughts on teaching programming and programming languages*, SIGPLAN Notices, Vo
   manage software production without the ability to program. This belief seems to arise from the mistaken view tl
   is the repeated construction of identical objects, while software production is the construction of unique objects,
   production of a newspaper — so that a software manager who cannot program is akin to a managing editor who

- Jalote, Pankaj (2002). *Software project management in practice*. Addison-Wesley. ISBN 0201737213.

Software Estimation

Software development efforts estimation is the process of predicting the most realistic use of effort rec
and/or noisy input. Effort estimates may be used as input to project plans, iteration plans, budgets, investment ana

State-of-practice

Published surveys on estimation practice suggest that expert estimation is the dominant strategy when estimating s

Typically, effort estimates are over-optimistic and there is a strong over-confidence in their accuracy. The mean eff
review of effort estimation error surveys, see [2]. However, the measurement of estimation error is not unproblema
strong over-confidence in the accuracy of the effort estimates is illustrated by the finding that, on average, if a softv
effort in a minimum-maximum interval, the observed frequency of including the actual effort is only 60-70% [3].

Currently the term "effort estimate" is used to denote as different concepts as most likely use of effort (modal valu
(median), the planned effort, the budgeted effort or the effort used to propose a bid or price to the client. This is l
and because the concepts serve different goals [4] [5].

History

Software researchers and practitioners have been addressing the problems of effort estimation for software develo
Nelson [7].

Most of the research has focused on the construction of formal software effort estimation models. The early model
from theories from other domains. Since then a high number of model building approaches have been evaluated,
regression trees, simulation, neural networks, Bayesian statistics, lexical analysis of requirement specifications, gene
computing, fuzzy logic modeling, statistical bootstrapping, and combinations of two or more of these models. T
estimation models COCOMO and SLIM have their basis in estimation research conducted in the 1970s and 1980s.
e.g., function points, is also based on research conducted in the 1970s and 1980s, but are re-appearing with modified
1990s and COSMIC (http://www.cosmicon.com) in the 2000s.

Estimation approaches

There are many ways of categorizing estimation approaches, see for example [9][10]. The top level categories are the

- Expert estimation: The quantification step, i.e., the step where the estimate is produced based on judgmental pr
- Formal estimation model: The quantification step is based on mechanical processes, e.g., the use of a formula de
- Combination-based estimation: The quantification step is based on a judgmental or mechanical combination of e

Below are examples of estimation approaches within each category.

Estimation approach	Category	Examples of support c
Analogy-based estimation	Formal estimation model	ANGEL, Weighted Micro Function Points
WBS-based (bottom up) estimation	Expert estimation	Project management software, company sp
Parametric models	Formal estimation model	COCOMO, SLIM, SEER-SEM
Size-based estimation models[11]	Formal estimation model	Function Point Analysis[12], Use Case Ana
Group estimation	Expert estimation	Planning poker, Wideband Delphi
Mechanical combination	Combination-based estimation	Average of an analogy-based and a Work b
Judgmental combination	Combination-based estimation	Expert judgment based on estimates from a

Selection of estimation approach

The evidence on differences in estimation accuracy of different estimation approaches and models suggest that ther or model in comparison to another depends strongly on the context [13]. This implies that different organizations [14], that may support the selection of estimation approach based on the expected accuracy of an approach include:

- Expert estimation is on average at least as accurate as model-based effort estimation. In particular, situations wi included in the model may suggest use of expert estimation. This assumes, of course, that experts with relevant
- Formal estimation models not tailored to a particular organization's own context, may be very inaccurate. Use o the estimation model's core relationships (e.g., formula parameters) are based on similar project contexts.
- Formal estimation models may be particularly useful in situations where the model is tailored to the organizatio is derived from similar projects and contexts), and/or it is likely that the experts' estimates will be subject to a s

The most robust finding, in many forecasting domains, is that combination of estimates from independent sources estimation accuracy [15] [16] [17].

In addition, other factors such as ease of understanding and communicating the results of an approach, ease of use c in a selection process.

Uncertainty assessment approaches

The uncertainty of an effort estimate can be described through a prediction interval (PI). An effort PI is based on value. For example, a project leader may estimate that the most likely effort of a project is 1000 work-hours and th work-hours. Then, the interval [500, 2000] work-hours is the 90% PI of the effort estimate of 1000 work-hours. F prediction limits, interval prediction, prediction region and, unfortunately, confidence interval. An important d uncertainty of an estimate, while confidence interval usually refers to the uncertainty associated with the parame mean value of a distribution of effort values. The confidence level of a PI refers to the expected (or subjective) proba

There are several possible approaches to calculate effort PIs, e.g., formal approaches based on regression or bootstra previous estimation error [20], and pure expert judgment of minimum-maximum effort for a given level of confide error has been found to systematically lead to more realistic uncertainty assessment than the traditional minimum-n

Assessing and interpreting the accuracy of effort estimates

The most common measures of the average estimation accuracy is the MMRE (Mean Magnitude of Relative Error),

$MRE = |\text{actual effort} - \text{estimated effort}| \ / \ |\text{actual effort}|$

This measure has been criticized [22] [23] [24] and there are several alternative measures, such as more symmetric m and Mean Variation from Estimate (MVFE) [27].

A high estimation error cannot automatically be interpreted as an indicator of low estimation ability. Alternative, c high complexity of development work, and more delivered functionality than originally estimated. A framework f included in [28].

Psychological issues related to effort estimation

There are many psychological factors potentially explaining the strong tendency towards over-optimistic effort estin These factors are essential even when using formal estimation models, because much of the input to these models is are: Wishful thinking, anchoring, planning fallacy and cognitive dissonance. A discussion on these and other factors

- It's easy to estimate what you know.
- It's hard to estimate what you know you don't know.
- It's very hard to estimate things that you don't know you don't know.

References

1. Jørgensen, M.. "A Review of Studies on Expert Estimation of Software Development Effort". http://simula.no/r
2. Molokken, K. Jorgensen, M.. "A review of software surveys on software effort estimation". http://ieeexplore.ieee.
3. Jørgensen, M. Teigen, K.H. Ribu, K.. "Better sure than safe? Over-confidence in judgement based software deve
http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6V0N-49N06GS-
5&_user=674998&_rdoc=1&_fmt=&_orig=search&_sort=d&view=c&_acct=C000036598&_version=1&_url
4. Edwards, J.S. Moores, T.T. (1994), "A conflict between the use of estimating and planning tools in the managem
3(2): 139-147.
5. Goodwin, P. (1998). Enhancing judgmental sales forecasting: The role of laboratory research. Forecasting with ju
91-112.
6. Farr, L. Nanus, B.. "Factors that affect the cost of computer programming". http://stinet.dtic.mil/oai/oai?verb=
7. Nelson, E. A. (1966). Management Handbook for the Estimation of Computer Programming Costs. AD-A648750
8. Anda, B. Angelvik, E. Ribu, K.. "Improving Estimation Practices by Applying Use Case Models". http://www.sp
9. Briand, L. C. and I. Wieczorek (2002). Resource estimation in software engineering. Encyclopedia of software en
10. Jørgensen, M. Shepperd, M.. "A Systematic Review of Software Development Cost Estimation Studies". http://s
11. Hill Peter (ISBSG) - Estimation Workbook 2 - published by International Software Benchmarking Standards Gr
ww.isbsg.org/ISBSGnew.nsf/WebPages/~GBL~Practical%20Project%20Estimation%202nd%20Edition)
12. Morris Pam - Overview of Function Point Analysis Total Metrics - Function Point Resource Centre (http://ww
13. Shepperd, M. Kadoda, G.. "Comparing software prediction techniques using simulation". http://ieeexplore.ieee.o
arnumber=965341.
14. Jørgensen, M.. "Estimation of Software Development Work Effort:Evidence on Expert Judgment and Formal Mo
http://simula.no/research/engineering/publications/Jorgensen.2007.2.
15. Winkler, R.L.. "Combining forecasts: A philosophical basis and some current issues Manager". http://www.scien
2B/2/d05dc6c369ab173c5792a05ea1be21d9.
16. Blattberg, R.C. Hoch, S.J.. "Database Models and Managerial Intuition: 50% Model + 50% Manager". http://wv
17. Jørgensen, M.. "Estimation of Software Development Work Effort:Evidence on Expert Judgment and Formal Mo
http://simula.no/research/engineering/publications/Jorgensen.2007.2.
18. Armstrong, J. S.. "Principles of forecasting: A handbook for researchers and practitioners". http://www.forecasti
19. Angelis, L. Stamelos, I.. "A simulation tool for efficient analogy based cost estimation". http://portal.acm.org/cit
20. Jørgensen, M. Sjøberg, D.I.K.. "An effort prediction interval approach based on the empirical distribution of pre
_ob=ArticleURL&_udi=B6V0B-47HC6S5-
1&_user=674998&_rdoc=1&_fmt=&_orig=search&_sort=d&view=c&_acct=C000036598&_version=1&_url
21. Jørgensen, M.. "Realism in assessment of effort estimation uncertainty: It matters how you ask". http://simula.n
22. Shepperd, M. Cartwright, M. Kadoda, G.. "On Building Prediction Systems for Software Engineers".
http://www.ingentaconnect.com/content/klu/emse/2000/00000005/00000003/00278191.
23. Kitchenham, B. Pickard, L.M. MacDonell, S.G. Shepperd,. "What accuracy statistics really measure". http://scit
prog=normal&id=IPSEFU000148000003000081000001&idtype=cvips&gifs=yes.
24. Foss, T. Stensrud, E. Kitchenham, B. Myrtveit, I.. "A Simulation Study of the Model Evaluation Criterion MMI
25. Miyazaki, Y. Terakado, M. Ozaki, K. Nozaki, H.. "Robust regression for developing software estimation models".
26. Lo, B. Gao, X.. "Assessing Software Cost Estimation Models: criteria for accuracy, consistency and regression". h
27. Hughes, R.T. Cunliffe, A. Young-Martos, F.. "Evaluating software development effort model-building techniques
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=689296.
28. Grimstad, S. Jørgensen, M.. "A Framework for the Analysis of Software Cost Estimation Accuracy".
http://simula.no/research/engineering/publications/Grimstad.2006.2/simula_pdf_file.
29. Jørgensen, M. Grimstad, S.. "How to Avoid Impact from Irrelevant and Misleading Information When Estimatin
http://simula.no/research/engineering/publications/Simula.SE.112.

External links

- Industry Productivity data for Input into Software Development Estimates and guidance and tools for Estimatic

http://www.isbsg.org

- Free first-order benchmarking utility from Software Benchmarking Organization: http://www.sw-benchmarking.
- Special Interest Group on Software Effort Estimation: http://www.forecastingprinciples.com/Software_Estimati
- General forecasting principles: http://www.forecastingprinciples.com
- Project estimation tools: http://www.projectmanagementguides.com/TOOLS/project_estimation_tools.html
- Downloadable research papers on effort estimation: http://simula.no/research/engineering/projects/best
- Mike Cohn's Estimating With Use Case Points from article from Methods & Tools: http://www.methodsandtool
- Resources on Software Estimation from Steve McConnell: http://www.construx.com/Page.aspx?nid=297
- Resources on Software Estimation from Dan Galorath: http://www.galorath.com/wp/

Cost Estimation

The ability to accurately estimate the time and/or cost taken for a project to come in to its successful conclusio clearly defined and well understood software development process has, in recent years, shown itself to be the most statistical estimation. In particular, the act of sampling more frequently, coupled with the loosening of constraints more rapid development times.

Methods

Popular methods for estimation in software engineering include:

- Analysis Effort method
- COCOMO
- COSYSMO
- Evidence-based Scheduling Refinement of typical agile estimating techniques using minimal measurement and to
- Function Point Analysis
- Parametric Estimating
- PRICE Systems Founders of Commercial Parametric models that estimates the scope, cost, effort and schedule f
- Proxy-based estimating (PROBE) (from the Personal Software Process)
- Program Evaluation and Review Technique (PERT)
- SEER-SEM Parametric Estimation of Effort, Schedule, Cost, Risk. Mimimum time and staffing concepts based
- SLIM
- The Planning Game (from Extreme Programming)
- Weighted Micro Function Points (WMFP)
- Wideband Delphi

External links

- Software Estimation chapter (http://www.stellman-greene.com/ch03) from Applied Software Project Manageme
- Article Estimating With Use Case Points (http://www.methodsandtools.com/archive/archive.php?id=25) from
- The Dynamics of Software Projects Estimation (http://softwaresurvival.blogspot.com/2006/11/dynamics-of-effo
- Resources on Software Estimation (http://www.construx.com/Page.aspx?nid=297) from Steve McConnell
- Links on tools and techniques of software estimation (http://www.uduko.com/topic_detail/details/47)
- Article Estimating techniques throughout the SDLC (http://www.gem-up.com/PDF/SK903V1_WP_Estimating

Development Speed

Even though there isn't a common way to calculate development speed, in recent trenda, software communities hav

The term velocity (just like is physics) is how fast the team, from the starting point until the goal. In the softwar iteration.

Tools

Introduction

Basically, for every step in the development process there are tools available.

- **Modelling and Case Tools:** StarUML, objectiF, Visio, ArgoUML
- **Writing Code:** IDEs like Eclipse, Netbeans, Visual Studio; Compilers and Debuggers; SourceControl like CV

- **Testing Code:** Testing frameworks like JUnit, FIT, TestNG, HTMLUnit; Coverage with Clover, NCover; Prc
- **Automation:** Build tools: make, Ant, Maven,
- **Documentation:** JavaDoc, Doxygen, NDoc; Wikis
- **Project Management, Bug Tracking,Continuous Integration:** Trac, Bugzilla, Mantis; CruiseContr
- **Re-engineering:** Decompiler: JAD; Obfuscators

Some of these tools we have talked about before, but some we still need to learn about.

Modelling and Case Tools

Computer-aided software engineering (**CASE**) is the scientific application of a set of tools and methods to result in high-quality, defect-free, and maintainable software products.[1] It also refers to methods for the together with automated tools that can be used in the software development process.[2]

Overview

The term "computer-aided software engineering" (CASE) can refer to the software used for the automated de computer code. The CASE functions include analysis, design, and programming. CASE tools automate metho producing structured computer code in the desired programming language.

CASE software supports the software process activities such as requirement engineering, design, program develop tools include design editors, data dictionaries, compilers, debuggers, system building tools, etc.

CASE also refers to the methods dedicated to an engineering discipline for the development of information system u

CASE is mainly used for the development of quality software which will perform effectively.

History

The ISDOS project at the University of Michigan initiated a great deal of interest in the whole concept of using cor requirements and developing systems. Several papers by Daniel Teichroew fired a whole generation of enthusiasts w was a CASE tool although it predated the term. His insights into the power of meta-meta-models was inspiring, p Program Director at University of Maryland University College.

Another major thread emerged as a logical extension to the DBMS directory. By extending the range of meta-data and used at runtime. This "active dictionary" became the precursor to the more modern "model driven execution graphical representation of any of the meta-data. It was the linking of the concept of a dictionary holding analysts together with the graphical representation of such data that gave rise to the earlier versions of I-CASE.

