CS 4320 / 7320
Software Engineering

Version Control Systems

This Week in Software Engineering

Semester Week	Week	Tuesday	Thursday	Reading	Assignment(s)	Assignment(s) Due
1	January 21, 2019	*Introduction to Engineering Software and Systems Theory * Ethics	* Software development lifecycles * Version Control and Configuration Management	1. Ethics in Software Engineering 2. The Software Development Lifecycle 3. Systems Theory Overview	1. GitHub Assignment 2. Resume Submission	January 28, 2019

Topics

- Version Control Systems
  - What are they
  - Types
- Terminologies
- Collaborative Development through VCS
- Introduction GIT

Version Control Systems

Centralized

- Work directly against a central server
- Subversion, CVS, Perforce, etc.

Distributed

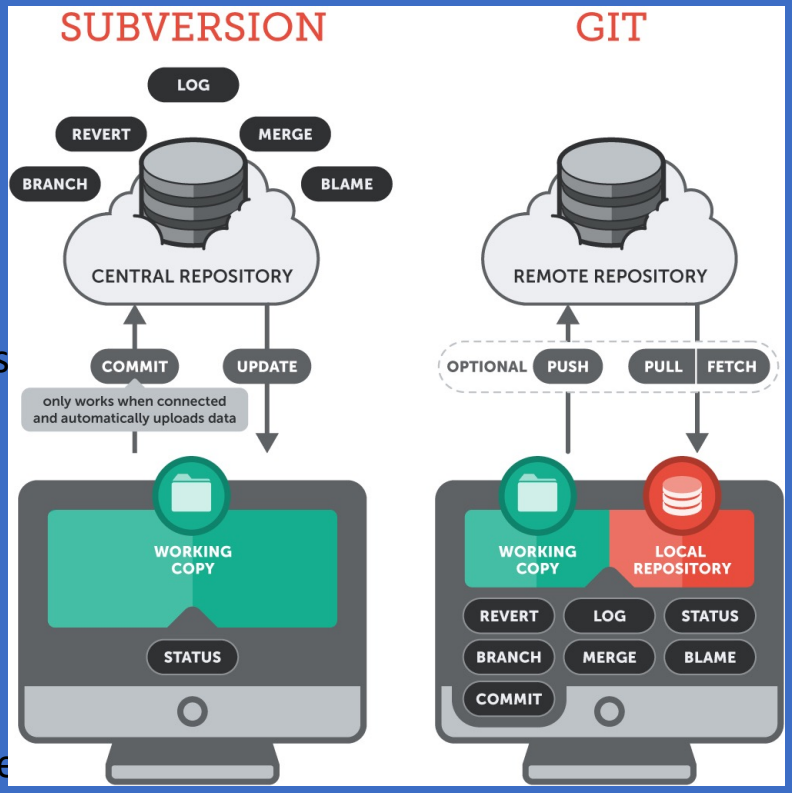
- Work locally, optionally push to remote repositories
- Git, Mercurial (Hg), etc.

Hosted solutions

- GitHub, BitBucket, Visual Studio Online, etc.

On-Premise solutions

- GitHub Enterprise, Stash, Team Foundation Server, e



git-tower.com

Version Control System

- Track changes and revisions to both files and file system structure of working environment
  - File Additions / Deletes
  - Folder Additions / Deletes
  - File Edits
- aka
  - Source control system
  - Source code control system
  - Revision control system

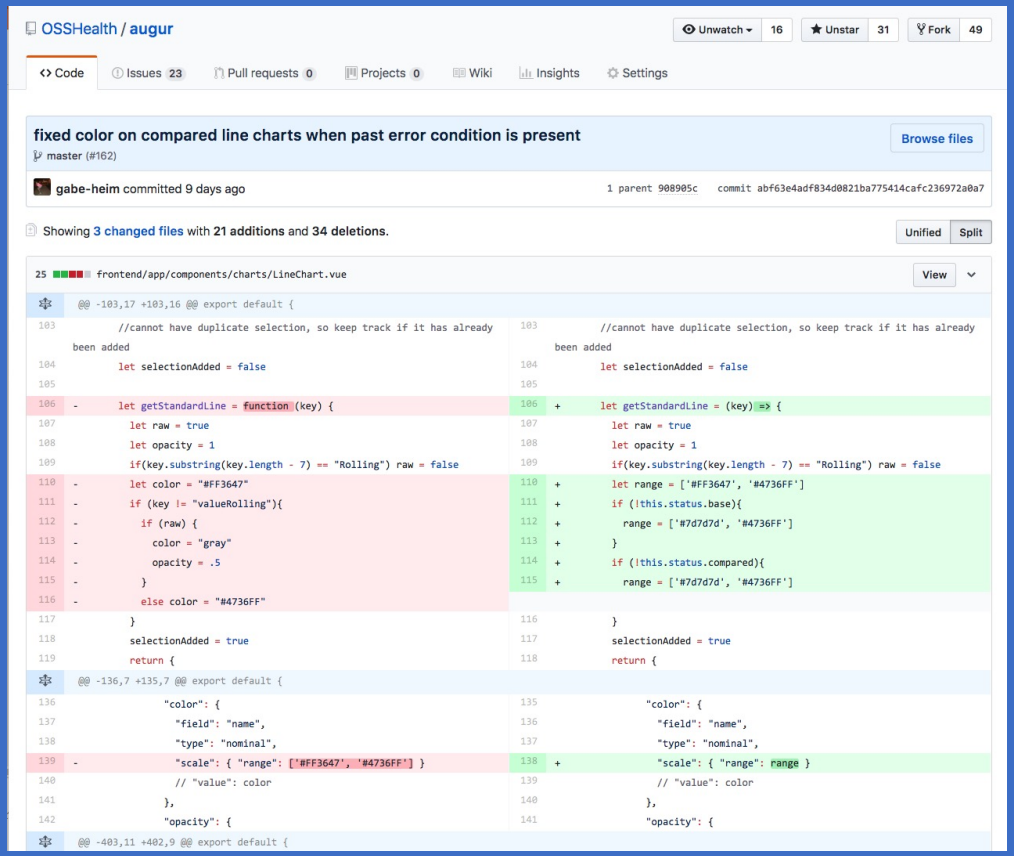
What is Version Control?

System to manage changes to files

- Who — Sean Goggins
- When — January 3, 2012
- What — Files and differential
- (Sometimes) Why — Fix Color …

Why use it?

- Revert to or review prior changes
- Maintain multiple versions
- Compare differences
- Share and collaborate
- Modern solutions might assume you have it!
  - Infrastructure-as-code
  - Continuous Integration and Delivery
- Google around for many more reasons!



