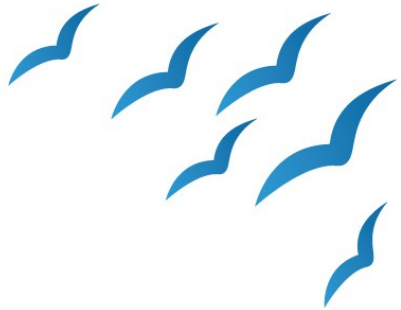
BE CSE-2 VII Semester FOSS

Unit-II.a- Fault Tolerant Design Principles and Open-source Methodology



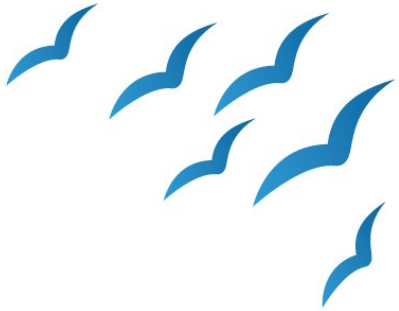
Unit-II topics

- Principles and Open-source Methodology
- History, Open Source Initiatives
- Open Standards, Principles
- Software freedoms
- Open source software development
- Licenses, Copyright Vs. Copyleft
- Patents
- Zero marginal cost
- Income generation opportunities
- Internationalization of Software

Open-source and Free Software

- Open-source is different from Free software
- Open-source philosophy is based on different values and its practical definition is different, but all open-source programs are in fact free software programs
- Free software respects users' freedom and community- users have the freedom to **run, study, copy & distribute, and change & improve the software**
- Users control the program and what it does for them
- When users don't control the program then it is called "**non-free**" or "**proprietary**" program
- Non-free program controls the users, and the developer controls the program which is an instrument of unjust power

"The Free Software Foundation (FSF) is a nonprofit with a worldwide mission to promote computer user freedom. We defend the rights of all software users."



Software freedoms

- Freedom-0: to run the program for any purpose
- Freedom-1: to study to study how the program works
- Freedom-2: to redistribute copies
- Freedom-3: to distribute copies of your modified versions to others
- Freedom-1, 2 precondition: source code
- A program is free software if it gives users adequately all of those freedoms, otherwise non-free, which is **unethical.**
- Free software doesn't mean "**noncommercial**", free program be available for commercial use, development, and distribution

Proprietary software is often Malware-FSF

- Proprietary software also called **non-free software**, doesn’t respect users’ freedom and community
- Puts its developer or owner in a position of power over its users, which an **injustice**
- Leads to malicious functionalities
- Power corrupts; tempts the developers to design program to mistreat its users (malware).
- Of course, the developer usually does not do this out of malice, but rather to profit more at the users’ expense. That does not make it any less nasty or more legitimate
- All modern proprietary software typically had these features

Some of the malicious functionalities

- Injustices of techniques
  - Additions, back doors, censorship, cover-ups, deception, DRM, fraud, incompatibility, insecurity, interference, jails, manipulation, obsolescence,sabotage, surveillance, tethers, tyrants, in the pipe
- Products or companies
  - Appliances, cars, games, mobiles, web pages, Adobe, Amazon, Apple, Google, Microsoft
- **Back door**: a feature that enables someone, who is not supposed to be in control of the computer where it is installed send it commands