The term CASE was originally coined by software company Nastec Corporation of Southfield, Michigan in 1982 wi also was the first microcomputer-based system to use hyperlinks to cross-reference text strings in documents—an ea DesignAid, was the first microprocessor-based tool to logically and semantically evaluate software and system desigr

Under the direction of Albert F. Case, Jr. vice president for product management and consulting, and Vaughn Fr expanded to support analysis of a wide range of structured analysis and design methodologies, notably Ed Yourdc time) SA/SD and Warnier-Orr (data driven).

The next entrant into the market was Excelerator from Index Technology in Cambridge, Mass. While DesignAid microcomputers, Index launched Excelerator on the IBM PC/AT platform. While, at the time of launch, and centralized database as did the Convergent Technologies or Burroughs machines, the allure of IBM was strong, and rash of offerings from companies such as Knowledgeware (James Martin, Fran Tarkenton and Don Addingto (METHOD/1, DESIGN/1, INSTALL/1, FCP).

CASE tools were at their peak in the early 1990s. At the time IBM had proposed AD/Cycle, which was an allianc IBM DB2 in mainframe and OS/2:

> *The application development tools can be from several sources: from IBM, from vendors, and from the customer Information Systems, Index Technology Corporation, and Knowledgeware, Inc. wherein selected products from t marketing program to provide offerings that will help to achieve complete life-cycle coverage.*[3]

With the decline of the mainframe, AD/Cycle and the Big CASE tools died off, opening the market for the mainstr of the early 1990s ended up being purchased by Computer Associates, including IEW, IEF, ADW, Cayenne, and Le

Supporting software

Alfonso Fuggetta classified CASE into 3 categories:[4]

1. *Tasks* support only specific tasks in the software process.
2. *Workbenches* support only one or a few activities.
3. *Environments* support (a large part of) the software process.

Workbenches and environments are generally built as collections of tools. Tools can therefore be either stand alone p

Tools

CASE tools are a class of software that automate many of the activities involved in various life cycle phases. For application, prototyping tools can be used to develop graphic models of application screens to assist end users to system designers can use automated design tools to transform the prototyped functional requirements into de generators to convert the design documents into code. Automated tools can be used collectively, as mentioned, c application requirements that get passed to design technicians who convert the requirements into detailed desig without the assistance of automated design software.[5]

Existing CASE tools can be classified along 4 different dimensions:

1. Life-cycle support
2. Integration dimension
3. Construction dimension
4. Knowledge-based CASE dimension[6]

Let us take the meaning of these dimensions along with their examples one by one:

Life-Cycle Based CASE Tools

This dimension classifies CASE Tools on the basis of the activities they support in the information systems life cycle

- Upper CASE Tools support strategic planning and construction of concept-level products and ignore the design a diagrams, Data flow diagram, Structure charts, Decision Trees, Decision tables, etc.
- Lower CASE Tools concentrate on the back end activities of the software life cycle, such as physical design, debu reengineering and reverse engineering.

Integration dimension

Three main CASE Integration dimensions have been proposed:[7]

1. CASE Framework
2. ICASE Tools
3. Integrated Project Support Environment(IPSE)

Workbenches

Workbenches integrate several CASE tools into one application to support specific software-process activities. Hence

- a homogeneous and consistent interface (presentation integration).
- easy invocation of tools and tool chains (control integration).
- access to a common data set managed in a centralized way (data integration).

CASE workbenches can be further classified into following 8 classes:[4]

1. Business planning and modeling
2. Analysis and design
3. User-interface development
4. Programming
5. Verification and validation
6. Maintenance and reverse engineering
7. Configuration management
8. Project management

Environments

An environment is a collection of CASE tools and workbenches that supports the software process. CASE environme

1. Toolkits
2. Language-centered
3. Integrated
4. Fourth generation
5. Process-centered

Toolkits

Toolkits are loosely integrated collections of products easily extended by aggregating different tools and worl programming, configuration management and project management. And the toolkit itself is environments exten Programmer's Work Bench and the VMS VAX Set. In addition, toolkits' loose integration requires user to activate files are unstructured and could be in different format, therefore the access of file from different tools may requir adding a new component is the formats of the files, toolkits can be easily and incrementally extended.[4]

Language-centered

The environment itself is written in the programming language for which it was developed, thus enabling users t different languages is a major issue for language-centered environments. Lack of process and data integration is als presentation and control integration. Interlisp, Smalltalk, Rational, and KEE are examples of language-centered env

Integrated

These environments achieve presentation integration by providing uniform, consistent, and coherent tool and wor concept: they have a specialized database managing all information produced and accessed in the environment Cohesion.[4]

Fourth-generation

Fourth-generation environments were the first integrated environments. They are sets of tools and workbenches su processing and business-oriented applications. In general, they include programming tools, simple configuration n generator to produce code in lower level languages. Informix 4GL, and Focus fall into this category.[4]

Process-centered

Environments in this category focus on process integration with other integration dimensions as starting points. A created by specialized tools. They usually consist of tools handling two functions:

- Process-model execution
- Process-model production

Examples are East, Enterprise II, Process Wise, Process Weaver, and Arcadia.[4]

Applications

All aspects of the software development life cycle can be supported by software tools, and so the use of tools from management software through tools for business and functional analysis, system design, code storage, compilers, tra

However, tools that are concerned with analysis and design, and with using design information to create parts (o tools. CASE applied, for instance, to a database software product, might normally involve:

- Modeling business / real-world processes and data flow
- Development of data models in the form of entity-relationship diagrams
- Development of process and function descriptions

Risks and associated controls

Common CASE risks and associated controls include:

- *Inadequate standardization*: Linking CASE tools from different vendors (design tool from Company X, programn standardized code structures and data classifications. File formats can be converted, but usually not economicall based on standard protocols and insisting on demonstrated compatibility. Additionally, if organizations obtain t acquiring them from a vendor that has a full line of products to ensure future compatibility if they add more too
- *Unrealistic expectations*: Organizations often implement CASE technologies to reduce development costs. Implen management must be willing to accept a long-term payback period. Controls include requiring senior managers t technologies.[5]
- *Slow implementation*: Implementing CASE technologies can involve a significant change from traditional develop tools the first time on critical projects or projects with short deadlines because of the lengthy training process. A less complex projects and gradually implementing the tools to allow more training time.[5]
- *Weak repository controls*: Failure to adequately control access to CASE repositories may result in security breac stored in the repository. Controls include protecting the repositories with appropriate access, version, and backu

References

1. Kuhn, D.L (1989). "Selecting and effectively using a computer aided software engineering tool". Annual Westingl Project.
2. P. Loucopoulos and V. Karakostas (1995). *System Requirements Engineering*. McGraw-Hill.
3. "AD/Cycle strategy and architecture", IBM Systems Journal, Vol 29, NO 2, 1990; p. 172.
4. Alfonso Fuggetta (December 1993). "A classification of CASE technology". *Computer* **26** (12): 25–38. doi:10.110 http://www2.computer.org/portal/web/csdl/abs/mags/co/1993/12/rz025abs.htm. Retrieved 2009-03-14.
5. Software Development Techniques (http://www.ffiec.gov/ffiecinfobase/booklets/d_a/10.html). In: *FFIEC InfoE*
6. Software Engineering: Tools, Principles and Techniques by Sangeeta Sabharwal, Umesh Publications
7. Evans R. Rock. *Case Analyst Workbenches: A Detailed Product Evaluation*. Volume 1, pp. 229–242 by

External links

- CASE Tools (http://case-tools.org/) A CASE tools' community with comments, tags, forums, articles, reviews, e
- CASE tool index (http://www.unl.csi.cuny.edu/faqs/software-enginering/tools.html) - A comprehensive list of C
- UML CASE tools (http://www.objectsbydesign.com/tools/umltools_byProduct.html) - A comprehensive list of and some related to MDA CASE Tools.

Compiler

A **compiler** is a computer program (or set of programs) that transforms source code written in a programming la another computer language (the *target language*, often having a binary form known as *object code*). The most transform source code is to create an executable program.

The name "compiler" is primarily used for programs that translate source code from a high-level programming la (e.g., assembly language or machine code). If the compiled program can run on a computer whose CPU or operatin on which the compiler runs, the compiler is known as a cross-compiler. A program that translates from a low level l *decompiler*. A program that translates between high-level languages is usually called a *language translator*, *source converter*. A *language rewriter* is usually a program that translates the form of expressions without a change of lang

A compiler is likely to perform many or all of the following operations: lexical analysis, preprocessing, parsing, s translation), code generation, and code optimization.

Program faults caused by incorrect compiler behavior can be very difficult to track down and work around; therefor lot of time ensuring the correctness of their software.

The term compiler-compiler is sometimes used to refer to a parser generator, a tool often used to help create the lex

History

Software for early computers was primarily written in assembly language for many years. Higher level programn until the benefits of being able to reuse software on different kinds of CPUs started to become significantly g compiler. The very limited memory capacity of early computers also created many technical problems when implem

Towards the end of the 1950s, machine-independent programming languages were first proposed. Subsequently, se developed. The first compiler was written by Grace Hopper, in 1952, for the A-0 programming language. The FOR at IBM is generally credited as having introduced the first complete compiler in 1957. COBOL was an early languag

In many application domains the idea of using a higher level language quickly caught on. Because of the expan increasing complexity of computer architectures, compilers have become more and more complex.

Early compilers were written in assembly language. The first *self-hosting* compiler — capable of compiling its own s and Mike Levin at MIT in 1962.[2] Since the 1970s it has become common practice to implement a compiler in th choices for implementation language. Building a self-hosting compiler is a bootstrapping problem—the first such co different language, or (as in Hart and Levin's Lisp compiler) compiled by running the compiler in an interpreter.

Compilers in education

Compiler construction and compiler optimization are taught at universities and schools as part of the compute implementation of a compiler for an educational programming language. A well-documented example is Niklaus Wir the 1970s.[3] In spite of its simplicity, the PL/0 compiler introduced several influential concepts to the field:

1. Program development by stepwise refinement (also the title of a 1971 paper by Wirth)[4]
2. The use of a recursive descent parser
3. The use of EBNF to specify the syntax of a language
4. A code generator producing portable P-code
5. The use of T-diagrams[5] in the formal description of the bootstrapping problem

Compilation

Compilers enabled the development of programs that are machine-independent. Before the development of FOR' 1950s, machine-dependent assembly language was widely used. While assembly language produces more reusable ar has to be modified or rewritten if the program is to be executed on different hardware architecture.

With the advance of high-level programming languages soon followed after FORTRAN, such as COBOL, C, BA compiler translates the high-level source programs into target programs in machine languages for the specific hard program.

The structure of a compiler

Compilers bridge source programs in high-level languages with the underlying hardware. A compiler requires 1) dete and efficient object code, 3) run-time organization, and 4) formatting output according to assembler and/or linker middle-end, and the backend.

The **frontend** checks whether the program is correctly written in terms of the programming language syntax an reported, if any, in a useful way. Type checking is also performed by collecting type information. The frontend the processing by the middle-end.

The **middle-end** is where optimization takes place. Typical transformations for optimization are removal of usel relocation of computation to a less frequently executed place (e.g., out of a loop), or specialization of computatic following backend. Most optimization efforts are focused on this part.

The **backend** is responsible for translating the IR from the middle-end into assembly code. The target instruction(registers. Backend utilizes the hardware by figuring out how to keep parallel FUs busy, filling delay slots, and s techniques are well-developed.

Compiler output

One classification of compilers is by the platform on which their generated code executes. This is known as the *targe*

A *native* or *hosted* compiler is one which output is intended to directly run on the same type of computer and op compiler is designed to run on a different platform. Cross compilers are often used when developing software development environment.

The output of a compiler that produces code for a virtual machine (VM) may or may not be executed on the compilers are not usually classified as native or cross compilers.

Compiled versus interpreted languages

Higher-level programming languages are generally divided for convenience into compiled languages and interpre language that *requires* it to be exclusively compiled or exclusively interpreted, although it is possible to design la usually reflects the most popular or widespread implementations of a language — for instance, BASIC is sometir existence of BASIC compilers and C interpreters.

Modern trends toward just-in-time compilation and bytecode interpretation at times blur the traditional categorizat

Some language specifications spell out that implementations *must* include a compilation facility; for example, C Common Lisp that stops it from being interpreted. Other languages have features that are very easy to implen example, APL, SNOBOL4, and many scripting languages allow programs to construct arbitrary source code at r passing it to a special evaluation function. To implement these features in a compiled language, programs must u compiler itself.

Hardware compilation

The output of some compilers may target hardware at a very low level, for example a Field Programmable Gate Arr Such compilers are said to be *hardware compilers* or synthesis tools because the source code they compile effectively output of the compilation are not instructions that are executed in sequence - only an interconnection of transistor for configuring FPGAs. Similar tools are available from Altera, Synplicity, Synopsys and other vendors.

Compiler construction

In the early days, the approach taken to compiler design used to be directly affected by the complexity of the pr available.

A compiler for a relatively simple language written by one person might be a single, monolithic piece of software. V is required, the design may be split into a number of relatively independent phases. Having separate phases means people. It also becomes much easier to replace a single phase by an improved one, or to insert new phases later (e.g.

The division of the compilation processes into phases was championed by the Production Quality Compiler-C introduced the terms *front end*, *middle end*, and *back end*.

All but the smallest of compilers have more than two phases. However, these phases are usually regarded as being p meet is open to debate. The front end is generally considered to be where syntactic and semantic processing take source code).

The middle end is usually designed to perform optimizations on a form other than the source code or machine co generic optimizations to be shared between versions of the compiler supporting different languages and target proces

The back end takes the output from the middle. It may perform more analysis, transformations and optimizati particular processor and OS.

This front-end/middle/back-end approach makes it possible to combine front ends for different languages with bac GNU Compiler Collection, LLVM, and the Amsterdam Compiler Kit, which have multiple front-ends, shared analys

One-pass versus multi-pass compilers

Classifying compilers by number of passes has its background in the hardware resource limitations of computers. C
have enough memory to contain one program that did all of this work. So compilers were split up into smaller prog
it) performing some of the required analysis and translations.

The ability to compile in a single pass has classically been seen as a benefit because it simplifies the job of writing
pass compilers. Thus, partly driven by the resource limitations of early systems, many early languages were spec
Pascal).

In some cases the design of a language feature may require a compiler to perform more than one pass over the sourc
which affects the translation of a statement appearing on line 10. In this case, the first pass needs to gather informa
the actual translation happening during a subsequent pass.

The disadvantage of compiling in a single pass is that it is not possible to perform many of the sophisticated optimi
exactly how many passes an optimizing compiler makes. For instance, different phases of optimization may analyse

Splitting a compiler up into small programs is a technique used by researchers interested in producing provably cor
requires less effort than proving the correctness of a larger, single, equivalent program.

While the typical multi-pass compiler outputs machine code from its final pass, there are several other types:

- A "source-to-source compiler" is a type of compiler that takes a high level language as its input and outputs a hi
  frequently take in a high level language program as an input and then transform the code and annotate it with p
  Fortran's DOALL statements).
- Stage compiler that compiles to assembly language of a theoretical machine, like some Prolog implementations
  - This Prolog machine is also known as the Warren Abstract Machine (or WAM).
  - Bytecode compilers for Java, Python, and many more are also a subtype of this.
- Just-in-time compiler, used by Smalltalk and Java systems, and also by Microsoft .NET's Common Intermediate
  - Applications are delivered in bytecode, which is compiled to native machine code just prior to execution.