AKA Revision Control, Source Control

Change Tracking

- Files (and folders) exist in a temporal condition
- File A:
  - T-1 has 100 lines
  - T-101 has 1000 lines
  - How was the file changed over time from T-1 to T-101 ?
- VCS can answer this question easily
- Can also roll-back to point in time, e.g., T-99

VCS Types

- VCS come in various flavors that roughly align to three types of systems:
  - Local
  - Client-Server
  - Networked / Distributed

VCS Types : Local

- Source Code Control System (SCCS)
  - Developed at Bell Labs, 70's
  - Critical early stage use in development of UNIX
  - Part of the *Single UNIX Specification*
- File are locally version controlled
- Collaboration is limited to a single system
- Some VCS still use internals

VCS Types : Client-Server

- Client programs read and write changes to a development tree that exist on a server
  - Multiple developers can pull down to push up changes
- Concurrent Versions System (CVS)
  - Or: Concurrent Versioning System
  - 80's
- Subversion (SVN)
  - One of most popular today
  - Early 2000s

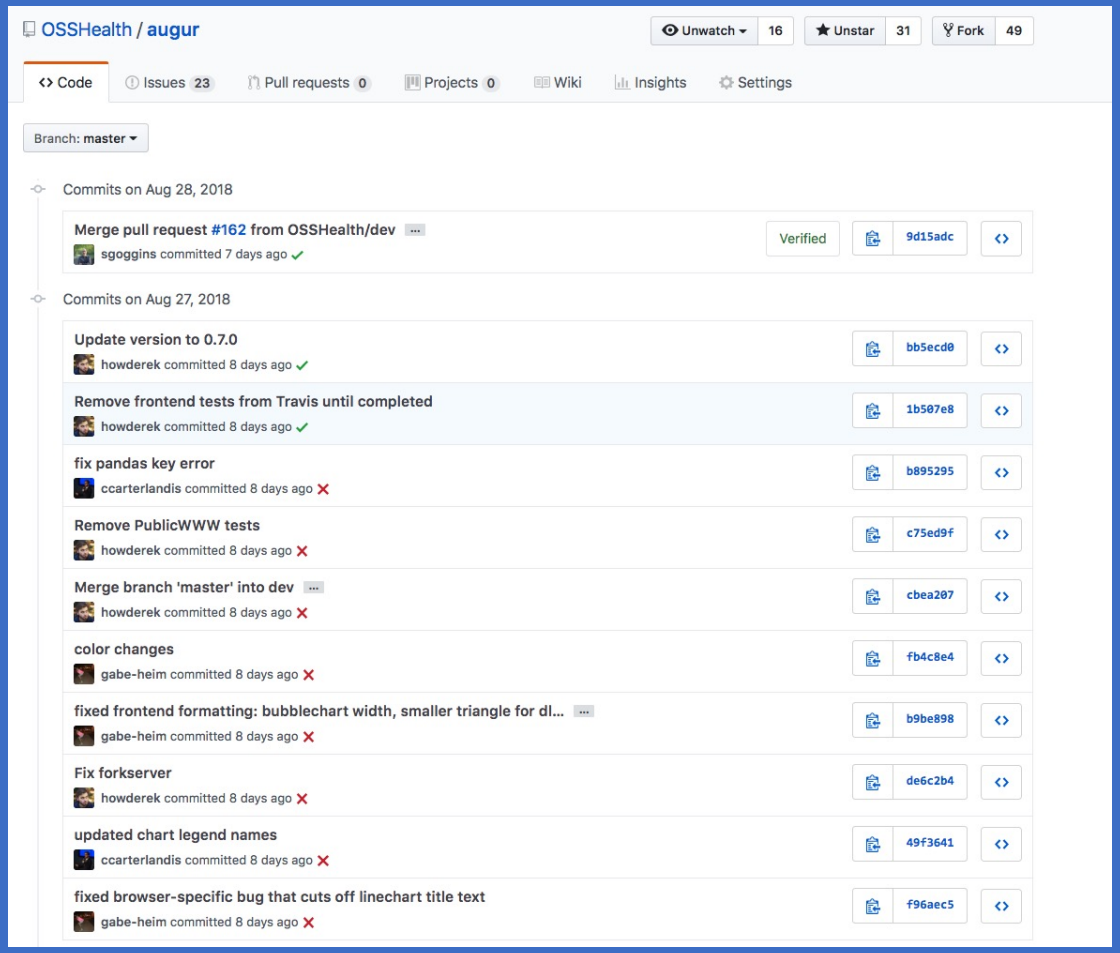
VCS Types : Distributed

- Decentralized VCS,
  - Built from the concepts of peer-to-peer trust
  - Each user has full repository in local storage
- GIT is most common
  - Developed by Linus Torvolds specifically for Linux Kernel development
  - 2005
  - GIT and other distributed VCS becoming the most popular VCS, close to SVN in usage

VCS Advantages

- Provides a control and tracking method to collaborative software development
  - Concepts can be applied to documents, e.g., non-software
- Changes (revisions)
  - Tracked by numerical or hash id
  - Timestamped
  - User stamped

Hash ID's


OSSHealth / augur
Unwatch 16
Unstar 31
Fork 49
Code
Issues 23
Pull requests 0
Projects 0
Wiki
Insights
Settings
Branch: master
Commits on Aug 28, 2018
Merge pull request #162 from OSSHealth/dev
sgoggins committed 7 days ago
Verified
9d15adc
Commits on Aug 27, 2018
Update version to 0.7.0
howderek committed 8 days ago
bb5ecd0
Remove frontend tests from Travis until completed
howderek committed 8 days ago
1b507e8
fix pandas key error
ccarterlandis committed 8 days ago
b895295
Remove PublicWWW tests
howderek committed 8 days ago
c75ed9f
Merge branch 'master' into dev
howderek committed 8 days ago
cbea207
color changes
gabe-heim committed 8 days ago
fb4c8e4
fixed frontend formatting: bubblechart width, smaller triangle for dl...
gabe-heim committed 8 days ago
b9be898
Fix forkserver
howderek committed 8 days ago
de6c2b4
updated chart legend names
ccarterlandis committed 8 days ago
49f3641
fixed browser-specific bug that cuts off linechart title text
gabe-heim committed 8 days ago
f96aec5

VCS Advantages

- Version control is important for development groups to function effectively
- Also utilized for non-software development
  - Word-processing
  - Configuration files
  - Content management systems
  - Database records

VCS Terminology

- Trunk / Main / Master
  - the primary development branch
  - often the receiver of changes from other branches that are used for small development efforts, e.g., bug-fixes
- Branching
  - Duplication of a folder structure for the purpose of isolating development work from the Trunk

Branch Demo

VCS Terminology

- Merge
  - Reconciling multiple changes to a version controlled resource
  - e.g., two versions of a file, the end result is one file with both sets of changes
- Fork
  - A branch that is not intended to be later merged
    - **NOTE, on GitHub, Forks are merged back using Pull Requests. More on this Later.
-

Merge and Fork Demos

VCS Terminology

- Tag
  - A read-only branch that serves as the end-point of a development effort / interval, e.g., a release
  - Captures a branch at a point in time
  - Labels the point in time
- Commit
  - Saving a change to live files into the repository's set of know edits, i.e., revisions

Tag Demo

VCS Terminology

- Baseline
  - The starting point of a branch
- Delta / Diff
  - A revision to one or more files or the file system (tree)
- Conflict
  - Two or more users have changed the same version controlled resource in a manner that cannot be automatically resolved by the VCS

VCS Terminology

- Head
  - The most recent / current / up-to-date version of a branch
- Update / Pull
  - Pulling in changes from other developers
- Working copy
  - The local working copy of files from the repository

GitHub

Global setup:

```
Set up git
  git config --global user.name "Rich Jones"
  git config --global user.email rich@anomos.info
```

Next steps:

```
  mkdir TestRepository
  cd TestRepository
  git init
  touch README
  git add README
  git commit -m 'first commit'
  git remote add origin git@github.com:Miserlou/TestRepository.git
  git push -u origin master
```

Existing Git Repo?

```
  cd existing_git_repo
  git remote add origin git@github.com:Miserlou/TestRepository.git
  git push -u origin master
```

GitHub Now!

GitHub

git

Goals

Use version control for your own projects

Background to make a case to bring version control to your team at $Work

Further Reading

There's a lot out there. Here are some highlights. Don't be intimidated, you don't need all this, but it may come in handy if you want to dive a bit deeper into the weeds. The key is starting to use it, just like PowerShell!