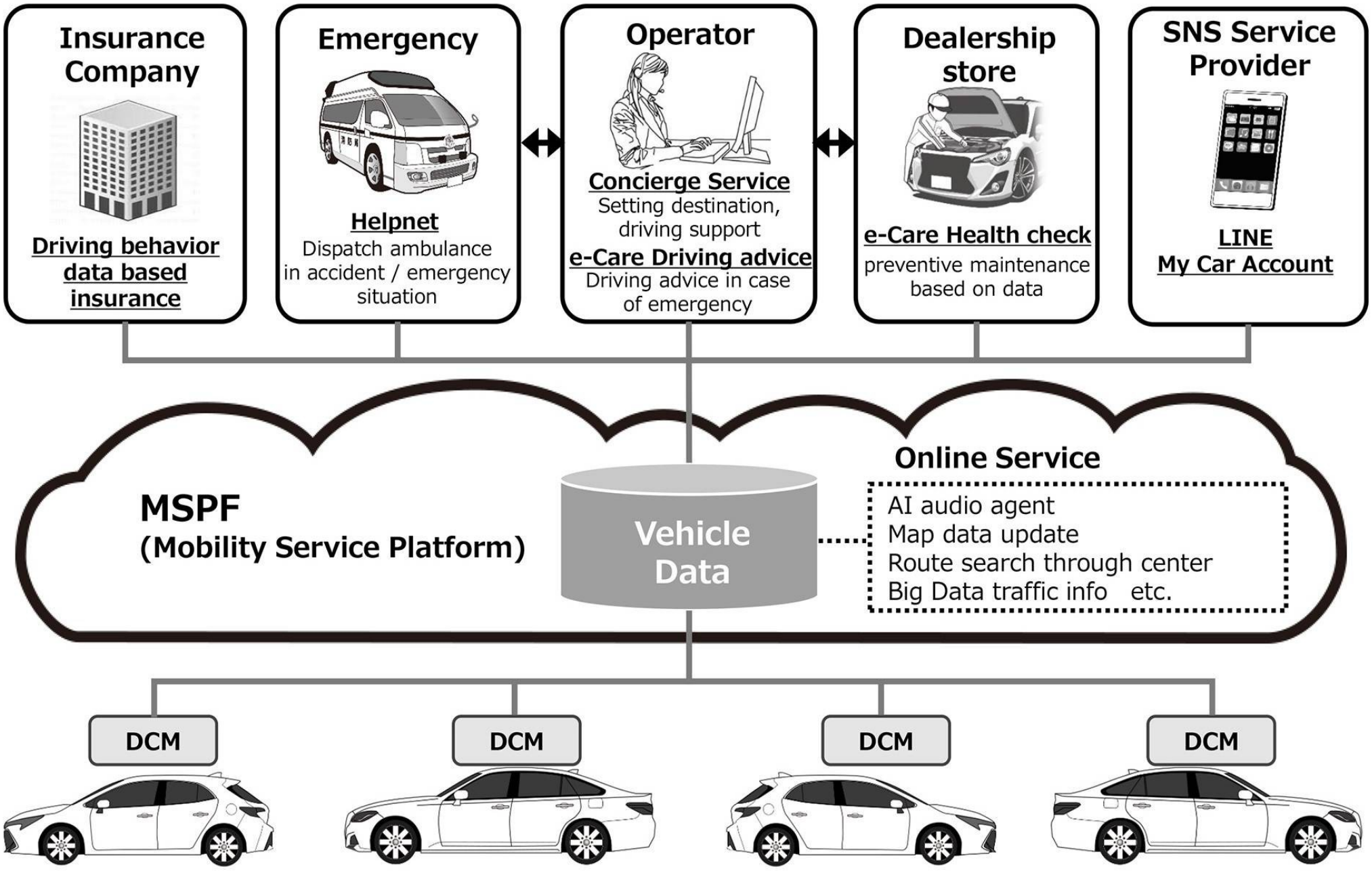
Some of the malicious functionalities

- ***DRM***: designed to restrict what users can do with the data in their computers
- ***Jail***: system that imposes censorship on application programs
- ***Tether***: functionality that requires permanent (or very frequent) connection to a server (Ex. Lumix Tether, TeamViewer)
- ***Tyrant***: system that rejects any OS not authorized by the manufacturer (Mac)
- Users of proprietary software are defenseless against these mistreatment
- Solution is **Free software** since it is controlled by users

Latest additions

- New Toyota vehicles upload data to AWS (car share, ride share, full-service lease, new corporate and consumer services line proactive vehicle maintenance notifications and driving behavior insurance) use Open Street Map based GPS
- TikTok exploited Android security Vulnerability (Mac Addresses used for advertisements)
- Epic Games- Apple App store cut off developer tools for developing tools for iOS or Mac OS (own payment gateway, 30% cut off)
- Thomas Le Bonneic: Apples's massive data collection of recording and data (Siri- Apple's vocal assistant)
- Google Nest talking over ADT – sent software update to its speaker devices using their back door that listens for alerts

https://regmedia.co.uk/2020/08/18/pr_download_toyota_connected_car_ecosystem.jpg



Principles of Open-source way

- Open-source way is set of principles derived from open-source software development models and applied more broadly to additional industries and domains
- Principles of open-source way
  - Transparency
  - Collaboration
  - Release early and often
  - Inclusive meritocracy
  - Community

History of Open Source Initiatives

- 1970-80’s several initiatives by academicians, researchers
- 1984, GNU project by Richard Stallman
- 1985, Free Software Foundation(**FSF**)
- **Aim:** Sharing and collaborative improvement of software
- Netscape had created an opportunity to educate and advocate for the technical superiority of an open source development process
- **Arguments:** Pragmatic, business-case grounds
- In 1998, **OSI** was launched by Jon Hall, Larry Augustin, Eric S Raymond, Bruce Perens and others. Linus Torvalds joined later.
- Open-source definition was based on Debian Free Software guidelines, but failed to create a trade mark for “open source”
- In Aprial 1998, many individuals inducing founding figures of Sendmail, Perl, Python, Apache, IETF, Internet S/W Consortium

History of Open Source Initiatives(cont.)

- Early period of OSS movement grown and become popular with the dot-com boom(1998-2000), Linux and many open source friendly companies
- Movement caught the attention of software industry, sponsoring by the established software companies like Corel (Corel Linux), Sun Micro-systems(OpenOffice.org), and IBM(OpenAFS)
- By the end of dot.com boom 2001, open source advocates had already fruit and the movement continued to further strengthen in the cost-cutting climate of the 2001-2003 recession
- **Actions:** 1. writing and spreading free software 2. building awareness, 3. legislation and 4. Internal conflict

Open Standards Principles

- An open standard is more than just a specification
- The principles behind the standard, practices of offering and operating of the standard makes the standard as Open Standard
- Bruce Perens created the open standards, principles and practices
- Open source principles
  - Availability, maximize end-user choice, no royalty, no discrimination, extension of subset, predatory practices, molecularity, and early & often

Open Standards Principles (cont.)

- Availability:
  - Available to all to read and implement
    1. Standard text and implementation must be made available for download
    2. Affordable (cost should not be more than text book)
    3. Licenses must not restrict any party implementing the standard using any form of software license
    4. Software reference platforms to be licensed in a way that is compatible with all forms of software licensing, both free software and proprietary
- Maximize end-user choice
  - Open standards create a fair, competitive market for implementations of the standard. They do not lock the customer in to a particular vendor or group
    1. Must allow a wide range of implementations, by business, academia, and public projects
    2. Must support a range of pricing, expensive to zero pricing

Open Standards Principles (cont.)