Front end

The front end analyzes the source code to build an internal representation of the program, called the intermediate r
mapping each symbol in the source code to associated information such as location, type and scope. This is done ove

1. **Line reconstruction**. Languages which strop their keywords or allow arbitrary spaces within identifiers requi
   to a canonical form ready for the parser. The top-down, recursive-descent, table-driven parsers used in the 1960s
   separate tokenizing phase. Atlas Autocode, and Imp (and some implementations of ALGOL and Coral 66) are ex
   Reconstruction phase.
2. Lexical analysis breaks the source code text into small pieces called *tokens*. Each token is a single atomic unit of
   token syntax is typically a regular language, so a finite state automaton constructed from a regular expression ca
   and the software doing lexical analysis is called a lexical analyzer or scanner.
3. Preprocessing. Some languages, e.g., C, require a preprocessing phase which supports macro substitution and cor
   syntactic or semantic analysis; e.g. in the case of C, the preprocessor manipulates lexical tokens rather than synt
   substitutions based on syntactic forms.
4. Syntax analysis involves parsing the token sequence to identify the syntactic structure of the program. This pha
   tokens with a tree structure built according to the rules of a formal grammar which define the language's syntax
   phases in the compiler.
5. Semantic analysis is the phase in which the compiler adds semantic information to the parse tree and builds the
   checking (checking for type errors), or object binding (associating variable and function references with their def
   initialized before use), rejecting incorrect programs or issuing warnings. Semantic analysis usually requires a com
   phase, and logically precedes the code generation phase, though it is often possible to fold multiple phases into o

Back end

The term *back end* is sometimes confused with *code generator* because of the overlapped functionality of generat
generic analysis and optimization phases in the back end from the machine-dependent code generators.

The main phases of the back end include the following:

1. Analysis: This is the gathering of program information from the intermediate representation derived from the inp
   dependence analysis, alias analysis, pointer analysis, escape analysis etc. Accurate analysis is the basis for any co
   also built during the analysis phase.
2. Optimization: the intermediate language representation is transformed into functionally equivalent but faster (or

elimination, constant propagation, loop transformation, register allocation and even automatic parallelization.

3. Code generation: the transformed intermediate language is translated into the output language, usually the nativ decisions, such as deciding which variables to fit into registers and memory and the selection and scheduling of a modes (see also Sethi-Ullman algorithm).

Compiler analysis is the prerequisite for any compiler optimization, and they tightly work together. For example, de

In addition, the scope of compiler analysis and optimizations vary greatly, from as small as a basic block to the pr optimization). Obviously, a compiler can potentially do a better job using a broader view. But that broad view is nc compilation time and memory space; this is especially true for interprocedural analysis and optimizations.

Interprocedural analysis and optimizations are common in modern commercial compilers from HP, IBM, SGI, Intel, for a long time for lacking powerful interprocedural optimizations, but it is changing in this respect. Another ope Open64, which is used by many organizations for research and commercial purposes.

Due to the extra time and space needed for compiler analysis and optimizations, some compilers skip them by defa which optimizations should be enabled.

Compiler correctness

Compiler correctness is the branch of software engineering that deals with trying to show that a compiler beha include developing the compiler using formal methods and using rigorous testing (often called compiler validation) o

Related techniques

Assembly language is a type of low-level language and a program that compiles it is more commonly known as an *a*

A program that translates from a low level language to a higher level one is a *decompiler*.

A program that translates between high-level languages is usually called a *language translator*, *source to source* usually applied to translations that do not involve a change of language.

International conferences and organizations

Every year, the **European Joint Conferences on Theory and Practice of Software** (ETAPS) sponso (CC), with papers from both the academic and industrial sectors.[6]

Notes

1. "IP: The World's First COBOL Compilers". interesting-people.org. 12 June 1997. http://www.interesting-people.
2. T. Hart and M. Levin. "The New Compiler, AIM-39 - CSAIL Digital Archive - Artificial Intelligence Laboratory publications/pdf/AIM-039.pdf.
3. "The PL/0 compiler/interpreter". http://www.246.dk/pl0.html.
4. "The ACM Digital Library". http://www.acm.org/classics/dec95/.
5. T diagrams were first introduced for describing bootstrapping and cross-compiling compilers in McKeeman et al. before that with his UNCOL in 1958, to which Bratman added in 1961: H. Bratman, "An alternate form of the others, including P.D. Terry, gave an explanation and usage of T-diagrams in their textbooks on the topic of cor ompilers/cha03g.htm). T-diagrams are also now used to describe client-server interconnectivity on the World Wi *Language to Illustrate WWW Technology* (http://pu.rbg.informatik.tu-darmstadt.de/docs/HJH-19990217-etal-T Germany
6. ETAPS (http://www.etaps.org/) - European Joint Conferences on Theory and Practice of Software. Cf. "CC" (C

References

- Compiler textbook references (http://www.informatik.uni-trier.de/~ley/db/books/compiler/index.html) A collec
- Aho, Alfred V.; Sethi, Ravi; and Ullman, Jeffrey D., *Compilers: Principles, Techniques and Tools* ISBN 0-201-1(/0,4096,0201100886,00.html). Also known as "The Dragon Book."
- Allen, Frances E., "A History of Language Processor Technology in IBM" (http://www.research.ibm.com/journa v.25, no.5, September 1981.
- Allen, Randy; and Kennedy, Ken, *Optimizing Compilers for Modern Architectures*, Morgan Kaufmann Publishe
- Appel, Andrew Wilson
  - *Modern Compiler Implementation in Java*, 2nd edition. Cambridge University Press, 2002. ISBN 0-521-8206(
  - *Modern Compiler Implementation in ML* (http://books.google.com/books?id=8APOYafUt-oC&printsec=fro
- Bornat, Richard, *Understanding and Writing Compilers: A Do It Yourself Guide* (http://www.cs.mdx.ac.uk/sta ISBN 0-333-21732-2

- Cooper, Keith D., and Torczon, Linda, *Engineering a Compiler*, Morgan Kaufmann, 2004, ISBN 1-55860-699-8.
- Leverett; Cattel; Hobbs; Newcomer; Reiner; Schatz; Wulf, *An Overview of the Production Quality Compiler-Cor*
- McKeeman, William Marshall; Horning, James J.; Wortman, David B., *A Compiler Generator* (http://www.cs.t 0-13-155077-2
- Muchnick, Steven, *Advanced Compiler Design and Implementation* (http://books.google.com/books?id=Pq7pHw Morgan Kaufmann Publishers, 1997. ISBN 1-55860-320-4
- Scott, Michael Lee, *Programming Language Pragmatics* (http://books.google.com/books?id=4LMtA2wOsPcC&p Morgan Kaufmann, 2005, 2nd edition, 912 pages. ISBN 0-12-633951-1 (The author's site on this book (http://w
- Srikant, Y. N.; Shankar, Priti, *The Compiler Design Handbook: Optimizations and Machine Code Generation* (h id=0K_jIsgyNpoC&printsec=frontcover), CRC Press, 2003. ISBN 0-8493-1240-X
- Terry, Patrick D., *Compilers and Compiler Generators: An Introduction with C++* (http://scifac.ru.ac.za/comp 1-85032-298-8,
- Wirth, Niklaus, *Compiler Construction* (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.70.3774&rep pages. Revised November 2005.

External links

- The comp.compilers newsgroup and RSS feed (http://compilers.iecc.com/)
- Hardware compilation mailing list (http://www.jiscmail.ac.uk/lists/hwcomp.html)
- Practical introduction to compiler construction using flex and yacc (http://www.onyxbits.de/content/blog/patri
- Book "Basics of Compiler Design (http://www.diku.dk/hjemmesider/ansatte/torbenm/Basics/)" by Torben Ægi

---

A **debugger** or **debugging tool** is a computer program that is used to test and debug other programs (the "tar on an *instruction set simulator* (ISS), a technique that allows great power in its ability to halt when specific condit executing the code directly on the appropriate (or the same) processor. Some debuggers offer two modes of operatio

A "crash" happens when the program cannot normally continue because of a programming bug. For example, th current version of the CPU or attempted to access unavailable or protected memory. When the program "crashes" in the original code if it is a **source-level debugger** or **symbolic debugger**, commonly now seen in integr **machine-language debugger** it shows the line in the disassembly (unless it also has online access to the origi assembly or compilation).

Typically, debuggers also offer more sophisticated functions such as running a program step by step (**single-s** program to examine the current state) at some event or specified instruction by means of a breakpoint, and trac modify the state of the program while it is running, rather than merely to observe it. It may also be possible to con or logical error.

The importance of a good debugger cannot be overstated. Indeed, the existence and quality of such a tool for a give if another language/platform is better-suited to the task.[*citation needed*]. The absence of a debugger, having once blind man in a dark room looking for a black cat that isn't there".[1] However, software can (and often does) b inevitable changes the presence of a debugger will make to a software program's internal timing. As a result, ev runtime problems in complex multi-threaded or distributed systems.

The same functionality which makes a debugger useful for eliminating bugs allows it to be used as a software crack software protection features. It often also makes it useful as a general testing verification tool test coverage and perf

Most current mainstream debugging engines, such as gdb and dbx provide console-based command line interfaces provide IDE integration, program animation, and visualization features. Some early mainframe debuggers such Interactive test/debug) provided this same functionality for the IBM System/360 and later operating systems, as lo

Language dependency

Some debuggers operate on a single specific language while others can handle multiple languages transparently. Fo Assembler subroutines and also PL/1 subroutines, the debugger may dynamically switch modes to accommodate th

Memory protection

Some debuggers also incorporate memory protection to avoid storage violations such as buffer overflow. This may memory is dynamically allocated from memory 'pools' on a task by task basis.

Hardware support for debugging

Most modern microprocessors have at least one of these features in their CPU design to make debugging easier:

- hardware support for single-stepping a program, such as the trap flag.
- An instruction set that meets the Popek and Goldberg virtualization requirements makes it easier to write debu debugged; such a CPU can execute the inner loops of the program under test at full speed, and still remain unde
- In-System Programming allows an external hardware debugger to re-program a system under test (for example,

ISP support also have other hardware debug support.

- Hardware support for code and data breakpoints, such as address comparators and data value comparators or, w
- JTAG access to hardware debug interfaces such as those on ARM architecture processors or using the Nexus cor extensive JTAG debug support.
- Microcontrollers with as few as six pins need to use low pin-count substitutes for JTAG, such as BDM, Spy-Bi-W bidirectional signaling on the RESET pin.

List of debuggers

- AppPuncher Debugger — for debugging Rich Internet Applications
- AQtime
- CA/EZTEST — was a CICS interactive test/debug software package
- CharmDebug (http://charm.cs.uiuc.edu/research/parallel_debug/) — a Debugger for Charm++
- CodeView
- DBG — a PHP Debugger and Profiler
- dbx
- DDD (Data Display Debugger)
- Distributed Debugging Tool (Allinea DDT)
- DDTLite — Allinea DDTLite for Visual Studio 2008
- DEBUG — the built-in debugger of DOS and Microsoft Windows
- Debugger for MySQL (http://www.mydebugger.com)
- Opera Dragonfly
- Dynamic debugging technique (DDT), and its octal counterpart Octal Debugging Technique
- Eclipse
- Embedded System Debug Plug-in for Eclipse
- FusionDebug
- gDEBugger (http://www.gremedy.com/gDEBuggerCL.php) OpenGL, OpenGL ES and OpenCL Debugger and I
- GNU Debugger (GDB), GNU Binutils
- Intel Debugger (IDB)
- Insight
- Parasoft Insure++
- iSYSTEM — In circuit debugger for Embedded Systems
- Interactive Disassembler (IDA Pro)
- Java Platform Debugger Architecture
- Jinx — a whole-system debugger for heisenbugs. It works transparently as a device driver.
- JSwat — open-source Java debugger
- LLDB
- MacsBug
- Nemiver — graphical C/C++ Debugger for the GNOME desktop environment
- OLIVER (CICS interactive test/debug) - a GUI equipped *instruction set simulator* (ISS)
- OllyDbg
- FlexTracer - shareware debugger for SQL statements
- Omniscient Debugger — Forward and backward debugger for Java
- pydbg
- IBM Rational Purify
- RealView Debugger — Commercial debugger produced for and designed by ARM
- sdb
- SIMMON (Simulation Monitor)
- SIMON (Batch Interactive test/debug) — a GUI equipped Instruction Set Simulator for batch

- SoftICE
- TimeMachine — Forward and backward debugger designed by Green Hills Software
- TotalView
- TRACE32 — In circuit debugger for Embedded Systems
- Turbo Debugger
- Ups — C, Fortran source level debugger
- Valgrind
- VB Watch Debugger — debugger for Visual Basic 6.0
- Microsoft Visual Studio Debugger
- WinDbg
- Xdebug — PHP debugger and profiler

Debugger front-ends

Some of the most capable and popular debuggers only implement a simple command line interface (CLI) — often t
via a graphical user interface (GUI) can be considered easier and more productive though. This is the reason f
subservient CLI-only debuggers via graphical user interface. Some GUI debugger front-ends are designed to be con
at one specific debugger.

List of debugger front-ends

- Many Integrated development environments come with integrated debuggers (or front-ends to standard debugge
  - Many Eclipse perspectives, e.g. the Java Development Tools (JDT) [13] (http://www.eclipse.org/jdt/index.p
- DDD is the standard front-end from the GNU Project. It is a complex tool that works with most common debug
  natively or with some external programs (for PHP).
- GDB (the GNU debugger) GUI
  - Emacs — Emacs editor with built in support for the GNU Debugger acts as the frontend.
  - KDbg — Part of the KDE development tools.
  - Nemiver — A GDB frontend that integrates well in the GNOME desktop environment.
  - xxgdb — X-window frontend for GDB and dbx debugger.
  - Qt Creator — multi-platform frontend for GDB (debugging example (http://doc.qt.nokia.com/qtcreator-2.0/
  - cgdb (http://cgdb.sourceforge.net/) — ncurses terminal program that mimics vim key mapping.
  - ccdebug (http://ccdebug.sourceforge.net/)— A graphical GDB frontend using the Qt toolkit.
  - Padb — has a parallel front-end to GDB allowing it to target parallel applications.
  - Allinea Distributed Debugging Tool — a parallel and distributed front-end to a modified version of GDB.
  - Xcode — contains a GDB front-end as well.
  - SlickEdit — contains a GDB front-end as well.
  - Eclipse C/C++ Development Tools (CDT) [14] (http://www.eclipse.org/cdt/) — includes visual debugging

References

- Jonathan B. Rosenberg, *How Debuggers Work: Algorithms, Data Structures, and Architecture*, John Wiley & So

1. http://www.berniecode.com/blog/2007/03/08/how-to-debug-javascript-with-visual-web-developer-express/

External links

- Debugging tools (http://www.dmoz.org//Computers/Programming/Development_Tools/Debugging/) at DMOZ
- Debugging Tools for Windows (http://www.microsoft.com/whdc/devtools/debugging/)
- OpenRCE: Various Debugger Resources and Plug-ins (http://www.openrce.org)
- Parallel computing development and debugging tools (http://www.dmoz.org//Computers/Parallel_Computing/

IDE

An **integrated development environment (IDE)** (also known as **integrated design environment** o
that provides comprehensive facilities to computer programmers for software development. An IDE normally consist

- a source code editor

- a compiler and/or an interpreter
- build automation tools
- a debugger

Sometimes a version control system and various tools are integrated to simplify the construction of a GUI. Many n an object inspector, and a class hierarchy diagram, for use with object-oriented software development.[1]

Overview

IDEs are designed to maximize programmer productivity by providing tightly-knit components with similar user programmer has to do less mode switching versus using discrete development programs. However, because an IDE is very nature, this higher productivity only occurs after a lengthy learning process.