Interactive guides

- GitHub's interactive guide – Learn Git in 15 minutes: https://try.github.io/
- Learn Git Branching: http://pcottle.github.io/learnGitBranching
- Interactive Cheat sheet: http://ndpsoftware.com/git-cheatsheet.html

References

- Official git reference: http://git-scm.com/docs
- Pro Git (free!): http://www.git-scm.com/book/en/v2 - the first two or three chapters are a great intro
- Understanding Branches: http://blog.thoughtram.io/git/rebase-book/2015/02/10/understanding-branches-in-git.html
- Git Explained: For Beginners: http://www.dotnetcodegeeks.com/2015/06/git-explained-for-beginners.html
- GitHub Flow – GitHub from a Browser: https://github.com/blog/1557-github-flow-in-the-browser

Contributing to Microsoft's DSC Resources on GitHub

- Guide to getting started with GitHub: https://github.com/PowerShell/DscResources/blob/master/GettingStartedWithGitHub.md
- DSC Contributions guide: https://github.com/PowerShell/DscResources/blob/master/CONTRIBUTING.md
- DSC Resource Style Guidelines: https://github.com/PowerShell/DscResources/blob/master/StyleGuidelines.md

More references

- Pro Git (free!): http://www.git-scm.com/book/en/v2 - the rest of the book :)
- A Visual Git Reference: http://marklodato.github.io/visual-git-guide/index-en.html
- Git From the Inside Out: https://codewords.recurse.com/issues/two/git-from-the-inside-out
- Git For Computer Scientists: http://eagain.net/articles/git-for-computer-scientists - Great read with pictures, don't be intimidated by the title
- Branching, forking, other concepts explained: http://stackoverflow.com/questions/3329943/git-branch-fork-fetch-merge-rebase-and-clone-what-are-the-differences

CS 4320 / 7320
Software Engineering

Software Engineering Lifecycle
& Basic Concepts

Software Engineering

- What is engineering, as a discipline?
- What is software?
- What are the consequences of poor engineering?

Software Engineering

- What is engineering, as a discipline?

"The application of scientific, economic, social, and practical knowledge in order to design, build, and maintain structures, machines, devices, systems, materials and processes"

- wikipedia

Software Engineering

- What is software?

Computer programs **and related documentation of requirements, designs, models, and manuals.**

- Sommerville

Software Engineering

- What are the consequences of poor engineering?

Horrible, horrible consequences!!!

Engineering Failures

Engineering Failures

- What are the consequences of poor engineering?

Source: http://en.wikipedia.org/wiki/File:TacomaNarrowsBridgeCollapse_in_color.jpg

Engineering Failures

- What are the consequences of poor engineering?

Tacoma Narrows Bridge Collapse (1940):

42 mph winds

- http://www.wsdot.wa.gov/tnbhistory/machine/machine3.htm
- Poor engineering practices did not account for aerodynamics of suspension bridge design
- See:

http://upload.wikimedia.org/wikipedia/commons/1/19/Tacoma_Narrows_Bridge_destruction.ogg

Engineering Failures

- What are the consequences of poor engineering?

But, what about today, with current computer modeling techniques?

2007 I-35 bridge collapse in Minneapolis

"An employee of the NTSB had written his doctoral thesis on possible failure scenarios of this specific bridge while he was a student at the nearby University of Minnesota. That thesis, including his computer model of the bridge for failure mode analysis, was used by the NTSB to aid in their investigation."

- http://en.wikipedia.org/wiki/I-35W_Mississippi_River_bridge

Engineering Failures

Source: http://en.wikipedia.org/wiki/File:I35_Bridge_Collapse_4crop.jpg

Engineering Failures

- What are the consequences of poor engineering?

Hyatt Regency, KCMO

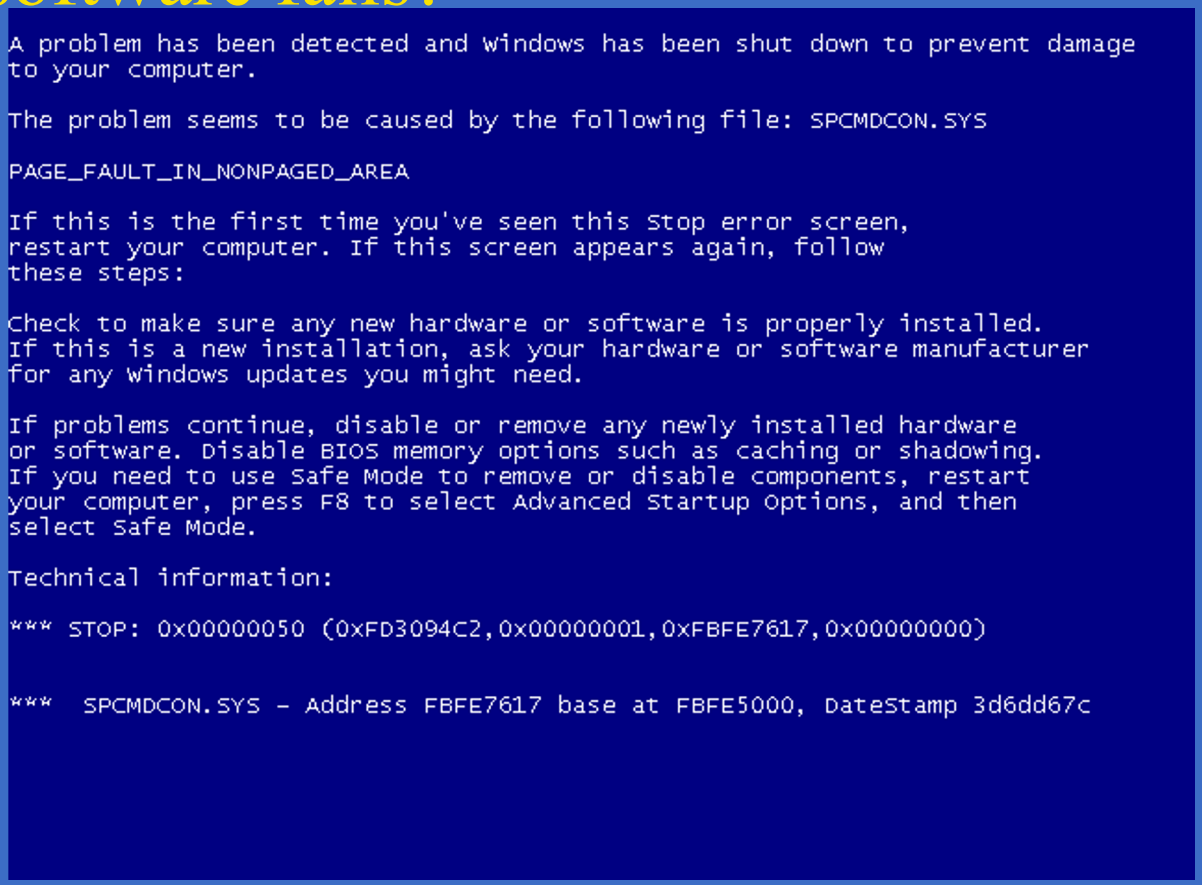
An elevated walkway gathered a crowd watching a dance below. The static load of crowd = failure

Source: http://en.wikipedia.org/wiki/File:Hyatt_Regency_collapse_floor_view.PNG

Engineering Failures

- What about when software fails?

Windows XP Blue Screen of Death



- Frustrating

Source: http://en.wikipedia.org/wiki/File:Windows_XP_BSOD.png

Engineering Failures

- What about when software fails?

Bill Gates demo of Windows 98

- Embarrassing !!!

Source: http://www.techweekeurope.co.uk/wp-content/uploads/2012/10/BILL-GATES-

Engineering Failures

- … or very costly! "

Date: 9/1997

(By James Gleick) It took the European Space Agency **10 years and $7 billion to produce Ariane 5**, a giant rocket capable of hurling a pair of three-ton satellites into orbit with each launch and intended to give Europe overwhelming supremacy in the commercial space business.