- No royalty
  - Open standards are free for all to implement, with no royalty or fee. Certification of compliance by the standards organization may have a fee, Thus
    - Patents embedded in standards must be licensed royalty-free, with non discriminatory terms
    - Certification programs should include a low or zero cost self-certification, but may include higher-cost programs with enhanced branding
- No discrimination
  - Open standards and organization that administer them do not favor one implementer over the other
  - Certification organizations must provide a path for low and zero-cost implementation

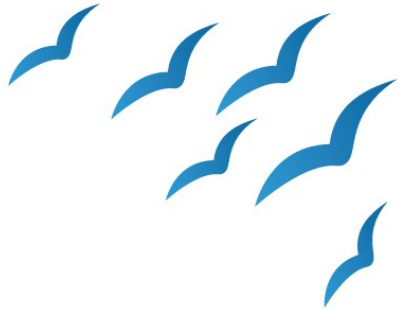
Open Standards Principles (cont.)

- Extension of Subset
  - Implementations of Open Standards may be extended, or offered in subset form
  - Certification organizations may decline to certify subset implementations and may place requirements upon extensions
- Predatory Practices
  - Open Standards may employ license terms that protect **sub-versions** of the standard by embrace-end-extend tactics
  - The license may require the publication of **reference** info. and license to create and redistribute S/W compatible with the extensions
  - It may not prohibit the implementation of extensions

Open Standards Principles (cont.)

- Modularity
  - It is the principle of maintaining, well-defined boundaries between components;
- Early and Often
  - Helps to make the things as early as possible
  - It includes not only software but also plans and design
  - Many open source projects the common methods of
    - Source code availability (daily as zip/cvs/sub versions - community have to compile or build)
    - Milestones builds (periodic basis, the admins or core developers of the project compile and make the resulting executable for download)
    - Nightly builds (nightly basis source code is compiled to generate the executable binaries. Build are made available for community members who wants to test the latest source code)

Review Questions



- What is software Engineering?
- What are the different software development methodologies?
  - Open source and closed source software development
- What are the different activities involved in software development?
  - Requirements, analysis, design, implementation, testing, deployment and maintenance
- What are the different software development models?
  - Agile, Clean room, Iterative, Spiral, XP, Waterfall, Scrum, RAD, RUP

Software Development Methodologies

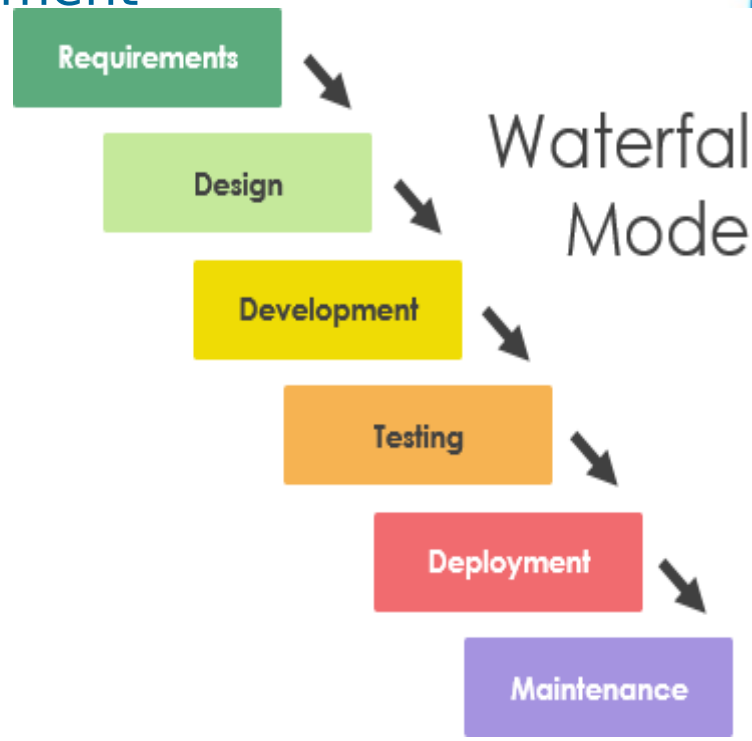
- Is a **structure** imposed on the development of a software product
- It includes procedures, techniques, tools and the documentation aids
- Helps system developers in their task of implementing a new system
- The intent of a methodology is to **formalize** what is being does and making it more repeatable
- Types of methodologies
  - Software Engineering Methodology
  - Open source methodology

Software Engineering Methodologies

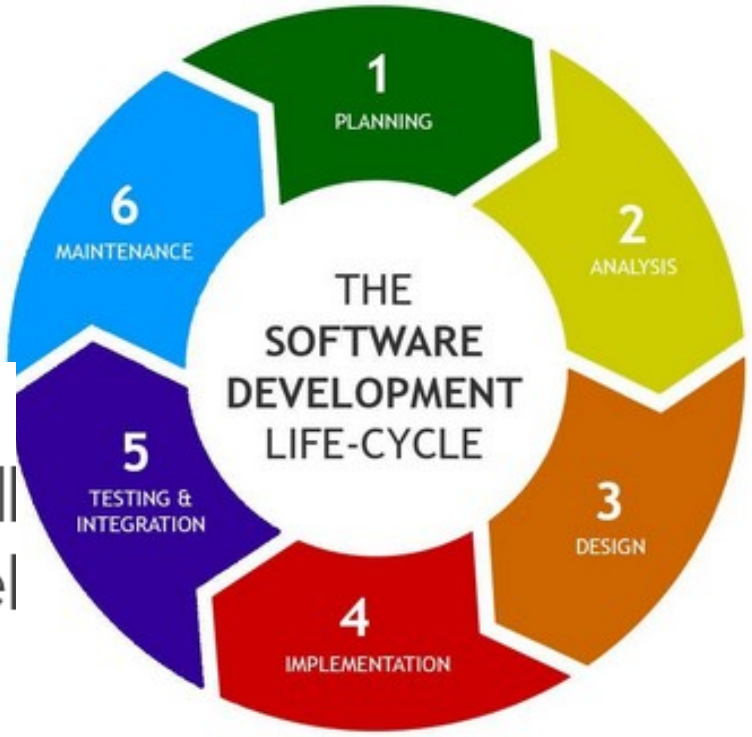
- Engineering Software
  - Software engineering methodologies are the **frame works** that tell us how to develop software systems.
  - Frame works define different phases of the development processes such as **planning, requirements analysis, design, implementation, testing and maintenance**
  - Choice of the frame work depends on the size of the system and the environment
  - Paradigms
    - Life-Cycle Paradigm
    - Prototyping Paradigm
    - Spiral Model Paradigm