Typically an IDE is dedicated to a specific programming language, allowing a feature set that most closely matcl language. However, there are some multiple-language IDEs, such as Eclipse, ActiveState Komodo, recent versions WinDev, and Xcode.

IDEs typically present a single program in which all development is done. This program typically provides ma compiling, deploying and debugging software. The aim is to abstract the configuration necessary to piece togethe unit, which theoretically reduces the time to learn a language, and increases developer productivity. It is also development tasks can further increase productivity. For example, code can be compiled while being written, provic While most modern IDEs are graphical, IDEs in use before the advent of windowing systems (such as Microsoft perform various tasks (Turbo Pascal is a common example). This contrasts with software development using unrelat

History

IDEs initially became possible when developing via a console or terminal. Early systems could not support one, si entering programs with punched cards (or paper tape, etc.) before submitting them to a compiler. Dartmouth BASI IDE (and was also the first to be designed for use while sitting in front of a console or terminal). Its IDE (part command-based, and therefore did not look much like the menu-driven, graphical IDEs prevalent today. How compilation, debugging and execution in a manner consistent with a modern IDE.

Keyboard Maestro [2]

Maestro I is a product from Softlab Munich and was the world's first integrated developm I was installed for 22,000 programmers worldwide. Until 1989, 6,000 installations existed i was arguably the world leader in this field during the 1970s and 1980s. Today one of the Information Technology at Arlington.

One of the first IDEs with a plug-in concept was Softbench. In 1995 Computerwoche cor received by developers since it would fence in their creativity.

Topics

Visual programming

Visual programming is a usage scenario in which an IDE is generally required. Visual IDEs allow users to crea building blocks or code nodes to create flowcharts or structure diagrams that are then compiled or interpreted. Thes

This interface has been popularized with the Lego Mindstorms system, and is being actively pursued by a number those found at Mozilla. KTechlab supports flowcode and is a popular opensource IDE and Simulator for developing the power of distributed programming (cf. LabVIEW and EICASLAB software). An early visual programming sy used to develop real-time music performance software since the 1980s. Another early example was Prograph, a data programming environment "Grape" is used to program qfix robot kits.

This approach is also used in specialist software such as Openlab, where the end users want the flexibility of a full p with one.

An open source visual programming system is Mindscript, which has extended functionality for cryptology, database

Language support

Some IDEs support multiple languages, such as Eclipse or NetBeans, both based on Java, or MonoDevelop, based or

Support for alternative languages is often provided by plugins, allowing them to be installed on the same IDE a C/C++, Ada, Perl, Python, Ruby, and PHP, among other languages.

Attitudes across different computing platforms

Many Unix programmers argue that traditional command-line POSIX tools constitute an IDE,Template:Who environment. Many programmers still use makefiles and their derivatives. Also, many Unix programmers use Emac tools. Data Display Debugger is intended to be an advanced graphical front-end for many text-based debugger stanc

On the various Microsoft Windows platforms, command-line tools for development are seldom used. Accordingly, t has a different design commonly creating incompatibilities. Most major compiler vendors for Windows still provi C++, Platform SDK, Microsoft .NET Framework SDK, nmake utility), Embarcadero Technologies (bcc32 compiler,

Additionally, the free software GNU tools (gcc, gdb, GNU make) are available on many platforms, including Windo

IDEs have always been popular on the Apple Macintosh's Mac OS, dating back to Macintosh Programmer's Work mid-1980s. Currently Mac OS X programmers can choose between limited IDEs, including native IDEs like Xcode, Netbeans. ActiveState Komodo is a proprietary IDE supported on the Mac OS.

References

1. Dana Nourie (2005-03-24). "Getting Started with an Integrated Development Environment". Sun Microsystems, Retrieved 2008-09-09.
2. Image credit: Museum of Information Technology at Arlington (http://mit-a.com/fourphase.shtml)
3. "Interaktives Programmieren als Systems-Schlager" (http://www.computerwoche.de/heftarchiv/1975/47/1205421

GUI Builder

A **graphical user interface builder** (or **GUI builder**), also known as **GUI designer**, is a software develo to arrange widgets using a drag-and-drop WYSIWYG editor. Without a GUI builder, a GUI must be built by feedback until the program is run.

User interfaces are commonly programmed using an event-driven architecture, so GUI builders also simplify creat outgoing and incoming events that trigger the functions providing the application logic.

List of GUI builders

Programs

- AutoIt
- Axure RP
- Cocoa/OpenStep
  - Interface Builder
- Embedded Wizard a commercial development tool focussed on user interface applications for embedded systems.
- Fast, Light Toolkit (FLTK)
  - FLUID
- GNUstep
  - Gorm
- GEM
  - Resource construction set
  - Interface by Shift Computer
  - ORCS (Otto's RCS)
  - K-Resource
  - Resource Master
  - Annabel Junior
  - WERCS by HiSoft
- GTK+
  - Glade Interface Designer
  - Gazpacho
  - Gideon Designer
- GUI Builder (http://gui-builder.com)
- Intrinsics
- Justinmind Prototyper
  - Motif
    - Builder Xcessory (http://www.ics.com/products/motif/guibuilders/bxpro/index.html)
    - Easymotif
    - ixbuild
    - UIMX (http://www.ics.com/products/motif/guibuilders/uimx/index.html)
    - X-Designer
- LucidChart
- Object Pascal
  - fpGUI UI Designer (included with fpGUI Toolkit)

- OpenWindows
  - guide (GUI builder)
- Pencil Project
- Qt
  - Qt Designer (http://web.archive.org/web/20080512225850/http://trolltech.com/products/qt/features/tools/
- Scaleform
- Tk (framework)
  - GUI Builder (http://spectcl.sourceforge.net/ko3-guib-docs/komodo-doc-guibuilder.html)
  - ActiveState Komodo
  - Visual Tcl (http://vtcl.sourceforge.net/) (dead project)
  - PureTkGUI (http://puretkgui.sourceforge.net/)
- Wavemaker open source, browser-based development platform for Ajax development based on Dojo, Spring, Hib
- Windows Presentation Foundation
  - Microsoft Expression Blend
- wxWidgets
  - wxGlade
  - wxFormBuilder (http://wiki.wxformbuilder.org/)
  - wxDesigner (http://www.wxdesigner-software.de/)
- XForms (toolkit)
  - fdesign
- Crank Storyboard Suite
  - Storyboard Designer (http://www.cranksoftware.com/products/crank_storyboard_designer.php)

IDE Plugins

- NetBeans GUI design tool, formerly known as *Matisse (http://netbeans.org/features/java/swing.html)*.
- Visual Editor (http://www.eclipse.org/vep/) - A free (Eclipse Public License) plugin for Eclipse on MS Windows
- Jigloo (http://www.cloudgarden.com/jigloo/) - A *free for non-commercial use* plugin for Eclipse on MS Window
- WxSmith (http://wiki.codeblocks.org/index.php?title=WxSmith_plugin) - A Code::Blocks plug-in for RAD edit
- Himalia Guilder (http://www.himalia.net/) (Only for Visual Studio 2005; no release since December '06.)

List of development environments

IDEs with GUI builders

- ActiveState Komodo
- Adobe Flash Builder
- Anjuta
- Ares
- CodeGear RAD Studio (former Borland Development Studio)
- Clarion
- Code::Blocks (http://www.codeblocks.org/)
- Gambas
- Just BASIC/Liberty BASIC
- KDevelop
- Lazarus
- Microsoft Visual Studio
- MonoDevelop
- MSEide+MSEgui (http://sourceforge.net/projects/mseide-msegui/)

- NetBeans
- Qt Creator
- REALbasic
- SharpDevelop
- Softwell Maker
- WinDev
- wxDev-C++
- Oracle Application Express

Source Control

Revision control, also known as **version control** or **source control** (and an aspect of **software conf** management of changes to documents, programs, and other information stored as computer files. It is most comm team of people may change the same files. Changes are usually identified by a number or letter code, termed the "revision". For example, an initial set of files is "revision 1". When the first change is made, the resulting set is "rev with a timestamp and the person making the change. Revisions can be compared, restored, and with some types of f

Version control systems (VCSs – singular VCS) most commonly run as stand-alone applications, but revision contro such as word processors (e.g., Microsoft Word, OpenOffice.org Writer, KWord, Pages, etc.), spreadsheets (e.g., M Numbers, etc.), and in various content management systems (e.g., Drupal, Joomla, WordPress). Integrated revi packages such as MediaWiki, DokuWiki, TWiki etc. In wikis, revision control allows for the ability to revert a pa allowing editors to track each other's edits, correct mistakes, and defend public wikis against vandalism and spam.

Software tools for revision control are essential for the organization of multi-developer projects.[1]

Overview

Revision control developed from formalized processes based on tracking revisions of early blueprints or blueli returning to any earlier state of the design, for cases in which an engineering dead-end was reached in the deve software engineering, revision control is any practice that tracks and provides control over changes to source cod control software to maintain documentation and configuration files as well as source code. Also, version contro "contract redline" and "legal blackline" are some of the earliest forms of revision control,[*citation needed*] and sophistication. An entire industry has emerged to service the document revision control needs of business and technology employed in these circles is subtle, powerful, and innovative. The most sophisticated techniques are begi changes to CAD files (see product data management), supplanting the "manual" electronic implementation of traditi

As teams design, develop and deploy software, it is common for multiple versions of the same software to be dep developers to be working simultaneously on updates. Bugs or features of the software are often only present in c problems and the introduction of others as the program develops). Therefore, for the purposes of locating and fix retrieve and run different versions of the software to determine in which version(s) the problem occurs. It may als software concurrently (for instance, where one version has bugs fixed, but no new features (branch), while the othe (trunk).

At the simplest level, developers could simply retain multiple copies of the different versions of the program, and l large software projects. While this method can work, it is inefficient as many near-identical copies of the program h developers, and often leads to mistakes. Consequently, systems to automate some or all of the revision control proce

Moreover, in software development, legal and business practice and other environments, it has become increasing team, the members of which may be geographically dispersed and may pursue different and even contrary interests. changes to documents and code may be extremely helpful or even necessary in such situations.

Revision control may also track changes to configuration files, such as those typically stored in /etc or /usr/local/ easily track changes made and a way to roll back to earlier versions should the need arise.

Source-management models

Traditional revision control systems use a centralized model where all the revision control functions take place on a time, without some method of managing access the developers may end up overwriting each other's work. Central "source management models": file locking and version merging.

Atomic operations

Computer scientists speak of *atomic* operations if the system is left in a consistent state even if the operation is inte Commits are operations which tell the revision control system you want to make a group of changes you have been have atomic commits; notably, the widely-used CVS lacks this feature.

File locking

The simplest method of preventing "concurrent access" problems involves locking files so that only one developer at Once one developer "checks out" a file, others can read that file, but no one else may change that file until that deve

File locking has both merits and drawbacks. It can provide some protection against difficult merge conflicts when a of files). However, if the files are left exclusively locked for too long, other developers may be tempted to bypass th serious problems.

experiment

Version merging

Most version control systems allow multiple developers to edit the same file at the same time. The first develope system may provide facilities to merge further changes into the central repository, and preserve the changes from th

Merging two files can be a very delicate operation, and usually possible only if the data structure is simple, as in te image file at all. The second developer checking in code will need to take care with the merge, to make sure th introduce its own logic errors within the files. These problems limit the availability of automatic or semi-automa specific merge plugin is available for the file types.

The concept of a *reserved edit* can provide an optional means to explicitly lock a file for exclusive write access, even

Baselines, labels and tags

Most revision control tools will use only one of these similar terms (baseline, label, tag) to refer to the action of ic ("try it with baseline *X*"). Typically only one of the terms *baseline*, *label*, or *tag* is used in documentation or discussi

In most projects some snapshots are more significant than others, such as those used to indicate published releases,

When both the term *baseline* and either of *label* or *tag* are used together in the same context, *label* and *tag* usual record of the snapshot, and *baseline* indicates the increased significance of any given label or tag.

Most formal discussion of configuration management uses the term *baseline*.

Distributed revision control

Distributed revision control (DRCS) takes a peer-to-peer approach, as opposed to the client-server approach of centr synchronize, each peer's working copy of the codebase is a bona-fide repository.[2] Distributed revision control conc peer. This results in some important differences from a centralized system:

- No canonical, reference copy of the codebase exists by default; only working copies.
- Common operations (such as commits, viewing history, and reverting changes) are fast, because there is no need

Rather, communication is only necessary when pushing or pulling changes to or from other peers.

- Each working copy effectively functions as a remote backup of the codebase and of its change-history, providing

Integration

Some of the more advanced revision-control tools offer many other facilities, allowing deeper integration with other IDEs such as Oracle JDeveloper, IntelliJ IDEA, Eclipse and Visual Studio. NetBeans IDE and Xcode come with int

Common vocabulary

Terminology can vary from system to system, but some terms in common usage include:[4][5]

Baseline

An approved revision of a document or source file from which subsequent changes can be made. See baselines, la

Branch

A set of files under version control may be **branched** or **forked** at a point in time so that, from that time for different ways independently of each other.

Change

A **change** (or **diff**, or **delta**) represents a specific modification to a document under version control. The gran control systems.

Change list

On many version control systems with atomic multi-change commits, a **changelist**, **change set**, or **patch** id represent a sequential view of the source code, allowing the examination of source "as of" any particular changeli

Checkout

A **check-out** (or **co**) is the act of creating a local working copy from the repository. A user may specify a spec noun to describe the working copy.

Commit

A **commit** (**checkin**, **ci** or, more rarely, **install**, **submit** or **record**) is the action of writing or merging the 'commit' and 'checkin' can also used in noun form to describe the new revision that is created as a result of com

Conflict

A conflict occurs when different parties make changes to the same document, and the system is unable to reconc changes, or by selecting one change in favour of the other.

Delta compression

Most revision control software uses delta compression, which retains only the differences between successive vers versions of files.

Dynamic stream

A stream in which some or all file versions are mirrors of the parent stream's versions.

Export

exporting is the act of obtaining the files from the repository. It is similar to **checking-out** except that it cr in a working copy. This is often used prior to publishing the contents, for example.

Head

Also sometime called **tip**, this refers to the most recent commit.

Import

importing is the act of copying a local directory tree (that is not currently a working copy) into the repository

Label

See **tag**.

Mainline

Similar to *trunk*, but there can be a mainline for each branch.