All it took to **explode** that rocket **less than a minute into its maiden voyage** last June, scattering fiery rubble across the mangrove swamps of French Guiana, was a small **computer program** trying to stuff a **64-bit number into a 16-bit space**.

...

This shutdown occurred 36.7 seconds after launch, when the guidance system's own computer tried to convert one piece of data -- the sideways velocity of the rocket -- from a 64-bit format to a 16-bit format. The number was too big, and an overflow error resulted. When the guidance system shut down, it passed control to an identical, redundant unit, which was there to provide backup in case of just such a failure. But the second unit had failed in the identical manner a few milliseconds before. And why not? It was running the same software.

"

Source: http://www.cse.lehigh.edu/~gtan/bug/softwarebug.html#ariane

Engineering Failures

- … or life threatening! "

Date: 9/14/**2004**

(IEEE Spectrum) -- It was an air traffic controller's worst nightmare. Without warning, on Tuesday, 14 September, at about 5 p.m. Pacific daylight time, air traffic controllers lost voice contact with 400 airplanes they were tracking over the southwestern United States. Planes started to head toward one another, something that occurs routinely under careful control of the air traffic controllers, who keep airplanes safely apart. But now the controllers had no way to redirect the planes' courses.

...

The controllers lost contact with the planes when the main voice communications system shut down unexpectedly. To make matters worse, a backup system that was supposed to take over in such an event crashed within a minute after it was turned on. The outage disrupted about 800 flights across the country.

...

Inside the control system unit is a countdown timer that ticks off time in milliseconds. The VCSU uses the timer as a pulse to send out periodic queries to the VSCS. It starts out at the highest possible number that the system's server and its software can handle—2^32. It's a number just over 4 billion milliseconds. When the counter reaches zero, the system runs out of ticks and can no longer time itself. So it shuts down.

Counting down from 2^32 to zero in milliseconds takes just under 50 days. The FAA procedure of having a technician reboot the VSCS every 30 days resets the timer to 2^32 almost three weeks before it runs out of digits. "

Source: http://www.cse.lehigh.edu/~gtan/bug/softwarebug.html#aircontrol

Software Engineering

- Software Engineering:

"Engineering discipline that is concerned with all aspects of software production"

- Sommerville

- Recall that software is:

Computer programs and related documentation of requirements, designs, models, and manuals.

Software Engineering

- Theories, methods and tools for professional software development
  - Computer Code/Programs
  - Requirements Documents
  - Design Documents
  - Data, Process, and other models
  - User manuals
- In other words; coding is a small portion of software engineering, the majority is technical documentation.

Technical Documents

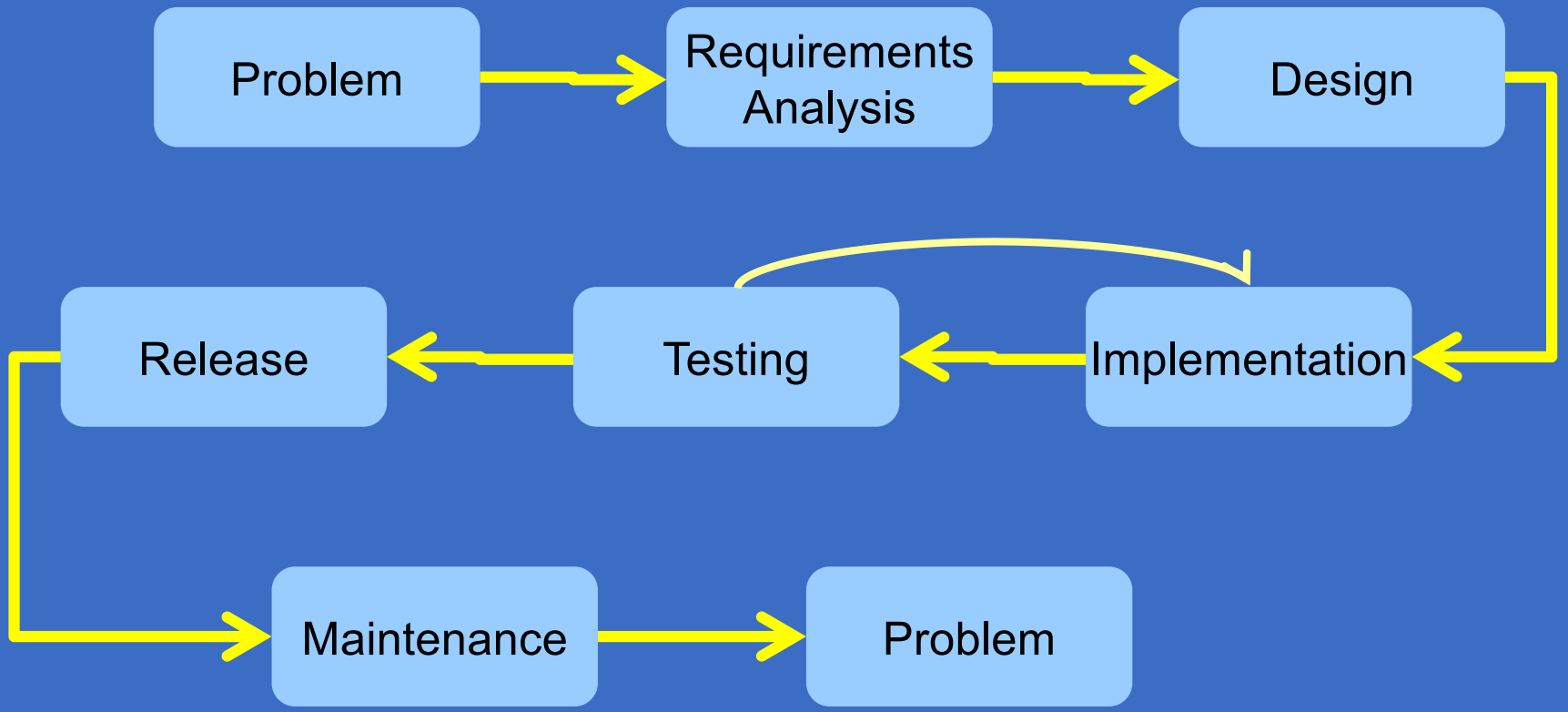
- Software engineering does not exist outside the creation of a variety of technical documents
- Analogous to:
  - Blue prints for construction projects
  - Logic design and wiring diagrams for hardware fabrication
  - Models, prototypes, and computer simulations for aircraft and space vehicles
- NASA happens to have some of the best software engineers for a reason

Software Engineering

- Why is it so important to get software engineering right?
- Software engineering is more than just
  - Designing a solution
  - Coding a solution
- Software engineering is also about process management

Software Engineering

- What is the high-level process?



Software Development Lifecycle

- Problem: Some entity (program, process, etc) that requires software to be created as the preferred solution.
- Requirements: The explicit, well defined, thorough goals of the software - i.e., what should it do?
- Design: The detailed plans that will facilitate the construction of computer code to sufficiently solve the problem.

Software Development Lifecycle

- Implementation: The construction of the computer program that is a solution to the problem.

- Testing: Validating the requirements are met and the outcomes of computer programs match the expectations set forth in the Requirements and Design stages.
  - Note: Testing can drive further implementation

Software Development Lifecycle

- Release: The transition of computer programs from the developers to the users.
- Maintenance: The continual upkeep of the software, including computer programs, documentation, requirements, design, testing, and enhancement/bug-fix implementation.
- Problem: ?

Software Development Lifecycle

- Problem: ???
  - A problem is the end of all software.
  - Usually, the software is no longer needed
- Obsolete or replaced
- New software takes the place of old software, as a clean solution to a problem that is usually different, or broader scoped, than the original problem.
  - Eventually, all software becomes obsolete.
  - The hardware that runs the most solid, perfect legacy code is eventually replaced.