Life Cycle Paradigm (Waterfall Model)

- It includes six activities
  1. System Engineering & Analysis
  2. Software requirements analysis
  3. Design
  4. Coding/implement
  5. Testing
  6. Maintenance (deployment support)



Waterfall Model



Life-Cycle Paradigm

Fig. 2.1. Life Cycle Paradigm.

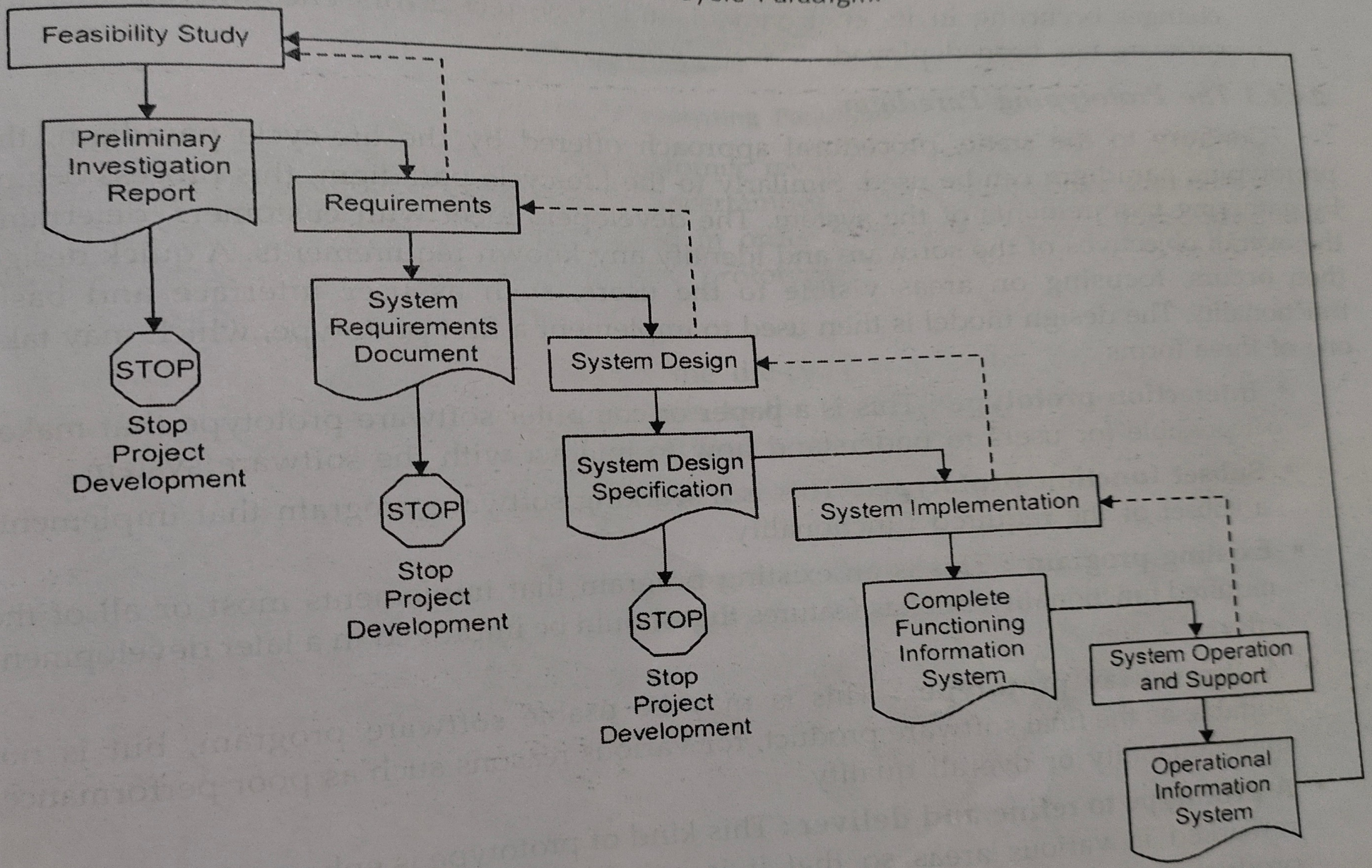


Fig. 2.2. Waterfall Model.