Merge

A **merge** or **integration** is an operation in which two sets of changes are applied to a file or set of files. Some

- A user, working on a set of files, **updates** or **syncs** their working copy with changes made, and checked in
- A user tries to **check-in** files that have been updated by others since the files were **checked out**, and the (typically, after prompting the user if it should proceed with the automatic merge, and in some cases only do
- A set of files is **branched**, a problem that existed before the branching is fixed in one branch, and the fix is
- A **branch** is created, the code in the files is independently edited, and the updated branch is later incorpora

Promote

The act of copying file content from a less controlled location into a more controlled location. For example, from

Repository

The **repository** is where files' current and historical data are stored, often on a server. Sometimes also called a

Resolve

The act of user intervention to address a conflict between different changes to the same document.

Reverse integration

The process of merging different team branches into the main trunk of the versioning system.

Revision

Also **version**: A version is any change in form. In SVK, a Revision is the state at a point in time of the entire t

Ring

[*citation needed*] See **tag**.

Share

The act of making one file or folder available in multiple branches at the same time. When a shared file is chang

Stream

A container for branched files that has a known relationship to other such containers. Streams form a hierarchy; workflow rules, subscribers, etc.) from its parent stream.

Tag

A **tag** or **label** refers to an important snapshot in time, consistent across many files. These files at that point n number. See baselines, labels and tags.

Trunk

The unique line of development that is not a branch (sometimes also called Baseline or Mainline)

Update

An **update** (or **sync**) merges changes made in the repository (by other people, for example) into the local **wo**

Working copy

The **working copy** is the local copy of files from a repository, at a specific time or revision. All work done to name. Conceptually, it is a *sandbox*.

References

1. "Rapid Subversion Adoption Validates Enterprise Readiness and Challenges Traditional Software Configuration http://www.open.collab.net/news/press/2007/svn_momentum.html. Retrieved October 27, 2010. "Version mana critical component of any development environment."
2. Wheeler, David. "Comments on Open Source Software / Free Software (OSS/FS) Software Configuration Manag Retrieved May 8, 2007.
3. O'Sullivan, Bryan. "Distributed revision control with Mercurial". http://hgbook.red-bean.com/hgbook.html. Ret
4. Collins-Sussman, Ben; Fitzpatrick, B.W. and Pilato, C.M. (2004). *Version Control with Subversion*. O'Reilly. IS
5. Wingerd, Laura (2005). *Practical Perforce*. O'Reilly. ISBN 0-596-10185-6. http://safari.oreilly.com/0596101856.
6. Collins-Sussman, Ben; Brian W. Fitpatrick, and C. Michael Pilato. "Version Control with Subversion". http://sv Retrieved 8 June 2010. "The G stands for merGed, which means that the file had local changes to begin with, bu changes."
7. *Accurev Concepts Manual, Version 4.7*. Accurev, Inc.. July, 2008.

External links

- Eric Sink's Source Control HOWTO (http://www.ericsink.com/scm/source_control.html) A primer on the basic
- Visual Guide to Version Control (http://betterexplained.com/articles/a-visual-guide-to-version-control/)
- Better SCM Initiative: Comparison (http://better-scm.berlios.de/comparison/) A useful summary of different sy

Build Tools

Build automation is the act of scripting or automating a wide variety of tasks that software developers do in the

- compiling computer source code into binary code
- packaging binary code
- running tests
- deployment to production systems
- creating documentation and/or release notes

History

Historically, developers used build automation to call compilers and linkers from inside a build script versus attemp use the command line to pass a single source module to a compiler and then to a linker to create the final deploya code modules, in a particular order, using the command line process is not a reasonable solution. The make scripti written to call in a series, the needed compile and link steps to build a software application. GNU Make [1] also source code dependency management as well as incremental build processing. This was the beginning of Build Autc and linkers. As the build process grew more complex, developers began adding pre and post actions around the copying of deployable objects to a test location. The term "build automation" now includes managing the pre and pc

New breed of solutions

In recent years, build management solutions have provided even more relief when it comes to automating the bui perform more automated build and workflow processing. Some solutions focus on automating the pre and post step and post build script processing and drive down into streamlining the actual compile and linker calls without mu integration builds where frequent calls to the compile process are required and incremental build processing is neede

Advanced build automation

Advanced build automation offers remote agent processing for distributed builds and/or distributed processing. Tl and linkers can be served out to multiple locations for improving the speed of the build. This term is often confuse step in a process or workflow can be sent to a different machine for execution. For example, a post step to the machines. Distributed processing can send the different test scripts to different machines. Distributed processing is n maven script, break it up and send it to different machines for compiling and linking. The distributed build proc dependencies in order to send the different compile and link steps to different machines. A build automation sol distributed builds. Some build tools can discover these relationships programmatically (Rational ClearMake distril user-configured dependencies (Platform LSF lsmake[4]) Build automation that can sort out source code depend activities in a parallelized mode. This means that the compiler and linkers can be called in multi-threaded mode usi

Not all build automation tools can perform distributed builds. Most only provide distributed processing support. handle C or C++. Build automation solutions that support distributed processing are often make based and many c

An example of a distributed build solution is Xoreax's IncrediBuild[5] for the Microsoft Visual Studio platform or tl a product environment so that it can run successfully on a distributed platform—library locations, environment vari

Advantages

- Improve product quality
- Accelerate the compile and link processing

- Eliminate redundant tasks
- Minimize "bad builds"
- Eliminate dependencies on key personnel
- Have history of builds and releases in order to investigate issues
- Save time and money - because of the reasons listed above.[7]

Types

- **On-Demand automation** such as a user running a script at the command line
- **Scheduled automation** such as a continuous integration server running a nightly build
- **Triggered automation** such as a continuous integration server running a build on every commit to a versior

Makefile

One specific form of build automation is the automatic generation of Makefiles. This is accomplished by tools like

- GNU Automake
- CMake
- imake
- qmake
- nmake
- wmake
- Apache Ant
- Apache Maven
- OpenMake Meister

Requirements of a build system

Basic requirements:

1. Frequent or overnight builds to catch problems early.[8][9][10]
2. Support for Source Code Dependency Management
3. Incremental build processing
4. Reporting that traces source to binary matching
5. Build acceleration
6. Extraction and reporting on build compile and link usage

Optional requirements:[11]

1. Generate release notes and other documentation such as help pages
2. Build status reporting
3. Test pass or fail reporting
4. Summary of the features added/modified/deleted with each new build

References

1. http://www.gnu.org/software/make/
2. *Dr. Dobb's Distributed Loadbuilds*, http://www.ddj.com/architect/184405385, retrieved 2009-04-13
3. *Dr. Dobb's Take My Build, Please*, http://www.ddj.com/architect/184415472
4. *LSF User's Guide - Using lsmake*, http://www.lle.rochester.edu/pub/support/lsf/10-lsmake.html, retrieved 2009
5. *Distributed Visual Studio Builds*, http://www.xoreax.com/solutions_vs.htm, retrieved 2009-04-08
6. *CMake - Cross platform make*, http://www.cmake.org/, retrieved 2010-03-27
7. http://www.denverjug.org/meetings/files/200410_automation.pdf
8. http://freshmeat.net/articles/view/392/
9. http://www.ibm.com/developerworks/java/library/j-junitmail/
10. http://buildbot.net/trac

11. http://www.cmcrossroads.com/content/view/12525/120/

Notes

- Mike Clark: *Pragmatic Project Automation*, The Pragmatic Programmers ISBN 0-9745140-3-9

Software Documentation

Software documentation or **source code documentation** is written text that accompanies computer soft
different things to people in different roles.

Involvement of people in software life

Documentation is an important part of software engineering. Types of documentation include:

1. Requirements - Statements that identify attributes, capabilities, characteristics, or qualities of a system. This is
2. Architecture/Design - Overview of softwares. Includes relations to an environment and construction principles to
3. Technical - Documentation of code, algorithms, interfaces, and APIs.
4. End User - Manuals for the end-user, system administrators and support staff.
5. Marketing - How to market the product and analysis of the market demand.

Requirements documentation

Requirements documentation is the description of what a particular software does or shall do. It is used throughou
also used as an agreement or as the foundation for agreement on what the software shall do. Requirements are prod
end users, customers, product managers, project managers, sales, marketing, software architects, usability enginee
requirements documentation has many different purposes.

Requirements come in a variety of styles, notations and formality. Requirements can be goal-like (e.g., *distributed
clicking a configuration file and select the 'build' function*), and anything in between. They can be specified as stat
formulas, and as a combination of them all.

The variation and complexity of requirements documentation makes it a proven challenge. Requirements may be im
what kind of documentation is needed and how much can be left to the architecture and design documentation, ar
variety of people that shall read and use the documentation. Thus, requirements documentation is often incon
software changes become more difficult—and therefore more error prone (decreased software quality) and time-consu

The need for requirements documentation is typically related to the complexity of the product, the impact of the
complex or developed by many people (e.g., mobile phone software), requirements can help to better communicate
impact on human life (e.g., nuclear power systems, medical equipment), more formal requirements documentation
two (e.g., very small mobile phone applications developed specifically for a certain campaign) very little requirement
later built upon, requirements documentation is very helpful when managing the change of the software and verifyin

Traditionally, requirements are specified in requirements documents (e.g. using word processing applications an
changing nature of requirements documentation (and software documentation in general), database-centric systems

Architecture/Design documentation

Architecture documentation is a special breed of design document. In a way, architecture documents are third deriv
documents being first). Very little in the architecture documents is specific to the code itself. These documents d
particular routine exists in the form that it does, but instead merely lays out the general requirements that would m
short on details but thick on explanation. It may suggest approaches for lower level design, but leave the actual exp

Another breed of design docs is the comparison document, or trade study. This would often take the form of a *u*
alternate approaches. It could be at the user interface, code, design, or even architectural level. It will outline wha
pros and cons of each. A good trade study document is heavy on research, expresses its idea clearly (without relying
impartial. It should honestly and clearly explain the costs of whatever solution it offers as best. The objective
particular point of view. It is perfectly acceptable to state no conclusion, or to conclude that none of the alternative
be approached as a scientific endeavor, not as a marketing technique.

A very important part of the design document in enterprise software development is the Database Design Do
Elements. The DDD includes the formal information that the people who interact with the database need. The p
players within the scene. The potential users are:

- Database Designer
- Database Developer
- Database Administrator
- Application Designer
- Application Developer

When talking about Relational Database Systems, the document should include following parts:

- Entity - Relationship Schema, including following information and their clear definitions:
  - Entity Sets and their attributes

- Relationships and their attributes
- Candidate keys for each entity set
- Attribute and Tuple based constraints

- Relational Schema, including following information:
  - Tables, Attributes, and their properties
  - Views
  - Constraints such as primary keys, foreign keys,
  - Cardinality of referential constraints
  - Cascading Policy for referential constraints
  - Primary keys

It is very important to include all information that is to be used by all actors in the scene. It is also very important

Technical documentation

This is what most programmers mean when using the term *software documentation*. When creating software, code various aspects of its intended operation. It is important for the code documents to be thorough, but not so ve overview documentation are found specific to the software application or software product being documented by A also the end customers or clients using this software application. Today, we see lot of high end applications in security, industry automation and a variety of other domains. Technical documentation has become important wi may change over a period of time with architecture changes. Hence, technical documentation has gained lot of impo

Often, tools such as Doxygen, NDoc, javadoc, EiffelStudio, Sandcastle, ROBODoc, POD, TwinText, or Universal extract the comments and software contracts, where available, from the source code and create reference manuals in into a *reference guide* style, allowing a programmer to quickly look up an arbitrary function or class.

The idea of auto-generating documentation is attractive to programmers for various reasons. For example, bec comments), the programmer can write it while referring to the code, and use the same tools used to create the sou the documentation up-to-date.

Of course, a downside is that only programmers can edit this kind of documentation, and it depends on them to documents nightly). Some would characterize this as a pro rather than a con.

Donald Knuth has insisted on the fact that documentation can be a very difficult afterthought process and has ad the source code and extracted by automatic means.

Elucidative Programming is the result of practical applications of Literate Programming in real programming documentation be stored separately. This paradigm was inspired by the same experimental findings that produced create and access information that is not going to be part of the source file itself. Such annotations are usually p porting, where third party source code is analysed in a functional way. Annotations can therefore help the documentation system would hinder progress. Kelp (http://kelp.sf.net/) stores annotations in separate files, linking

User documentation

Unlike code documents, user documents are usually far more diverse with respect to the source code of the program,

In the case of a software library, the code documents and user documents could be effectively equivalent and are wo

Typically, the user documentation describes each feature of the program, and assists the user in realizing these fea troubleshooting assistance. It is very important for user documents to not be confusing, and for them to be up to d is very important for them to have a thorough index. Consistency and simplicity are also very valuable. User do software will do. API Writers are very well accomplished towards writing good user documents as they would be we See also Technical Writing.

There are three broad ways in which user documentation can be organized.

1. **Tutorial:** A tutorial approach is considered the most useful for a new user, in which they are guided through e
2. **Thematic:** A thematic approach, where chapters or sections concentrate on one particular area of interest, is o convey their ideas through a knowledge based article to facilitating the user needs. This approach is usually prac user population is largely correlated with the troubleshooting demands [2], [3].
3. **List or Reference:** The final type of organizing principle is one in which commands or tasks are simply listed This latter approach is of greater use to advanced users who know exactly what sort of information they are lool

A common complaint among users regarding software documentation is that only one of these three approaches w provided software documentation for personal computers to online help that give only reference information on co experienced users get the most out of a program is left to private publishers, who are often given significant assistan

Marketing documentation

For many applications it is necessary to have some promotional materials to encourage casual observers to spend three purposes:-

1. To excite the potential user about the product and instill in them a desire for becoming more involved with it.

2. To inform them about what exactly the product does, so that their expectations are in line with what they will l
3. To explain the position of this product with respect to other alternatives.

One good marketing technique is to provide clear and memorable *catch phrases* that exemplify the point we wish anything else provided by the manufacturer.

Notes

1. Woelz, Carlos. "The KDE Documentation Primer". http://i18n.kde.org/docs/doc-primer/index.html. Retrieved 1
2. Microsoft. "Knowledge Base Articles for Driver Development". http://www.microsoft.com/whdc/driver/kernel/k
3. Prekaski, Todd. "Building web and Adobe AIR applications from a shared Flex code base". http://www.adobe.c 2009.

External links

- kelp (http://kelp.sf.net/) - a source code annotation framework for architectural, design and technical document
- ISO documentation standards committee (http://www.hci.com.au/iso) - International Organization for Standard

Static Code Analysis

This is a list of tools for static code analysis.

Historical products

- Lint — The original static code analyzer of C code.