Characteristic Goals for SWE

- Maintainability:
  - Designed to evolve to meet the changing needs of the customer
  - Change is inevitable, hence *Change Management*
- Dependability:
  - Reliable, performs consistently as expected
  - Should not cause damage or fail
- Security:
  - Operate with integrity, must be able to trust the results of the software

Characteristic Goals for SWE

- Efficiency:
  - No wasteful consumption of resources, e.g.
- network bandwidth, CPU cycles,
- RAM, Secondary storage
  - Responsive to environment and/or users
- Acceptability (compatibility):
  - Software should be designed for the appropriate user base
  - Does the user understand, and can they efficiently operate the software, compatible with environment

The Software Development Lifecycle

Topics covered

- Software process models
- Process activities
- Coping with change
- Process improvement

The software process

A structured set of activities required to develop a software system.

Many different software processes but all involve:

- Specification – defining what the system should do;
- Design and implementation – defining the organization of the system and implementing the system;
- Validation – checking that it does what the customer wants;
- Evolution – changing the system in response to changing customer needs.

A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

Software process descriptions

When we describe and discuss processes, we usually talk about the activities in these processes such as specifying a data model, designing a user interface, etc. and the ordering of these activities.

Process descriptions may also include:

- Products, which are the outcomes of a process activity;
- Roles, which reflect the responsibilities of the people involved in the process;
- Pre- and post-conditions, which are statements that are true before and after a process activity has been enacted or a product produced.

Plan-driven and agile processes

Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan.

In agile processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.

In practice, most practical processes include elements of both plan-driven and agile approaches.

There are no right or wrong software processes.

Software process models

Software process models

The waterfall model

- Plan-driven model. Separate and distinct phases of specification and development.

Incremental development

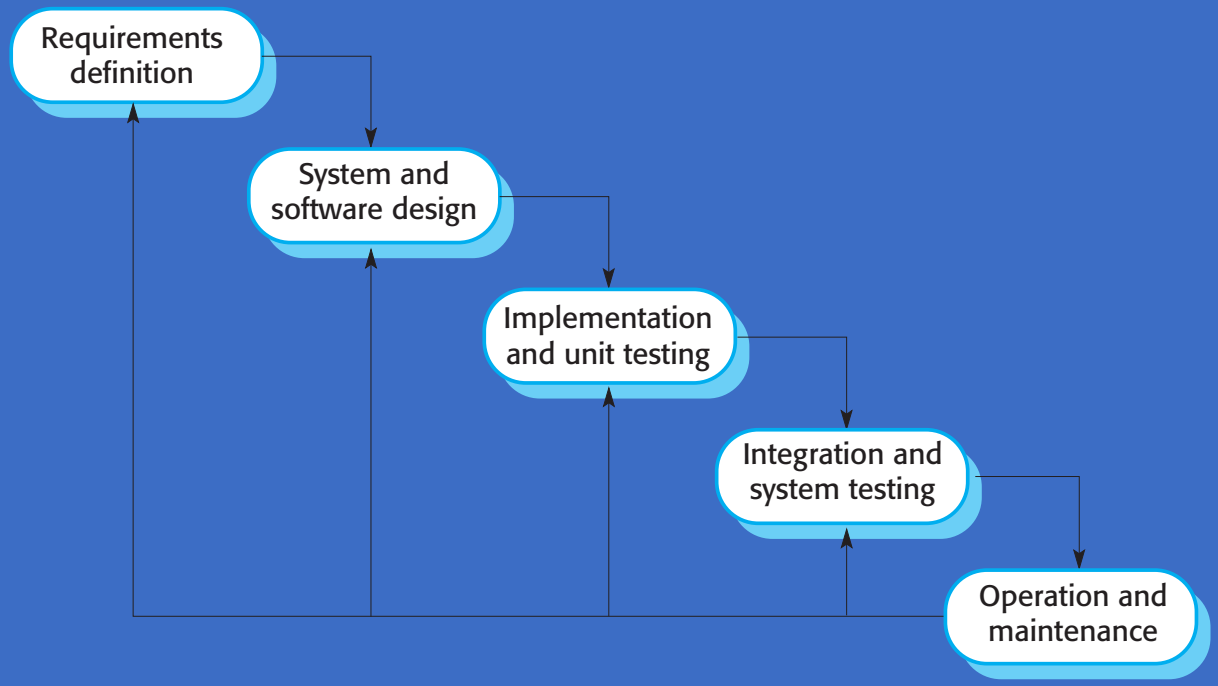
- Specification, development and validation are interleaved. May be plan-driven or agile.

Integration and configuration

- The system is assembled from existing configurable components. May be plan-driven or agile.

In practice, most large systems are developed using a process that incorporates elements from all of these models.

The waterfall model



Waterfall model phases

There are separate identified phases in the waterfall model:

- Requirements analysis and definition
- System and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance

The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, a phase has to be complete before moving onto the next phase.

Waterfall model problems

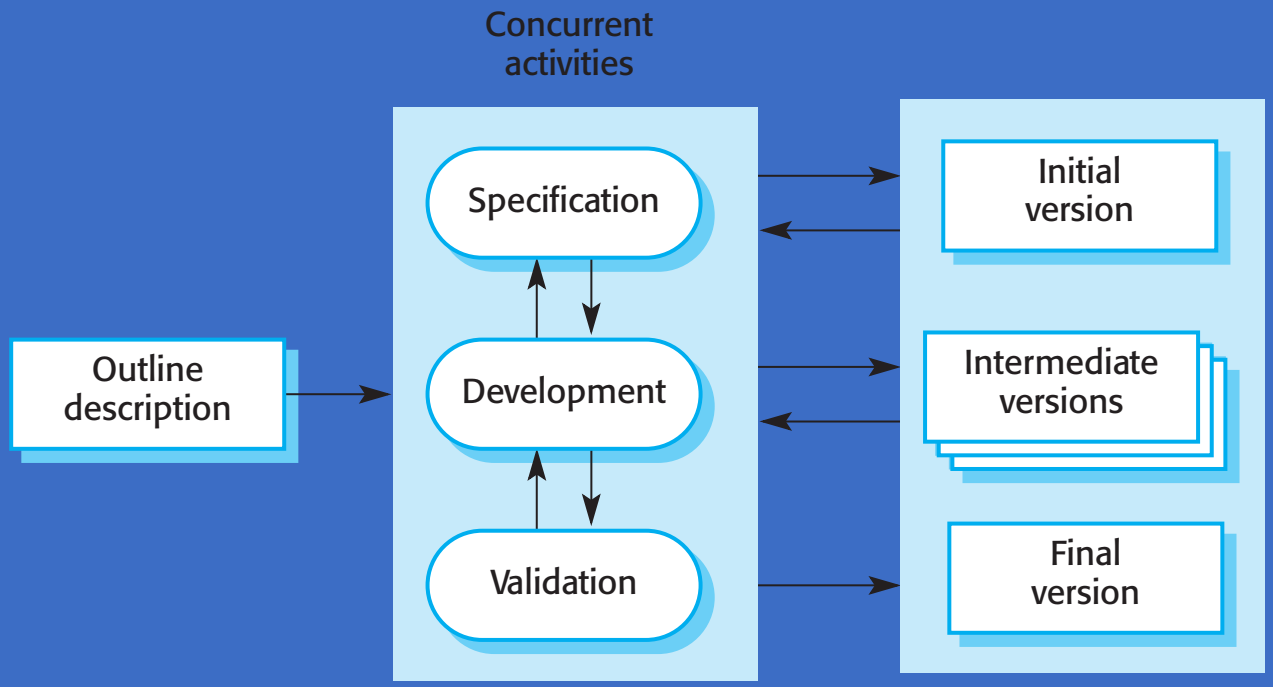
Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.

- Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
- Few business systems have stable requirements.

The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.

- In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

Incremental development



Incremental development benefits

The cost of accommodating changing customer requirements is reduced.

- The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.

It is easier to get customer feedback on the development work that has been done.

- Customers can comment on demonstrations of the software and see how much has been implemented.

More rapid delivery and deployment of useful software to the customer is possible.

- Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

Incremental development problems

The process is not visible.

- Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.

System structure tends to degrade as new increments are added.

- Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

Integration and configuration

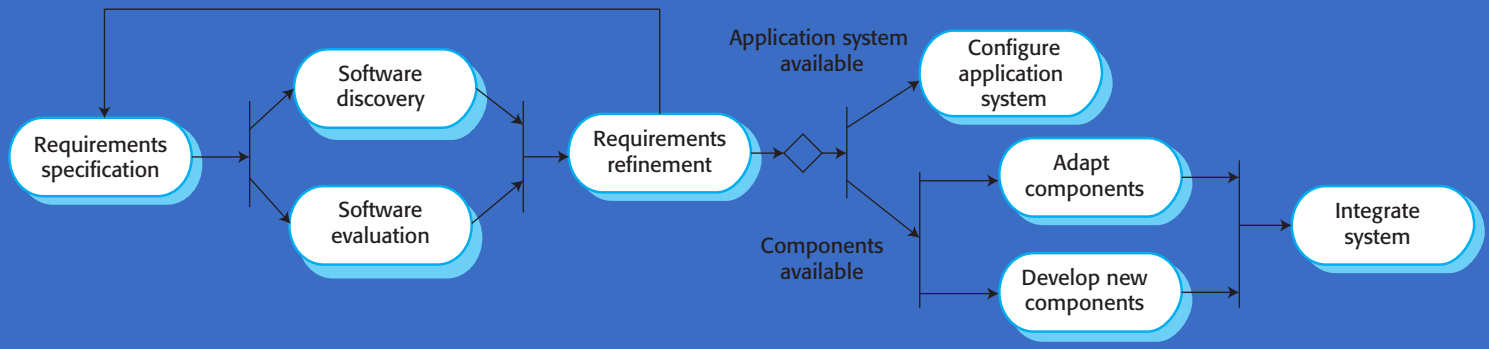
Based on software reuse where systems are integrated from existing components or application systems (sometimes called COTS -Commercial-off-the-shelf) systems).

Reused elements may be configured to adapt their behaviour and functionality to a user's requirements

Types of reusable software

- Stand-alone application systems (sometimes called COTS) that are configured for use in a particular environment.
- Collections of objects that are developed as a package to be integrated with a component framework such as .NET or J2EE.
- Web services that are developed according to service standards and which are available for remote invocation.

Reuse-oriented software engineering



Key process stages

Requirements specification

Software discovery and evaluation

Requirements refinement

Application system configuration

Component adaptation and integration

Advantages and disadvantages

- Reduced costs and risks as less software is developed from scratch
- Faster delivery and deployment of system
- But requirements compromises are inevitable so system may not meet real needs of users
- Loss of control over evolution of reused system elements

SDLC activities

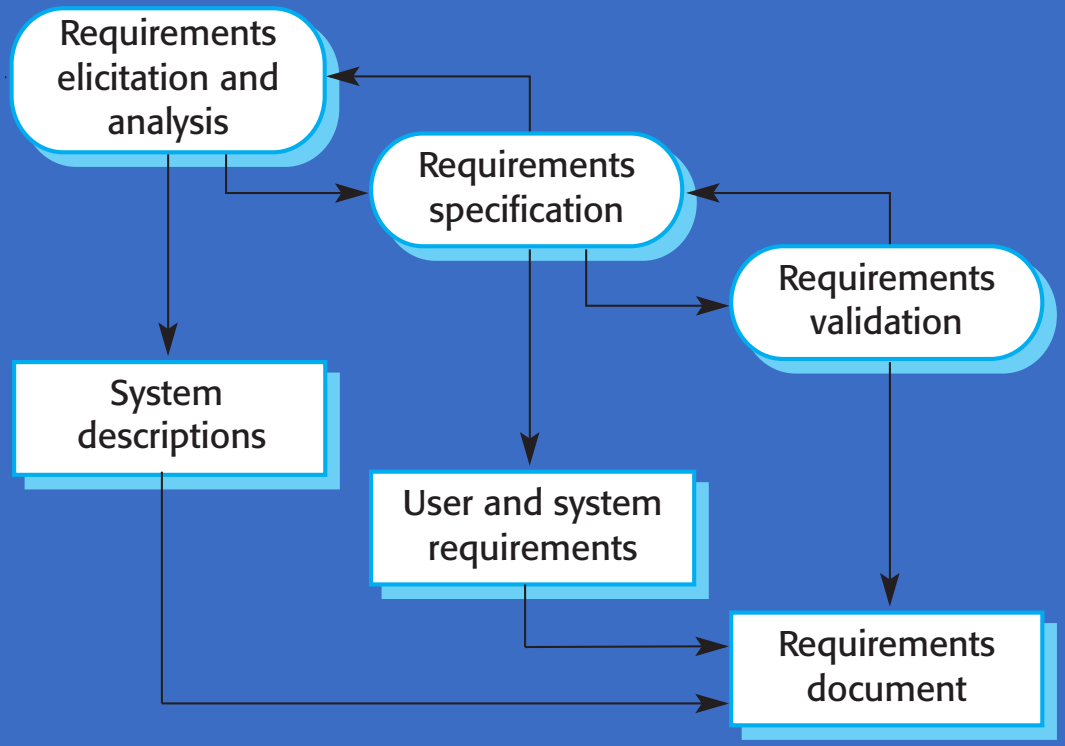
SDLC activities

Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.

The four basic process activities of specification, development, validation and evolution are organized differently in different development processes.

For example, in the waterfall model, they are organized in sequence, whereas in incremental development they are interleaved.

The requirements engineering process



Software specification

The process of establishing what services are required and the constraints on the system's operation and development.

Requirements engineering process

- Requirements elicitation and analysis
  - What do the system stakeholders require or expect from the system?
- Requirements specification
  - Defining the requirements in detail
- Requirements validation
  - Checking the validity of the requirements

Software design and implementation

The process of converting the system specification into an executable system.

Software design

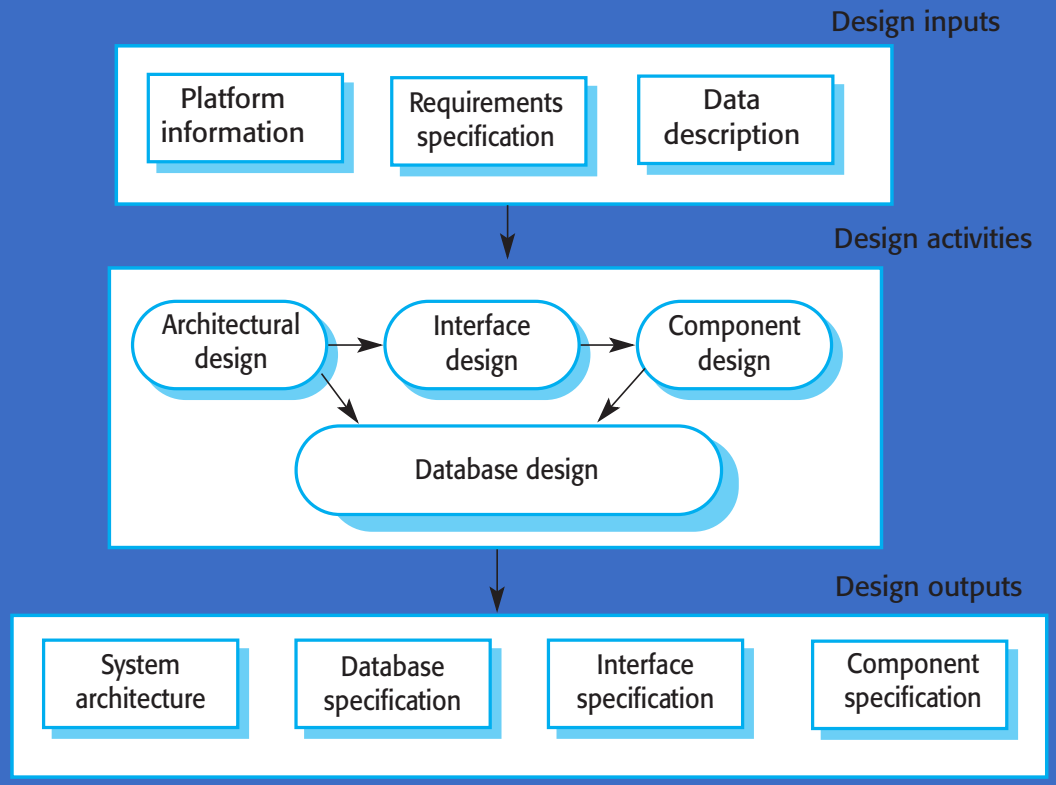
- Design a software structure that realises the specification;