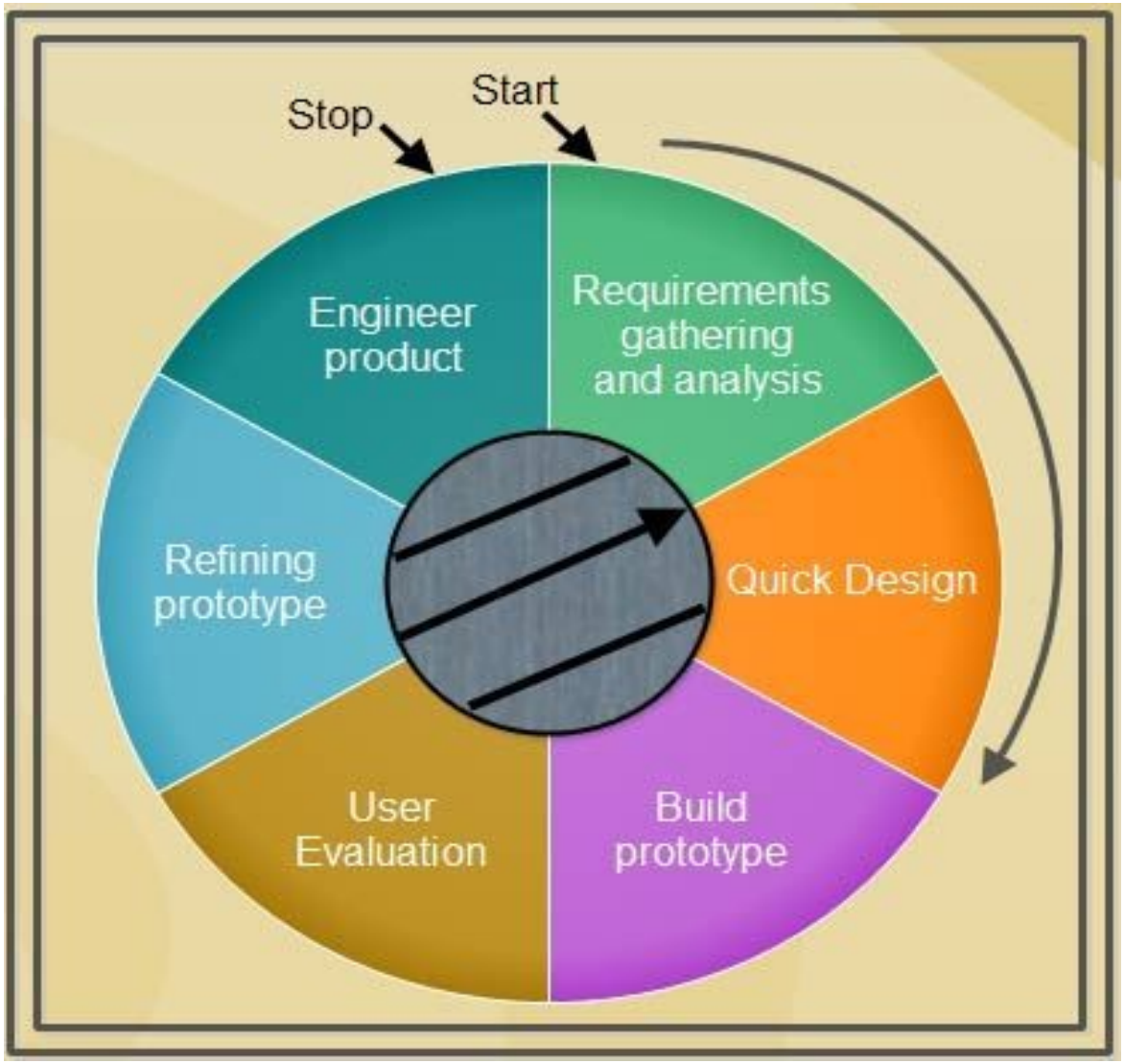
Prototyping Paradigm

- Prototype model may be applied,
  - When the detailed information related to **input and output** requirements of the system is not available
  - When **all the requirements** may not be known at the start of the development of the system
  - When a system **does not exist** or in case of a large and complex system where there is no manual process to determine the requirements
- User interact and experiment with a working model of the system and gives the user an actual feel of the system
- At any stage, if the user is not satisfied with the prototype, it can be discarded and an entirely new system can be developed

Prototyping Paradigm(cont.)

- Initial prototype
  - Create user interfaces without any substantial coding
  - Abbreviate a version of the system that will perform a limited subset of functions (Wireframing tools)
  - Illustrate the functions of the system using the components
- Prototype is developed based on the currently known requirements
- Development of prototype obviously undergoes with **design, coding, & testing**, but each of these phases is not done very formally or thoroughly
- Client can get an actual feel of the system, because the interactions; better understanding about the requirements

Prototyping Paradigm

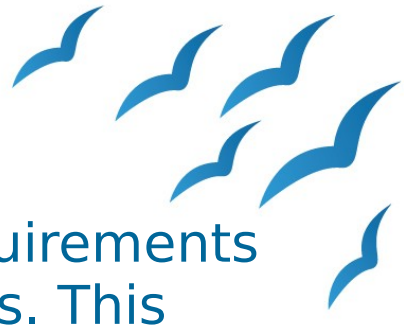


https://ecomputernotes.com/software-engineering/explain-prototyping-model

Prototype can be any one of the following

- Interaction prototype
- Subset function prototype
- Existing program
- A throwaway prototype
- A prototype to refine and deliver

Prototyping Paradigm(cont.)