Open-source or Non-commercial products

Multi-language

- PMD Copy/Paste Detector (CPD) — PMDs duplicate code detection for (e.g.) Java, JSP, C, C++ and PHP co
- Sonar — A continuous inspection engine to manage the technical debt (unit tests, complexity, duplication, desig languages are Java, Flex, PHP, PL/SQL, Cobol and Visual Basic 6.
- Yasca — Yet Another Source Code Analyzer, a plugin-based framework for scanning arbitrary file types, with pl ColdFusion, COBOL, and other file types. It integrates with other scanners, including FindBugs, JLint, PMD, a

.NET (C#, VB.NET and all .NET compatible languages)

- FxCop — Free static analysis for Microsoft .NET programs that compile to CIL. Standalone and integrated in s
- Gendarme — Open-source (MIT License) equivalent to FxCop created by the Mono project. Extensible rule-bas those that contain code in ECMA CIL format.
- StyleCop — Analyzes C# source code to enforce a set of style and consistency rules. It can be run from inside o download from Microsoft.

ActionScript

- Apparat — A language manipulation and optimization framework consisting of intermediate representations for

C

- BLAST (Berkeley Lazy Abstraction Software verification Tool) — A software model checker for C programs bas
- Clang — A compiler that includes a static analyzer.
- Frama-C — A static analysis framework for C.
- Lint — The original static code analyzer for C.
- Sparse — A tool designed to find faults in the Linux kernel.
- Splint — An open source evolved version of Lint (for C).

C++

- cppcheck — Open-source tool that checks for several types of errors, including the use of STL.

Java

- Checkstyle — Besides some static code analysis, it can be used to show violations of a configured coding standar

- FindBugs — An open-source static bytecode analyzer for Java (based on Jakarta BCEL) from the University of
- Hammurapi — (Free for non-commercial use only) versatile code review solution.
- PMD — A static ruleset based Java source code analyzer that identifies potential problems.
- Soot — A language manipulation and optimization framework consisting of intermediate languages for Java.
- Squale — A platform to manage software quality (also available for other languages, using commercial analysis t

JavaScript

- Closure Compiler — JavaScript optimizer that rewrites JavaScript code to make it faster and more compact. It
- JSLint — JavaScript syntax checker and validator.

Objective-C

- Clang — The free Clang project includes a static analyzer. As of version 3.2, this analyzer is included in Xcode.[
- Oclint — OCLint is a static code analysis tool for improving quality and reducing defects by inspecting C, C++
- Faux Pas — Faux Pas inspects your iOS or Mac app's Xcode project and warns about possible bugs, as well as
- Facebook Infer — Open Source Tool by Facebook to detect bugs in Android and iOS apps [4]
- Sonar for Objective C — Open Source Sonar plugin for xcode. [5]
- Sonar for Objective C (Commercial version) — Paid Sonar plugin for xcode .[6]

Commercial products

Multi-language

- Axivion Bauhaus Suite — A tool for C, C++, C#, Java and Ada code that comprises various analyses such as
- Black Duck Suite — Analyze the composition of software source code and binary files, search for reusable code,
  obligations associated with mixed-origin code, and monitor related security vulnerabilities.
- CAST Application Intelligence Platform — Detailed, audience-specific dashboards to measure quality and produ
  C/C++, Struts, Spring, Hibernate and all major databases.
- Checkmarx CxSuite — Source code analysis tool which identifies application security vulnerabilities in the follow
  VB.NET, APEX, Ruby, Javascript, ASP, Perl, Android, Objective C, PL/SQL, HTML5, Python and Groovy.
- Coverity Static Analysis (formerly Coverity Prevent) — Identifies security vulnerabilities and code defects in C,
  Analysis and Architecture Analysis.
- DMS Software Reengineering Toolkit — Supports custom analysis of C, C++, C#, Java, COBOL, PHP, Visual
  dead code analysis, and style checking.
- Compuware DevEnterprise — Analysis of COBOL, PL/I, JCL, CICS, DB2, IMS and others.
- Fortify — Helps developers identify software security vulnerabilities in C/C++, .NET, Java, JSP, ASP.NET, C
  T-SQL, python and COBOL as well as configuration files.
- GrammaTech CodeSonar — Analyzes C,C++.
- Imagix 4D — Identifies problems in variable usage, task interaction and concurrency, particularly in embedded
  and documenting C, C++ and Java software.
- Intel - Intel Parallel Studio XE: Contains **Static Security Analysis** (SSA) feature supports C/C++ and Fo
- JustCode — Code analysis and refactoring productivity tool for JavaScript, C#, Visual Basic.NET, and ASP.NI
- Klocwork Insight — Provides security vulnerability and defect detection as well as architectural and build-over-
- Kiuwan (https://www.kiuwan.com) – Software Analytics end-to-end platform for static code analysis, defect det
  cycle and application governance features. It supports over 25 languages, including Objective-C, Java, JSP, Java
  SQL, SQL, Visual Basic, Visual Basic .NET, Android (operating system).
- Lattix, Inc. LDM — Architecture and dependency analysis tool for Ada, C/C++, Java, .NET software systems.
- LDRA Testbed — A software analysis and testing tool suite for C, C++, Ada83, Ada95 and Assembler (Intel, F
- Micro Focus (formerly Relativity Technologies) Modernization Workbench — Parsers included for COBOL (mul
  Natural (inc. ADABAS), Java, Visual Basic, RPG, C & C++ and other legacy languages; Extensible SDK to su
  Function Points), Business Rule Mining, Componentisation and SOA Analysis. Rich ad hoc diagramming, AST

- Ounce Labs (from 2010 IBM Rational Appscan Source) — Automated source code analysis that enables organiza languages including Java, JSP, C/C++, C#, ASP.NET and VB.Net.
- Parasoft — Analyzes Java (Jtest), JSP, C, C++ (C++test), .NET (C#, ASP.NET, VB.NET, etc.) using .TEST configuration files for security[7], compliance[8], and defect prevention.
- Polyspace — Uses abstract interpretation to detect and prove the absence of certain run-time errors in source co
- Rational Asset Analyzer (IBM); Supports COBOL(multiple variants), PL/I, Java
- Rational Software Analyzer — Supports Java, C/C++ (and others available through extensions)
- Security Reviewer (http://www.securityreviewer.com) 1500+ Rules with up to 12 variants each, specialized per l languages: ABAP, Android Mobile, ASP, ASPX, C, C++, CSS, Objective-C, COBOL, C#, Forms, HTML5, Jav PL/SQL and T-SQL and TeradataSQL, VB.net, Visual Basic 6, Windows Mobile, XML, XPath. NIST and CVE Practices. SQALE dashboard.
- SofCheck Inspector — Provides static detection of logic errors, race conditions, and redundant code for Java and itself.
- SourceMeter — A platform-independent, command-line static source code analyzer for Java, C/C++, RPG IV (
- Sotoarc/Sotograph — Architecture and quality in-depth analysis and monitoring for Java, C#, C and C++
- Syhunt Sandcat — Detects security flaws in PHP, Classic ASP and ASP.NET web applications.
- Understand — Analyzes C,C++, Java, Ada, Fortran, Jovial, Delphi, VHDL, HTML, CSS, PHP, and JavaScript
- Veracode — Finds security flaws in application binaries and bytecode without requiring source. Supported langu Java, JSP, ColdFusion, and PHP.
- Visual Studio Team System — Analyzes C++,C# source codes. only available in team suite and development e

.NET

Products covering multiple .NET languages.

- CodeIt.Right — Combines Static Code Analysis and automatic Refactoring to best practices which allows autom VB.NET.
- CodeRush — A plugin for Visual Studio, it addresses a multitude of short comings with the popular IDE. Includ analysis.
- JustCode — Add-on for Visual Studio 2005/2008/2010 for real-time, solution-wide code analysis for C#, VB.NE
- NDepend — Simplifies managing a complex .NET code base by analyzing and visualizing code dependencies, by different versions of the code. Integrates into Visual Studio.
- ReSharper — Add-on for Visual Studio 2003/2005/2008/2010 from the creators of IntelliJ IDEA, which also pro
- Kalistick — Mixing from the Cloud: static code analysis with best practice tips and collaborative tools for Agile

Ada

- Ada-ASSURED — A tool that offers coding style checks, standards enforcement and pretty printing features.
- AdaCore CodePeer — Automated code review and bug finder for Ada programs that uses control-flow, data-flow
- LDRA Testbed — A software analysis and testing tool suite for Ada83/95.
- SofCheck Inspector — Provides static detection of logic errors, race conditions, and redundant code for Ada. Pro

C / C++

- FlexeLint — A multiplatform version of PC-Lint.
- Green Hills Software DoubleCheck — A software analysis tool for C/C++.
- Intel - Intel Parallel Studio XE: Contains **Static Security Analysis** (SSA) feature
- LDRA Testbed — A software analysis and testing tool suite for C/C++.
- Monoidics INFER — A sound tool for C/C++ based on Separation Logic.
- PC-Lint — A software analysis tool for C/C++.
- PVS-Studio — A software analysis tool for C,C++,C++11,C++/CX.
- QA-C (and QA-C++) — Deep static analysis of C/C++ for quality assurance and guideline enforcement.
- Red Lizard's Goanna — Static analysis for C/C++ in Eclipse and Visual Studio.

- SourceMeter — A platform-independent, command-line static source code analyzer for Java, C/C++, RPG IV (

Java

- Jtest — Testing and static code analysis product by Parasoft.
- LDRA Testbed — A software analysis and testing tool suite for Java.
- SemmleCode — Object oriented code queries for static program analysis.
- SonarJ — Monitors conformance of code to intended architecture, also computes a wide range of software metric
- Kalistick — A Cloud-based platform to manage and optimize code quality for Agile teams with DevOps spirit
- SourceMeter — A platform-independent, command-line static source code analyzer for Java, C/C++, RPG IV (

Formal methods tools

Tools that use a formal methods approach to static analysis (e.g., using static program assertions):

- ESC/Java and ESC/Java2 — Based on Java Modeling Language, an enriched version of Java.
- Polyspace — Uses abstract interpretation (a formal methods based technique[10]) to detect and prove the absen
- SofCheck Inspector — Statically determines and documents pre- and postconditions for Java methods; statically
- SPARK Toolset including the SPARK Examiner — Based on the SPARK programming language, a subset of A

References

1. "Static Analysis in Xcode". Apple. http://developer.apple.com/mac/library/featuredarticles/StaticAnalysis/inde
2. "Static Analysis". Oclint. http://oclint.org/. Retrieved 2015-09-06.
3. "Static Analysis". Faux Pas. http://fauxpasapp.com/. Retrieved 2015-09-06.
4. "Static Analysis". Facebook. http://fbinfer.com/. Retrieved 2015-09-06.
5. "Static Analysis in Sonar". Boto. https://github.com/octo-technology/sonar-objective-c. Retrieved 2015-09-06.
6. "Static Analysis". Boto. http://www.sonarsource.com/products/plugins/languages/objective-c/. Retrieved 2015-(
7. Parasoft Application Security Solution (http://www.parasoft.com/jsp/solutions/application_security_solution.js
8. Parasoft Compliance Solution (http://www.parasoft.com/jsp/solutions/compliance.jsp?itemId=339)
9. SourceMeter (https://www.sourcemeter.com)
10. Cousot, Patrick (2007). "The Role of Abstract Interpretation in Formal Methods". IEEE International Conferenc http://ieeexplore.ieee.org/Xplore/login.jsp?url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F4343908%2F434 Retrieved 2010-11-08.

External links

- Java Static Checkers (http://www.dmoz.org//Computers/Programming/Languages/Java/Development_Tools/
- List of Java static code analysis plugins for Eclipse (http://www.eclipsepluginscentral.com/Web_Links-index-req
- List of static source code analysis tools for C (http://www.spinroot.com/static/)
- List of Static Source Code Analysis Tools (https://www.cert.org/secure-coding/tools.html) at CERT
- SAMATE-Source Code Security Analyzers (http://samate.nist.gov/index.php/Source_Code_Security_Analyzer
- SATE - Static Analysis Tool Exposition (http://samate.nist.gov/SATE.html)
- "A Comparison of Bug Finding Tools for Java" (http://www.cs.umd.edu/~jfoster/papers/issre04.pdf), by Nick I Compares Bandera, ESC/Java 2, FindBugs, JLint, and PMD.
- "Mini-review of Java Bug Finders" (http://www.oreillynet.com/digitalmedia/blog/2004/03/minireview_of_java
- Parallel Lint (http://www.ddj.com/218000153), by Andrey Karpov
- Integrate static analysis into a software development process (http://www.embedded.com/shared/printableArtic static analysis into a software development process
- Static Analysis Tools for C/C++ - Polyspace (http://www.mathworks.com/products/polyspace/index.html)
- Errors detected in Open Source projects by the PVS-Studio developers through static analysis (http://www.viva

Profiling

In software engineering, **program profiling**, **software profiling** or simply **profiling**, a form of dynamic prog a program's behavior using information gathered as the program executes. The usual purpose of this analysis is overall speed, decrease its memory requirement or sometimes both.

- A **(code) profiler** is a **performance analysis** tool that, most commonly, measures only the frequency and
  (e.g. memory profilers) in addition to more comprehensive profilers, capable of gathering extensive performance
- An instruction set simulator which is also — by necessity — a profiler, can measure the totality of a program's

Gathering program events

Profilers use a wide variety of techniques to collect data, including hardware interrupts, code instrumentation, counters. The usage of profilers is 'called out' in the performance engineering process.

Use of profilers

Program analysis tools are extremely important for understanding program behavior. Computer architects need such Software writers need tools to analyze their programs and identify critical sections of code. Compiler writers often u prediction algorithm is performing... (ATOM, PLDI, '94)

The output of a profiler may be:-

- A statistical *summary* of the events observed (a **profile**)

  Summary profile information is often shown annotated against the source code statements where the events occu program.

```
/* ----------- source------------------------ count */
0001            IF X = "A"                  0055
0002              THEN DO
0003                ADD 1 to XCOUNT           0032
0004              ELSE
0005            IF X = "B"                  0055
```

- A stream of recorded events (a **trace**)

  For sequential programs, a summary profile is usually sufficient, but performance problems in parallel programs time relationship of events, thus requiring a full trace to get an understanding of what is happening.
  The size of a (full) trace is linear to the program's instruction path length, making it somewhat impractical. A t terminated at another point to limit the output.

- An ongoing interaction with the hypervisor (continuous or periodic monitoring via on-screen display for instance

  This provides the opportunity to switch a trace on or off at any desired point during execution in addition to vi provides the opportunity to suspend asynchronous processes at critical points to examine interactions with othe

History

Performance analysis tools existed on IBM/360 and IBM/370 platforms from the early 1970s, usually based on ti timer intervals to detect "hot spots" in executing code. This was an early example of sampling (see below). In performance monitoring features.

Profiler-driven program analysis on Unix dates back to at least 1979, when Unix systems included a basic tool "pro used. In 1982, gprof extended the concept to a complete call graph analysis [1]

In 1994, Amitabh Srivastava and Alan Eustace of Digital Equipment Corporation published a paper describing profiler. That is, at compile time, it inserts code into the program to be analyzed. That inserted code outputs an known as "instrumentation".