Implementation

- Translate this structure into an executable program;

The activities of design and implementation are closely related and may be inter-leaved.

A general model of the design process



Design activities

Architectural design, where you identify the overall structure of the system, the principal components (subsystems or modules), their relationships and how they are distributed.

Database design, where you design the system data structures and how these are to be represented in a database.

Interface design, where you define the interfaces between system components.

Component selection and design, where you search for reusable components. If unavailable, you design how it will operate.

System implementation

The software is implemented either by developing a program or programs or by configuring an application system.

Design and implementation are interleaved activities for most types of software system.

Programming is an individual activity with no standard process.

Debugging is the activity of finding program faults and correcting these faults.

Software validation

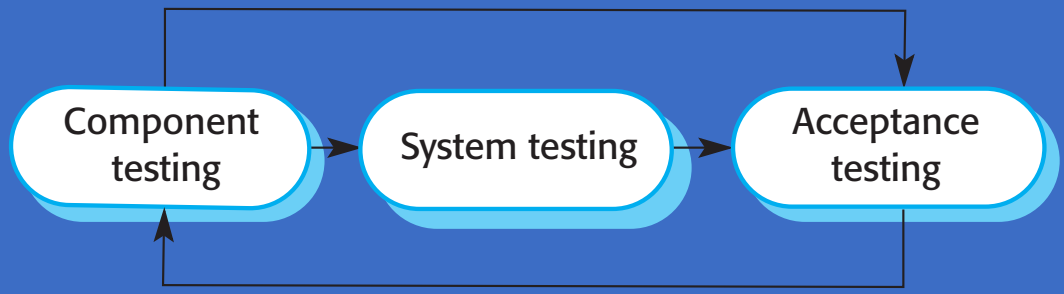
Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.

Involves checking and review processes and system testing.

System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.

Testing is the most commonly used V & V activity.

Stages of testing



Testing stages

Component testing

- Individual components are tested independently;
- Components may be functions or objects or coherent groupings of these entities.

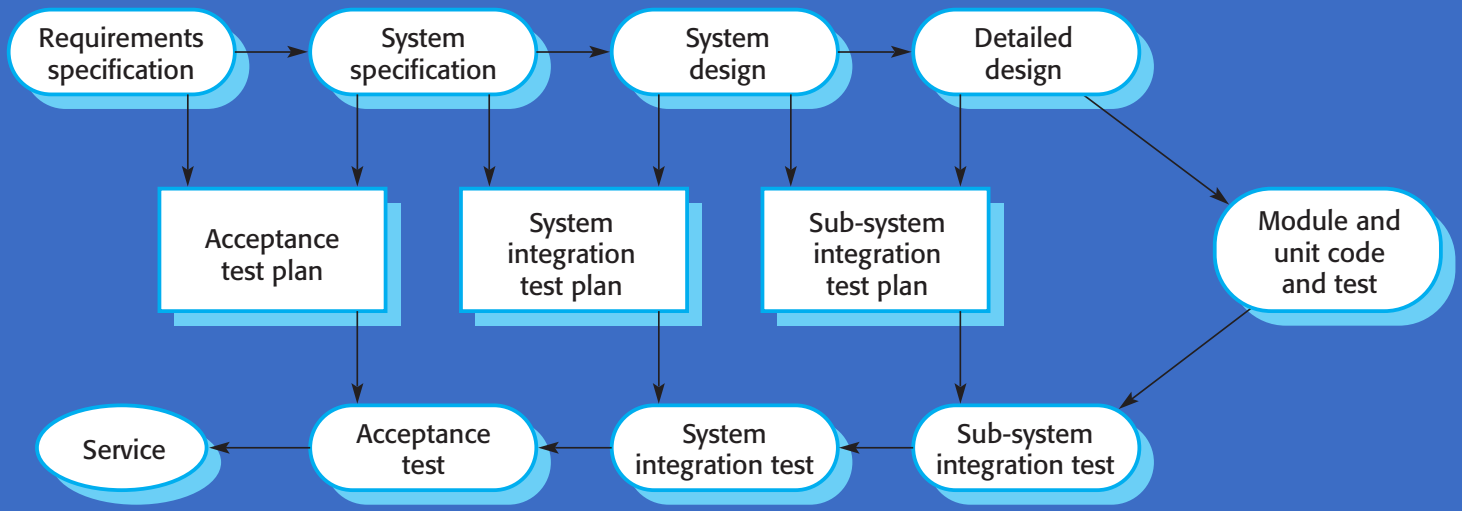
System testing

- Testing of the system as a whole. Testing of emergent properties is particularly important.

Customer testing

- Testing with customer data to check that the system meets the customer’s needs.

Testing phases in a plan-driven software process (V-model)



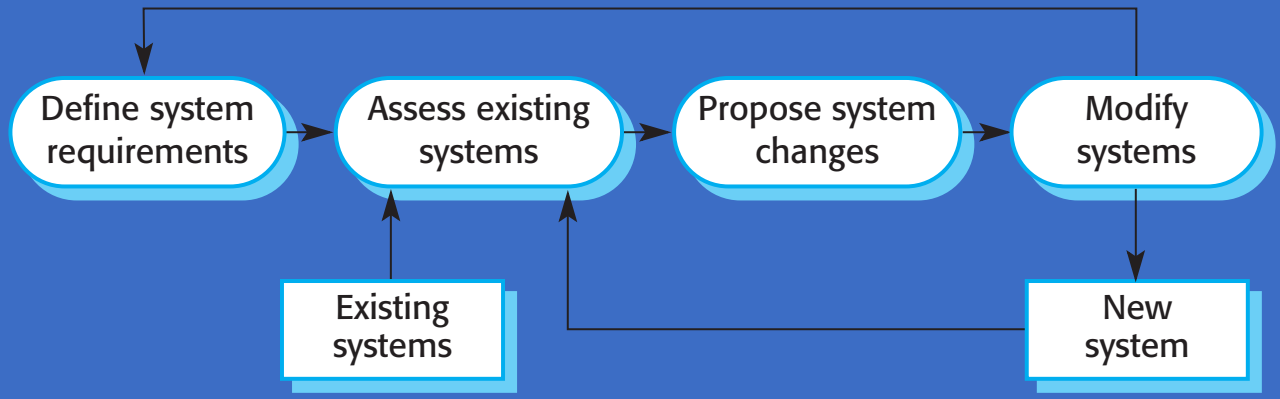
Software evolution

Software is inherently flexible and can change.