- Model starts with building prototype with initial requirements and collects feedback from the user and then refines. This cycle repeated until quality production system is developed

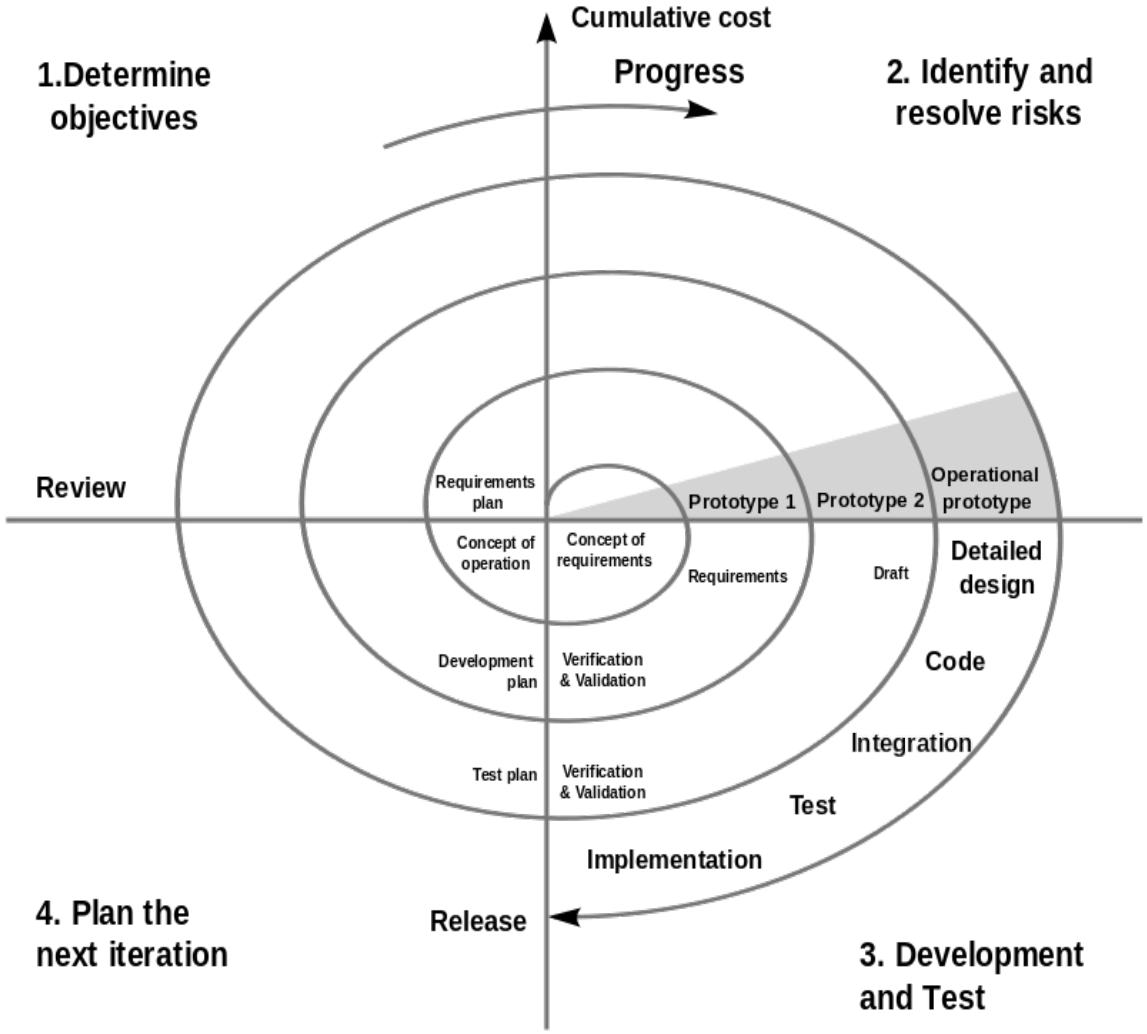
Steps in prototype model

1. Requirements gathering and Analysis -(interview user to collect requirements and then analyze)
2. Quick Design/Preliminary design- (not a detailed design, includes important aspects to give an idea to user and helps to develop the prototype)
3. Build Prototype -(info. Gathered from quick design is modified for representing the working model)
4. User Evaluation- (present the model for the evaluation -strengths and weaknesses, collect comments and suggestions and provide to the developer)
5. Refine Prototype- (refine with additional info. Given by the user; this process is continued until user satisfies)
6. Engineer product- final user accepted system is evaluated thoroughly followed by the routine maintenance on regular basis for preventing large-scale failures and minimize down time)

Spiral Model

- Spiral model adds risk analysis and RAD prototyping to the waterfall method
- Each cycle involves the same sequence of steps as waterfall model
- Major activities in spiral model are
  - **Planning** :Determine objectives and constraints of the project and define the alternatives
  - **Risk Analysis**: Analysis of alternatives and identification/resolution of risk
  - **Engineering**: Development of the “next-level” product
  - **Customer Evaluation**: Evaluation of the product engineered

Spiral Model (cont.)



- **Objectives**: functionality, performance, hardware/software interface,critical success factors etc.
- **Alternatives**: build, reuse, buy, sub-contract etc.
- **Constraints:** cost, schedule, interface etc.
- **Next-level product** activities: create design, review design, develop code, inspect code, test code
- **Plan next Phase** activities:develop project plan, configuration plan, test plan and installation plan