In 2004, both the gprof and ATOM papers appeared on the list of the 50 most influential PLDI papers of all time.[3

Profiler types based on output

Flat profiler

Flat profilers compute the average call times, from the calls, and do not break down the call times based on the call

Call-graph profiler

Call graph profilers show the call times, and frequencies of the functions, and also the call-chains involved based on

Methods of data gathering

Event-based profilers

The programming languages listed here have event-based profilers:

- Java: the JVMTI (JVM Tools Interface) API, formerly JVMPI (JVM Profiling Interface), provides hooks to pro leave.
- .NET: Can attach a profiling agent as a COM server to the CLR. Like Java, the runtime then provides various leave, object creation, etc. Particularly powerful in that the profiling agent can rewrite the target application's b
- Python: Python profiling includes the profile module, hotshot (which is call-graph based), and using the 'sys.setp python_{call,return,exception}.
- Ruby: Ruby also uses a similar interface like Python for profiling. Flat-profiler in profile.rb, module, and ruby-pr

Statistical profilers

Some profilers operate by sampling. A sampling profiler probes the target program's program counter at regular int less numerically accurate and specific, but allow the target program to run at near full speed.

The resulting data are not exact, but a statistical approximation. *The actual amount of error is usually more than the expected error in it is the square-root of n sampling periods.* [4]

In practice, sampling profilers can often provide a more accurate picture of the target program's execution than ot thus don't have as many side effects (such as on memory caches or instruction decoding pipelines). Also since the would otherwise be hidden. They are also relatively immune to over-evaluating the cost of small, frequently calle spent in user mode versus interruptible kernel mode such as system call processing.

Still, kernel code to handle the interrupts entails a minor loss of CPU cycles, diverted cache usage, and is unable t (microsecond-range activity).

Dedicated hardware can go beyond this: some recent MIPS processors JTAG interface have a PCSAMPLE register,

Some of the most commonly used statistical profilers are AMD CodeAnalyst, Apple Inc. Shark, gprof, Intel VTune a

Instrumenting profilers

Some profilers **instrument** the target program with additional instructions to collect the required information.

Instrumenting the program can cause changes in the performance of the program, potentially causing inaccurate res program execution, typically always slowing it. However, instrumentation can be very specific and be carefully con depends on the placement of instrumentation points and the mechanism used to capture the trace. Hardware suppo on just one machine instruction. The impact of instrumentation can often be deducted (i.e. eliminated by subtractio

gprof is an example of a profiler that uses both instrumentation and sampling. Instrumentation is used to gather ca sampling.

Instrumentation

- **Manual**: Performed by the programmer, e.g. by adding instructions to explicitly calculate runtimes, simply cou Response Measurement standard.
- **Automatic source level**: instrumentation added to the source code by an automatic tool according to an ins
- **Compiler assisted**: Example: "gcc -pg ..." for gprof, "quantify g++ ..." for Quantify
- **Binary translation**: The tool adds instrumentation to a compiled binary. Example: ATOM
- **Runtime instrumentation**: Directly before execution the code is instrumented. The program run is fully su
- **Runtime injection**: More lightweight than runtime instrumentation. Code is modified at runtime to have jur

Interpreter instrumentation

- **Interpreter debug** options can enable the collection of performance metrics as the interpreter encounters eac three examples that usually have complete control over execution of the target code, thus enabling extremely co

Hypervisor/Simulator

- **Hypervisor**: Data are collected by running the (usually) unmodified program under a hypervisor. Example: SI
- **Simulator** and **Hypervisor**: Data collected interactively and selectively by running the unmodified program Interactive test/debug) and IBM OLIVER (CICS interactive test/debug).

References

1. *gprof: a Call Graph Execution Profiler* (http://docs.freebsd.org/44doc/psd/18.gprof/paper.pdf)
2. Atom: A system for building customized program analysis tools, Amitabh Srivastava and Alan Eustace, 1994 (ht
(download (http://www.ece.cmu.edu/~ece548/tools/atom/man/wrl_94_2.pdf))
3. 20 Years of PLDI (1979 - 1999): A Selection, Kathryn S. McKinley, Editor (http://www.cs.utexas.edu/users/mc
4. Statistical Inaccuracy of gprof Output (http://lgl.epfl.ch/teaching/case_tools/doc/gprof/gprof_12.html)

- Dunlavey, "Performance tuning with instruction-level cost derived from call-stack sampling", ACM SIGPLAN N
- Dunlavey, "Performance Tuning: Slugging It Out!", Dr. Dobb's Journal, Vol 18, #12, November 1993, pp 18–26

External links

- Article "Need for speed — Eliminating performance bottlenecks (http://www.ibm.com/developerworks/rational/ applications using IBM Rational Application Developer.
- Profiling Runtime Generated and Interpreted Code using the VTune™ Performance Analyzer (http://software.i ng.pdf)

Code Coverage

Code coverage is a measure used in software testing. It describes the degree to which the source code of a progra and is therefore a form of white box testing.[1] In time, the use of code coverage has been extended to the field of di hardware description languages (HDLs).

Code coverage was among the first methods invented for systematic software testing. The first published reference w

Code coverage is one consideration in the safety certification of avionics equipment. The standard by which avio documented in DO-178B.[2]

Coverage criteria

To measure how well the program is exercised by a test suite, one or more *coverage criteria* are used.

Basic coverage criteria

There are a number of coverage criteria, the main ones being:[3]

- **Function coverage** - Has each function (or subroutine) in the program been called?
- **Statement coverage** - Has each node in the program been executed?
- **Decision coverage** (not the same as **branch coverage** - see Position Paper CAST10.[4]) - Has every edge each branch of each control structure (such as in IF and CASE statements) been met as well as not met?
- **Condition coverage** (or predicate coverage) - Has each boolean sub-expression evaluated both to true and fa
- **Condition/decision coverage** - Both decision and condition coverage should be satisfied.

For example, consider the following C++ function:

```
int foo(int x, int y)
{
    int z = 0;
    if ((x>0) && (y>0)) {
        z = x;
    }
    return z;
}
```

Assume this function is a part of some bigger program and this program was run with some test suite.

- If during this execution function 'foo' was called at least once, then *function coverage* for this function is satisfie
- *Statement coverage* for this function will be satisfied if it was called e.g. as foo(1,1), as in this case, every line in
- Tests calling foo(1,1) and foo(1,0) will satisfy *decision coverage*, as in the first case the if condition is satisfied ar
- *Condition coverage* can be satisfied with tests that call foo(1,1), foo(1,0) and foo(0,0). These are necessary as in evaluates false. At the same time, the first case makes (y>0) true while the second and third make it false.

In languages, like Pascal, where standard boolean operations are not short circuited, condition coverage does not r
fragment of code:

```
if a and b then
```

Condition coverage can be satisfied by two tests:

- a=true, b=false
- a=false, b=true

However, this set of tests does not satisfy decision coverage as in neither case will the if condition be met.

Fault injection may be necessary to ensure that all conditions and branches of exception handling code have adequa

Modified condition/decision coverage

For safety-critical applications (e.g. for avionics software) it is often required that **modified condition/c**
condition/decision criteria with requirements that each condition should affect the decision outcome independently.

```
if (a or b) and c then
```

The condition/decision criteria will be satisfied by the following set of tests:

- a=true, b=true, c=true
- a=false, b=false, c=false

However, the above tests set will not satisfy modified condition/decision coverage, since in the first test, the valu
output. So, the following test set is needed to satisfy MC/DC:

- a=**false**, b=**false**, c=true
- a=**true**, b=false, c=**true**
- a=false, b=**true**, c=**true**
- a=true, b=true, c=**false**

The bold values influence the output, each variable must be present as an influencing value at least once with false a

Multiple condition coverage

This criteria requires that all combinations of conditions inside each decision are tested. For example, the code fragn

- a=false, b=false, c=false
- a=false, b=false, c=true
- a=false, b=true, c=false
- a=false, b=true, c=true
- a=true, b=false, c=false
- a=true, b=false, c=true
- a=true, b=true, c=false
- a=true, b=true, c=true

Other coverage criteria

There are further coverage criteria, which are used less often:

- **Linear Code Sequence and Jump (LCSAJ) coverage** - has every LCSAJ been executed?
- **JJ-Path coverage** - have all jump to jump paths [5] (aka LCSAJs) been executed?
- **Path coverage** - Has every possible route through a given part of the code been executed?
- **Entry/exit coverage** - Has every possible call and return of the function been executed?
- **Loop coverage** - Has every possible loop been executed zero times, once, and more than once?

Safety-critical applications are often required to demonstrate that testing achieves 100% of some form of code covera

Some of the coverage criteria above are connected. For instance, path coverage implies decision, statement and ent
every statement is part of a branch.

Full path coverage, of the type described above, is usually impractical or impossible. Any module with a succession can result in an infinite number of paths. Many paths may also be infeasible, in that there is no input to the p

However, a general-purpose algorithm for identifying infeasible paths has been proven to be impossible (such an practical path coverage testing instead attempt to identify classes of code paths that differ only in the number o cover all the path classes.

In practice

The target software is built with special options or libraries and/or run under a special environment such that every to the function points in the source code. This process allows developers and quality assurance personnel to look conditions (error handling and the like) and helps reassure test engineers that the most important conditions (func see what areas of code have not been exercised and the tests are updated to include these areas as necessary. Combi yet manageable, set of regression tests.

In implementing code coverage policies within a software development environment one must consider the following:

- What are coverage requirements for the end product certification and if so what level of code coverage is require Branch/Decision, Modified Condition/Decision Coverage(MC/DC), LCSAJ (Linear Code Sequence and Jump)
- Will code coverage be measured against tests that verify requirements levied on the system under test (DO-178E
- Is the object code generated directly traceable to source code statements? Certain certifications, (ie. DO-178B L "Then, additional verification should be performed on the object code to establish the correctness of such generat

Test engineers can look at code coverage test results to help them devise test cases and input or configuration sets forms of code coverage used by testers are statement (or line) coverage and path (or edge) coverage. Line coverage code were executed to complete the test. Edge coverage reports which branches or code decision points were execut as a percentage. The meaning of this depends on what form(s) of code coverage have been used, as 67% path covera

Generally, code coverage tools and libraries exact a performance and/or memory or other resource cost which is u used in the lab. As one might expect, there are classes of software that cannot be feasibly subjected to these cov through analysis rather than direct testing.

There are also some sorts of defects which are affected by such tools. In particular, some race conditions or simi coverage environments; and conversely, some of these defects may become easier to find as a result of the additional

Software tools

Tools for C / C++

- VectorCAST
- Parasoft C++test
- Cantata++
- Insure++
- IBM Rational Pure Coverage
- Tessy
- Testwell CTC++
- Trucov
- CodeScroll

Tools for C# .NET

- Parasoft dotTEST
- NCover
- Testwell CTC++ (with C# add on)

Tools for Java

- Parasoft Jtest
- Clover
- Cobertura
- Structure 101
- EMMA
- Serenity
- Testwell CTC++ (with Java and Android add on)

Tools for PHP

- PHPUnit, also need Xdebug to make coverage reports

Hardware tools

- Aldec
- Atrenta
- Cadence Design Systems
- JEDA Technologies
- Mentor Graphics
- Nusym Technology
- Simucad Design Automation
- Synopsys

Notes

1. Kolawa, Adam; Huizinga, Dorota (2007). *Automated Defect Prevention: Best Practices in Software Managemen* http://www.wiley.com/WileyCDA/WileyTitle/productCd-0470042125.html.
2. RTCA/DO-178(b), *Software Considerations in Airborne Systems and Equipment Certification, Radio Technical*
3. Glenford J. Myers (2004). *The Art of Software Testing, 2nd edition*. Wiley. ISBN 0471469122.
4. [8] (http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/cast_papers/media/cast-10.pdf)
5. M. R. Woodward, M. A. Hennell, "On the relationship between two control-flow coverage criteria: all JJ-paths a 440
6. Dorf, Richard C.: *Computers, Software Engineering, and Digital Devices*, Chapter 12, pg. 15. CRC Press, 2006. s.google.com/books?id=jykvlTCoksMC&pg=PT386&lpg=PT386&dq=%22infeasible+path%22+%22halting+pro ZfGa_IxYlfSNA&hl=en&sa=X&oi=book_result&resnum=1&ct=result)
7. Software Considerations in Airborne System and Equipment Certification-RTCA/DO-178B, RTCA Inc., Washir

External links

- Branch Coverage for Arbitrary Languages Made Easy (http://www.semdesigns.com/Company/Publications/Tes
- Code Coverage Analysis (http://www.bullseye.com/coverage.html) by Steve Cornett
- Code Coverage Introduction (http://www.javaranch.com/newsletter/200401/IntroToCodeCoverage.html)
- Comprehensive paper on Code Coverage & tools selection (http://archive.is/20121127093400/qualinfra.blogspot. Jayachandran
- Development Tools (Java) (http://www.dmoz.org//Computers/Programming/Languages/Java/Development_T
- Development Tools (General) (http://www.dmoz.org//Computers/Programming/Software_Testing/Products_a
- Systematic mistake analysis of digital computer programs (http://doi.acm.org/10.1145/366246.366248)
- FAA CAST Position Papers [15] (http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/ca

Project Management

Project management software is a term covering many types of software, including estimation and planning collaboration software, communication, quality management and documentation or administration systems, which a

Tasks or activities of project management software

Scheduling

One of the most common purposes is to schedule a series of events or tasks and the complexity of the schedule ca challenges include:

- Events which depend on one another in different ways or dependencies
- Scheduling people to work on, and resources required by, the various tasks, commonly termed resource schedulin
- Dealing with uncertainties in the estimates of the duration of each task

Providing information

Project planning software can be expected to provide information to various people or stakeholders, and can be u project(s). Typical requirements might include:

- Tasks lists for people, and allocation schedules for resources
- Overview information on how long tasks will take to complete
- Early warning of any risks to the project
- Information on workload, for planning holidays
- Evidence
- Historical information on how projects have progressed, and in particular, how actual and planned performance a
- Optimum utilization of available resource

Approaches to project management software

Desktop

Project management software can be implemented as a program that runs on the desktop of each user. This typicall

Desktop applications typically store their data in a file, although some have the ability to collaborate with other us based project plan can be shared between users if it's on a networked drive and only one user accesses it at a time.

Desktop applications *can* be written to run in a heterogeneous environment of multiple operating systems, although

Web-based

Project management software can be implemented as a Web application, accessed through an intranet, or an extran

This has all the usual advantages and disadvantages of web applications:

- Can be accessed from any type of computer without installing software on user's computer
- Ease of access-control
- Naturally multi-user
- Only one software version and installation to maintain
- Centralized data repository
- Typically slower to respond than desktop applications
- Project information not available when the user (or server) is offline
- Some solutions allow the user to go offline with a copy of the data

Personal

A personal project management application is one used at home, typically to manage lifestyle or home projects. Tl project management software typically involves simpler interfaces. See also *non-specialised tools* below.

Single user

A single-user system is programmed with the assumption that only one person will ever need to edit the project pl few people are involved in top-down project planning. Desktop applications generally fall into this category.

Collaborative

A collaborative system is designed to support multiple users modifying different sections of the plan at once; for ex those estimates get integrated into the overall plan. Web-based tools, including extranets, generally fall into this cat has live Internet access. To address this limitation, some software tools using client–server architecture provide a ri task information to other project team members through a central server when users connect periodically to the net others' as read only) to work on them while not on the network. When reconnecting to the database, all changes are

Integrated

An integrated system combines project management or project planning, with many other aspects of company lif project, the list of project customers becomes a customer relationship management module, and each person o functionality associated with their projects.