As requirements change through changing business circumstances, the software that supports the business must also evolve and change.

Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new.

System evolution



Coping with change

Coping with change

Change is inevitable in all large software projects.

- Business changes lead to new and changed system requirements
- New technologies open up new possibilities for improving implementations
- Changing platforms require application changes

Change leads to rework so the costs of change include both rework (e.g. re-analyzing requirements) as well as the costs of implementing new functionality

Reducing the costs of rework

Change anticipation, where the software process includes activities that can anticipate possible changes before significant rework is required.

- For example, a prototype system may be developed to show some key features of the system to customers.

Change tolerance, where the process is designed so that changes can be accommodated at relatively low cost.

- This normally involves some form of incremental development. Proposed changes may be implemented in increments that have not yet been developed. If this is impossible, then only a single increment (a small part of the system) may have be altered to incorporate the change.

Coping with changing requirements

System prototyping, where a version of the system or part of the system is developed quickly to check the customer's requirements and the feasibility of design decisions. This approach supports change anticipation.

Incremental delivery, where system increments are delivered to the customer for comment and experimentation. This supports both change avoidance and change tolerance.

Software prototyping

A prototype is an initial version of a system used to demonstrate concepts and try out design options.

A prototype can be used in:

- The requirements engineering process to help with requirements elicitation and validation;
- In design processes to explore options and develop a UI design;
- In the testing process to run back-to-back tests.

Benefits of prototyping

Improved system usability.

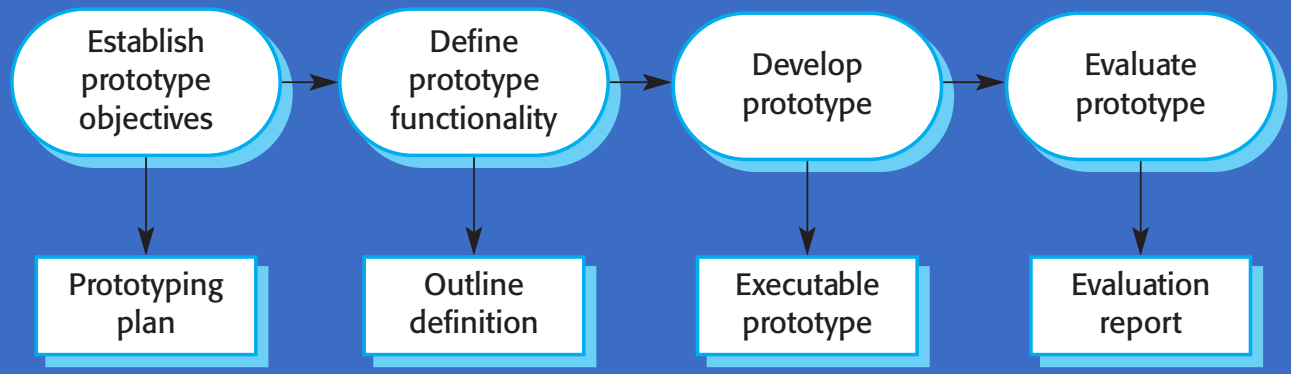
A closer match to users' real needs.

Improved design quality.

Improved maintainability.

Reduced development effort.

The process of prototype development



Prototype development

May be based on rapid prototyping languages or tools

May involve leaving out functionality

- Prototype should focus on areas of the product that are not well-understood;
- Error checking and recovery may not be included in the prototype;
- Focus on functional rather than non-functional requirements such as reliability and security

Throw-away prototypes

Prototypes should be discarded after development as they are not a good basis for a production system:

- It may be impossible to tune the system to meet non-functional requirements;
- Prototypes are normally undocumented;
- The prototype structure is usually degraded through rapid change;
- The prototype probably will not meet normal organisational quality standards.

Incremental delivery

Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.

User requirements are prioritised and the highest priority requirements are included in early increments.

Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.

Incremental development and delivery

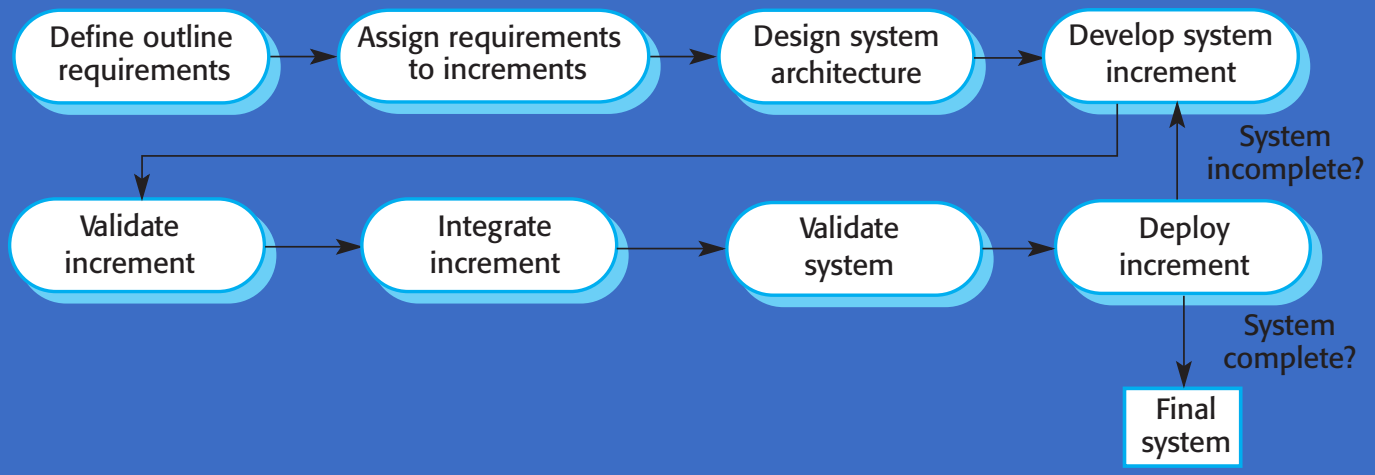
Incremental development

- Develop the system in increments and evaluate each increment before proceeding to the development of the next increment;
- Normal approach used in agile methods;
- Evaluation done by user/customer proxy.

Incremental delivery

- Deploy an increment for use by end-users;
- More realistic evaluation about practical use of software;
- Difficult to implement for replacement systems as increments have less functionality than the system being replaced.

Incremental delivery



Incremental delivery advantages

Customer value can be delivered with each increment so system functionality is available earlier.

Early increments act as a prototype to help elicit requirements for later increments.

Lower risk of overall project failure.

The highest priority system services tend to receive the most testing.

Incremental delivery problems

Most systems require a set of basic facilities that are used by different parts of the system.

- As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.

The essence of iterative processes is that the specification is developed in conjunction with the software.

- However, this conflicts with the procurement model of many organizations, where the complete system specification is part of the system development contract.

Bibliography

Ackoff, Russell Lincoln, Lauren. Johnson, and Systems Thinking in Action Conference. 1997. From mechanistic to social systemic thinking : a digest of a talk by Russell Ackoff. Cambridge, Mass.: Pegasus Communications. https://www.youtube.com/watch?v=yGN5DBpW93g.

Boehm, Barry W. 1988. A spiral model of software development and enhancement. Computer 21 (5): 61–72.

Buxton, John N, and Brian Randell. 1970. Software Engineering Techniques: Report on a Conference Sponsored by the NATO Science Committee. NATO Science Committee; available from Scientific Affairs Division, NATO.

Kim, Daniel H. 2000. Introduction to Systems Thinking.

McClure, Robert M. 1968. NATO SOFTWARE ENGINEERING CONFERENCE 1968.

Sommerville, Ian, and others. 2011. *Software Engineering*. Boston: Pearson,.