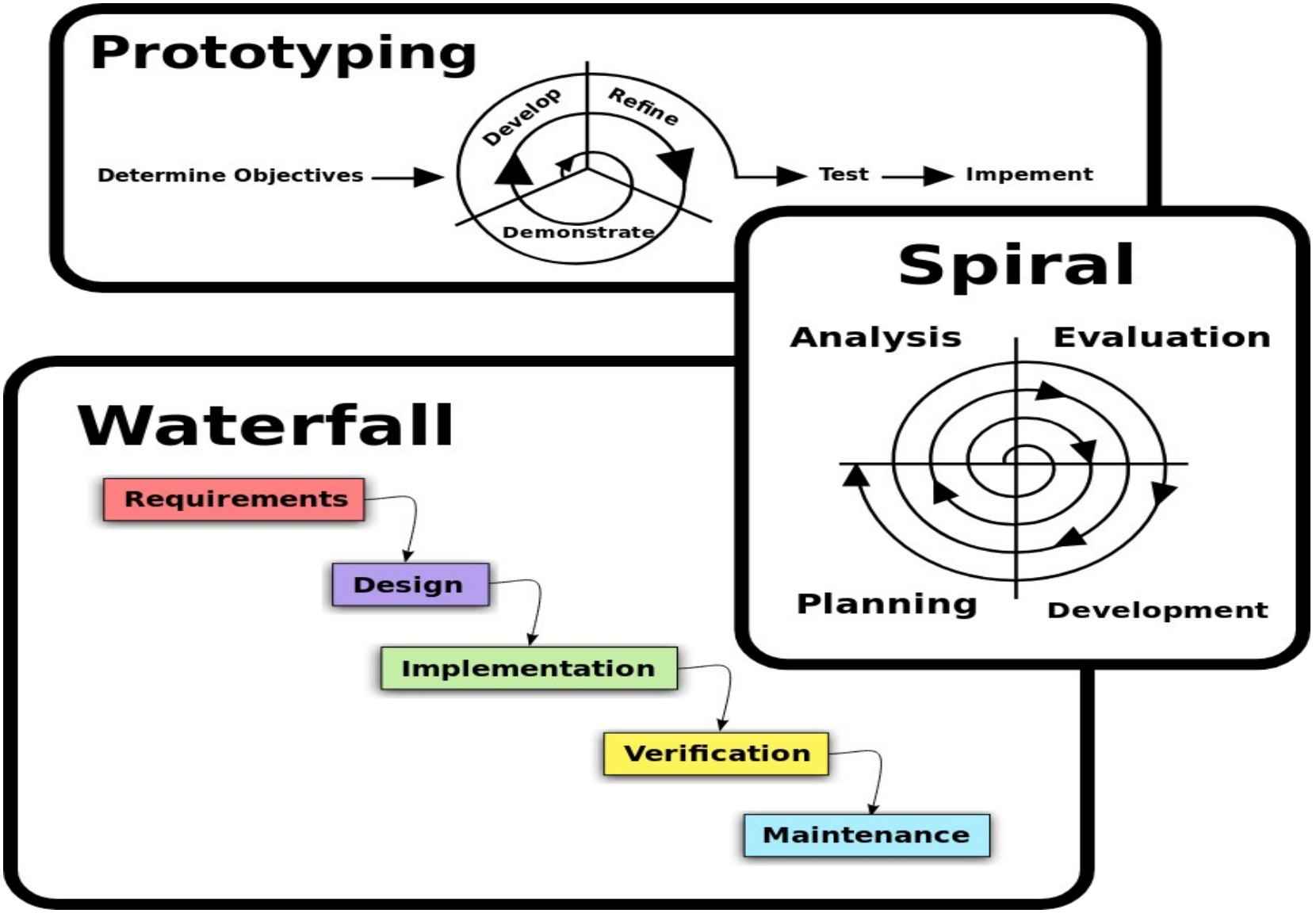
https://en.wikipedia.org/wiki/Spiral_model

https://www.tutorialspoint.com/adaptive_software_development/sdlc_spiral_model.htm

Strengths and weaknesses

- **Strengths**
  - Provides early indication of risks without involving much costs
  - User can see the system early (RAD tools)
  - Critical high-risk functions are developed first
  - Design does not have to be perfect
  - Users can closely involved in all life-cycle steps
  - Early and frequent feedback from users
- **Weaknesses**
  - May be difficult to define objectives, milestones
  - Time overhead in planing, resetting objectives, risk analysis and prototyping and too large for small and large projects
  - Complex to understand new team members
  - Risk assessment expertise is required
  - Spiral model may continue indefinitely
  - Developer must be reassigned during non-developmental phase activities

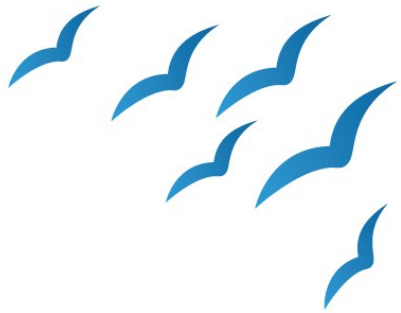
Common methodologies (1980-90s



Agile Methodologies

- Based on agile manifesto and adaptive in nature. Agile methodologies ensure
  - Team collaboration
  - Customer collaboration
  - Constant and continuous communication
  - Response to changes
  - Readiness of a working product
  - Several agile methods came into existence, promoting iterative and incremental development