Similarly, specialised tools like SourceForge integrate project management software with source control (CVS) softw integrated into the same system.

Non-specialised tools

While specialised software may be common, and heavily promoted by each vendor, there are a vast range of other s

- Calendaring software can often handle scheduling as easily as dedicated software
- Spreadsheets are very versatile, and can be used to calculate things not anticipated by the designers.

Criticisms of project management software

The following may apply in general, or to specific products, or to some specific functions within products.

- May not be derived from a sound project management method. For example, displaying the Gantt chart view by rather than identifying objectives, deliverables and the imposed logical progress of events (dig the trench first to
- May be inconsistent with the type of project management method. For example, traditional (e.g. Waterfall) vs. a
- Focuses primarily on the planning phase and does not offer enough functionality for project tracking, control and the first paper print-out of a project plan, which is simply a snapshot at one moment in time. The plan is dynam tasks that are completed early, late, re-sequenced, etc. Good management software should not only facilitate this changes.
- Does not make a clear distinction between the planning phase and post planning phase, leading to user confusion example, shortening the duration of a task when an additional human resource is assigned to it while the project
- Offer complicated features to meet the needs of project management or project scheduling professionals, which m features may be so complicated as to be of no use to anyone. Complex task prioritization and resource leveling a and overallocation is often more simply resolved manually.
- Some people may achieve better results using simpler technique, (e.g. pen and paper), yet feel pressured into usi www.edwardtufte.com/bboard/q-and-a-fetch-msg?msg_id=00008c&topic_id=1&topic=)).
- Similar to PowerPoint, project management software might shield the manager from important interpersonal cor
- New types of software are challenging the traditional definition of Project Management. Frequently, users of pro For instance, managing the ongoing marketing for an already-released product is not a "project" in the traditiona resources working on something with a discrete beginning/end. Groupware applications now add "project manag project management. Classically-trained Project Managers may argue whether this is "sound project managemen de-facto definition of the term Project Management may change.
- When there are multiple larger projects, project management software can be very useful. Nevertheless, one shou is involved, as management software incurs a larger time-overhead than is worthwhile.

Books

- Eric Uyttewaal: *Dynamic Scheduling With Microsoft(r) Project 2000: The Book By and For Professionals*, ISBN
- George Suhanic: *Computer-Aided Project Management*, ISBN 0-19-511591-0
- Richard E. Westney: *Computerized Management of Multiple Small Projects*, ISBN 0-8247-8645-9

Continuous Integration

In software engineering, **continuous integration** (**CI**) implements *continuous* processes of applying quality con aims to improve the quality of software, and to reduce the time taken to deliver it, by replacing the traditional prac

Theory

When embarking on a change, a developer takes a copy of the current code base on which to work. As other deve ceases to reflect the repository code. When developers submit code to the repository they must first update their c The more changes the repository contains, the more work developers must do before submitting their own changes.

Eventually, the repository may become so different from the developers' baselines that they enter what is sometimes the time it took to make their original changes. In a worst-case scenario, developers may have to discard their chang

Continuous integration involves integrating early and often, so as to avoid the pitfalls of "integration hell". The prac

The rest of this article discusses best practice in how to achieve continuous integration, and how to automate this p

Recommended practices

Continuous integration - as the practice of frequently integrating one's new or changed code with the existing code

remains between commit and build, and such that no errors can arise without developers noticing them and correc every commit to a repository, rather than a periodically scheduled build. The practicalities of doing this in a mu trigger a short timer after each commit, then to start a build when either this timer expires, or after a rather longer Hudson offer this scheduling automatically.

Another factor is the need for a version control system that supports atomic commits, i.e. all of a developer's chang to build from only half of the changed files.

Maintain a code repository

This practice advocates the use of a revision control system for the project's source code. All artifacts required to b in the revision control community, the convention is that the system should be buildable from a fresh checkout an Martin Fowler also mentions that where branching is supported by tools, its use should be minimised[*citation n* creating multiple versions of the software that are maintained simultaneously. The mainline (or trunk) should be the

Automate the build

A single command should have the capability of building the system. Many build-tools, such as make, have existed IBM Rational Build Forge are frequently used in continuous integration environments. Automation of the bu deployment into a production-like environment. In many cases, the build script not only compiles binaries, but a media (such as Windows MSI files, RPM or DEB files).

Make the build self-testing

Once the code is built, all tests should run to confirm that it behaves as the developers expect it to behave.

Everyone commits to the baseline every day

By committing regularly, every committer can reduce the number of conflicting changes. Checking in a week's worth difficult to resolve. Early, small conflicts in an area of the system cause team members to communicate about the ch

Many programmers recommend committing all changes at least once a day (once per feature built), and in addition

Every commit (to baseline) should be built

The system should build commits to the current working version in order to verify that they integrate correctly. A this may be done manually. For many, continuous integration is synonymous with using Automated Continuous In version control system for changes, then automatically runs the build process.

Keep the build fast

The build needs to complete rapidly, so that if there is a problem with integration, it is quickly identified.

Test in a clone of the production environment

Having a test environment can lead to failures in tested systems when they deploy in the production environ environment in a significant way. However, building a replica of a production environment is cost prohibitive. In version of the actual production environment to both alleviate costs while maintaining technology stack composition

Make it easy to get the latest deliverables

Making builds readily available to stakeholders and testers can reduce the amount of rework necessary when rebuild reduces the chances that defects survive until deployment. Finding errors earlier also, in some cases, reduces the am

Everyone can see the results of the latest build

It should be easy to find out where/whether the build breaks and who made the relevant change.

Automate deployment

Most CI systems allow the running of scripts after a build finishes. In most situations, it is possible to write a scri at. A further advance in this way of thinking is Continuous Deployment, which calls for the software to be deploye defects or regressions[5].

History

Continuous Integration emerged in the Extreme Programming (XP) community, and XP advocates Martin Fowl Fowler's paper[6] is a popular source of information on the subject. Beck's book *Extreme Programming Explained* term.

Advantages and disadvantages

Advantages

Continuous integration has many advantages:

- when unit tests fail or a bug emerges, developers might revert the codebase back to a bug-free state, without wa
- developers detect and fix integration problems continuously - avoiding last-minute chaos at release dates, (when
- early warning of broken/incompatible code
- early warning of conflicting changes
- immediate unit testing of all changes
- constant availability of a "current" build for testing, demo, or release purposes
- immediate feedback to developers on the quality, functionality, or system-wide impact of code they are writing
- frequent code check-in pushes developers to create modular, less complex code[*citation needed*]
- metrics generated from automated testing and CI (such as metrics for code coverage, code complexity, and featu and help develop momentum in a team[*citation needed*]

Disadvantages

- initial setup time required
- well-developed test-suite required to achieve automated testing advantages
- large-scale refactoring can be troublesome due to continuously changing code base

- hardware costs for build machines can be significant

Many teams using CI report that the advantages of CI well outweigh the disadvantages.[8] The effect of finding a time and money over the lifespan of a project.

Software

To support continuous integration, software tools such as automated build software can be employed.

Software tools for continuous integration include:

- AnthillPro — continuous integration server by Urbancode
- Apache Continuum — continuous integration server supporting Apache Maven and Apache Ant. Supports CVS,
- Apache Gump — continuous integration tool by Apache
- Automated Build Studio — proprietary automated build, continuous integration and release management systen
- Bamboo — proprietary continuous integration server by Atlassian Software Systems
- BuildBot — Python/Twisted-based continuous build system
- BuildForge - proprietary automated build engine by IBM / Rational
- BuildMaster — proprietary application lifecycle management and continuous integration tool by Inedo
- CABIE - Continuous Automated Build and Integration Environment — open source, written in Perl; works witl
- Cascade — proprietary continuous integration tool; provides a checkpointing facility to build and test changes b
- codeBeamer — proprietary collaboration software with built-in continuous integration features
- CruiseControl — Java-based framework for a continuous build process
- CruiseControl.NET — .NET-based automated continuous integration server
- CruiseControl.rb - Lightweight, Ruby-based continuous integration server that can build any codebase, not only
- ElectricCommander — proprietary continuous integration and release management solution from Electric Cloud
- FinalBuilder Server — proprietary automated build and continuous integration server by VSoft Technologies
- Go — proprietary agile build and release management software by Thoughtworks
- Jenkins (formerly known as Hudson) — MIT-licensed, written in Java, runs in servlet container, supports CVS, and shell scripts
- Software Configuration and Library Manager — software configuration management system for z/OS by IBM R
- QuickBuild - proprietary continuous integration server with free community edition featuring build life cycle ma
- TeamCity — proprietary continuous-integration server by JetBrains with free professional edition
- Team Foundation Server — proprietary continuous integration server and source code repository by Microsoft
- Tinderbox — Mozilla-based product written in Perl
- Rational Team Concert — proprietary software development collaboration platform with built-in build engine by

See comparison of continuous integration software for a more in depth feature matrix.

Further reading

- Duvall, Paul M. (2007). *Continuous Integration. Improving Software Quality and Reducing Risk*. Addison-Wesle

References

1. Cunningham, Ward (05 Aug 2009). "Integration Hell". *WikiWikiWeb*. http://c2.com/cgi/wiki?IntegrationHell. R
2. Brauneis, David (01 January 2010). "[OSLC Possible new Working Group - Automation"]. *open-services.net Con* services.net/pipermail/community_open-services.net/2010-January/000214.html. Retrieved 16 February 2010.
3. Taylor, Bradley. "Rails Deployment and Automation with ShadowPuppet and Capistrano". http://blog.railsmac shadowpuppet-and-capistrano/.
4. Fowler, Martin. "Continuous Integration". http://martinfowler.com/articles/continuousIntegration.html#Practic
5. See Continuous deployment in 5 easy steps - O'Reilly Radar (http://radar.oreilly.com/2009/03/continuous-deplo impossible fifty times a day. - Timothy Fitz (http://timothyfitz.wordpress.com/2009/02/10/continuous-deploym
6. Fowler, Martin. "Continuous Integration". http://www.martinfowler.com/articles/continuousIntegration.html.
7. Beck, Kent (1999). *Extreme Programming Explained*. ISBN 0-201-61641-6.
8. Richardson, Jared (September 2008). "Agile Testing Strategies at No Fluff Just Stuff Conference". Boston, Mass

External links

- Comparison of continuous integration software (http://en.wikipedia.org/wiki/Comparison_of_continuous_integ
- Continuous integration (http://www.martinfowler.com/articles/continuousIntegration.html) by Martin Fowler (
- Continuous Integration at the Portland Pattern Repository (http://www.c2.com/cgi/wiki?ContinuousIntegratio
- Cross platform testing at the Portland Pattern Repository (http://c2.com/cgi/wiki?CrossPlatformTesting)
- Continuous Integration Server Feature Matrix (http://confluence.public.thoughtworks.org/display/CC/CI+Feat
- Continuous Integration: The Cornerstone of a Great Shop (http://www.methodsandtools.com/archive/archive.p
- A Recipe for Build Maintainability and Reusability (http://jayflowers.com/joomla/index.php?option=com_cont
- Continuous Integration anti-patterns (http://www.ibm.com/developerworks/java/library/j-ap11297/) by Paul D
- Extreme programming (http://www.extremeprogramming.org/rules/integrateoften.html)

Bug Tracking

A **bug tracking system** is a software application that is designed to help quality assurance and programmers ke
a type of issue tracking system.

Many bug-tracking systems, such as those used by most open source software projects, allow users to enter bug re
organization doing software development. Typically bug tracking systems are integrated with other software project

Having a bug tracking system is extremely valuable in software development, and they are used extensively by co
tracking system is considered one of the "hallmarks of a good software team".[1]

Components

A major component of a bug tracking system is a database that records facts about known bugs. Facts may inc
behavior, and details on how to reproduce the bug; as well as the identity of the person who reported it and any pro

Typical bug tracking systems support the concept of the life cycle for a bug which is tracked through status assig
configure permissions based on status, move the bug to another status, or delete the bug. The system should also a
bug in a particular status can be moved. Some systems will e-mail interested parties, such as the submitter and assig

Usage

The main benefit of a bug-tracking system is to provide a clear centralized overview of development requests (inc
their state. The prioritized list of pending items (often called backlog) provides valuable input when defining the pro

In a corporate environment, a bug-tracking system may be used to generate reports on the productivity of progr
results because different bugs may have different levels of severity and complexity. The severity of a bug may not
different opinions among the managers and architects.

A *local bug tracker (LBT)* is usually a computer program used by a team of application support professionals
developers. Using an LBT allows support professionals to track bugs in their "own language" and not the "langu
professionals to track specific information about users who have called to complain — this information may not a
tracking systems when an LBT is in place.

Bug tracking systems as a part of integrated project management systems

Bug and issue tracking systems are often implemented as a part of integrated project management systems. This
development process, fixing bugs in several product versions, automatic generation of a product knowledge base and

Distributed bug tracking

Some bug trackers are designed to be used with distributed revision control software. These distributed bug tracke
updated while a developer is offline.[3] Distributed bug trackers include Bugs Everywhere, and Fossil.

Recently, commercial bug tracking systems have also begun to integrate with distributed version control. FogBu
Kiln.[4]

Although wikis and bug tracking systems are conventionally viewed as distinct types of software, ikiwiki can also b
as well, in an integrated distributed manner. However, its query functionality is not as advanced or as user-friendly
statements can be made about org-mode, although it is not wiki software as such.

Bug tracking and test management

While traditional test management tools such as HP Quality Center and Rational Software come with their own
systems.[*citation needed*]

References

1. Joel Spolsky (November 08 2000). "Painless Bug Tracking". http://www.joelonsoftware.com/articles/fog0000000
2. Multiple (wiki). "Bug report". *Docforge*. http://docforge.com/wiki/Bug_report. Retrieved 2010-03-09.
3. Jonathan Corbet (May 14 2008). "Distributed bug tracking". *LWN.net*. http://lwn.net/Articles/281849/. Retriev

4. "FogBugz Features". *Fogbugz.com*. http://www.fogcreek.com/FogBugz/learnmore.html. Retrieved 2010-10-29.
5. Joey Hess (6 April 2007). "Integrated issue tracking with Ikiwiki". *LinuxWorld.com*. IDG. http://www.linuxworl Retrieved 7 January 2009.

External links

- Bug Tracking Software (http://www.dmoz.org/Computers/Software/Configuration_Management/Bug_Trackin
- How to Report Bugs Effectively (http://www.chiark.greenend.org.uk/~sgtatham/bugs.html)
- List of distributed bug tracking software (http://dist-bugs.kitenet.net/software/)

Decompiler

Introduction to Software Engineering/Tools/Decompiler

Obfuscation

Introduction to Software Engineering/Tools/Obfuscation

Re-engineering

Introduction

Introduction to Software Engineering/Reengineering

Reverse Engineering

Introduction to Software Engineering/Reengineering/Reverse Engineering

Round-trip Engineering

Introduction to Software Engineering/Reengineering/Round-trip Engineering

Authors

Introduction to Software Engineering/Authors

License

GNU Free Documentation License

GNU Free Documentation License

Note: current version of this book can be found at http://en.wikibooks.org/wiki/Introduction_to_Software_Enginee