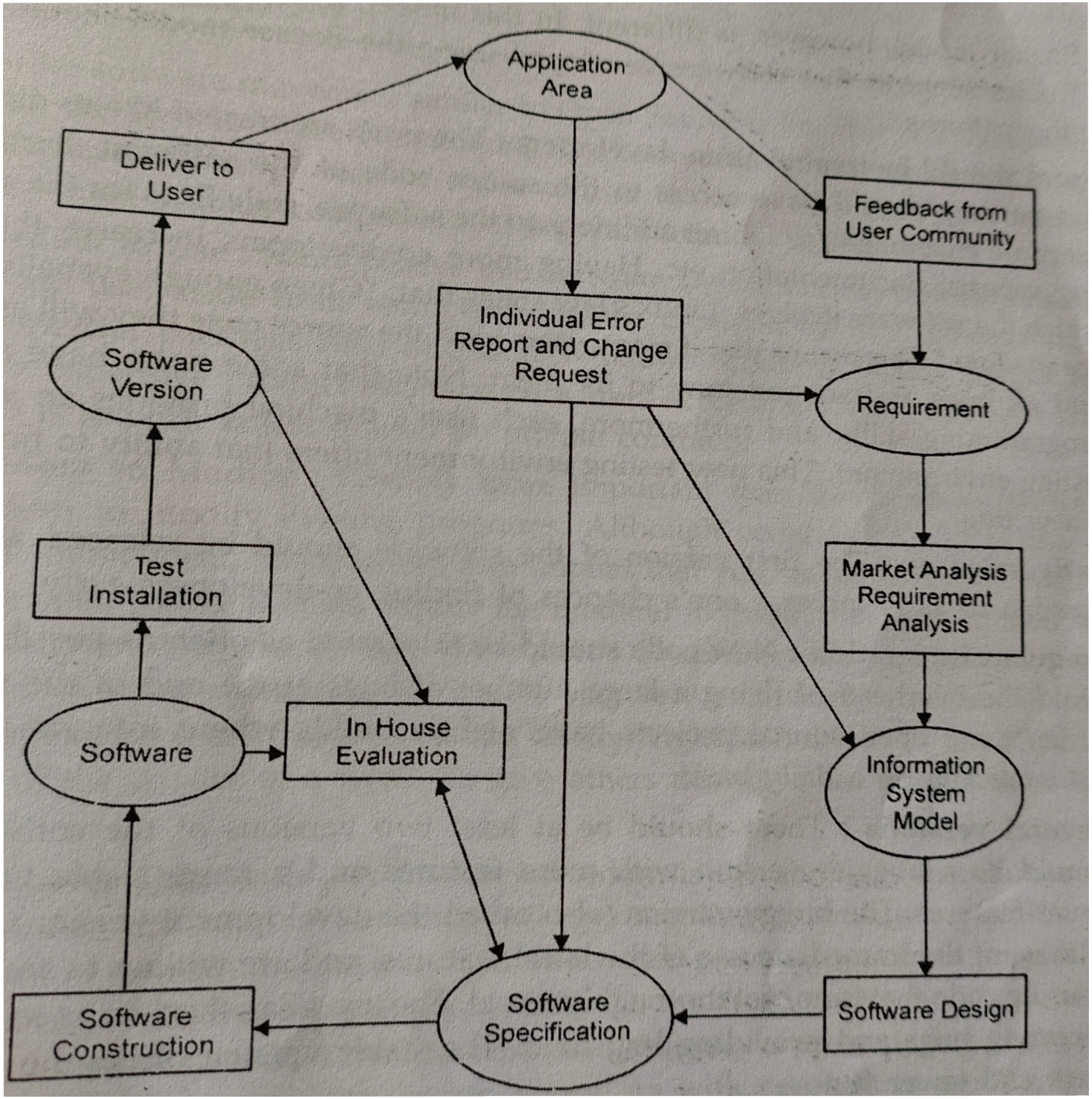
Strengths and Weaknesses



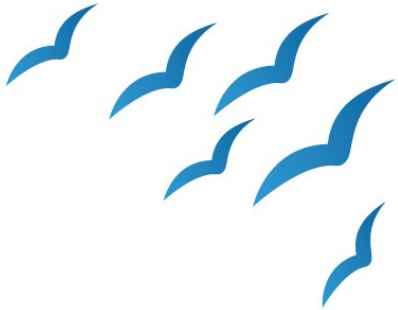
- Strengths
  - Early and frequent releases;
  - Accommodation of changing requirements
  - Daily communication among the customer and developers
  - Project built around motivate individuals; Self-organizing teams
  - Simplicity, focusing on what is immediate requirement; no building for future or overburdening code
  - Regular reflection to adjust behavior to improve effectiveness
- Weaknesses
  - Customer availability may not be possible
  - Team should be experienced to follow rules, expected to have estimation, communication and negotiation skills,
  - Appropriate planning is required to decide upon the functionalities
  - Teams may not be able to organize themselves, design needs to be simple, and main tenable

Open source Methodology

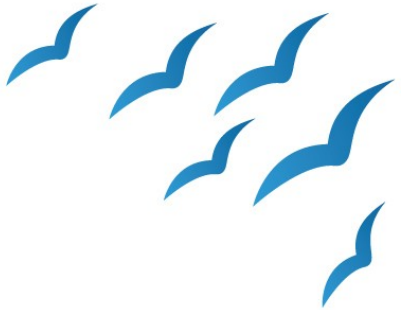
- Software is created by a community of programmers rather than a single vendor
- Ex: Mozilla Web Browser, source code was created by programmers from different organizations and made freely available to others to modify and copy as they wish
- Open source software has become popular and many for profit corporations are adapting open source products into their business models


Application Area
Deliver to User
Feedback from User Community
Individual Error Report and Change Request
Software Version
Requirement
Test Installation
Market Analysis Requirement Analysis
Software
In House Evaluation
Information System Model
Software Construction
Software Specification
Software Design

Open-source Software Development



- **Software development process**
  - **Activities and steps**
    - Requirements, Architectures, Design, Implementation, Testing, Deployment
  - **Models**
    - Agile, Clean room, Iterative, RAD, RUP, Spiral, Waterfall, XP, Scrum
  - **Supporting disciplines**
    - Configuration management, Documentation, Software Quality Assurance(SQA), Project management, User experience design



Questions

- What are the different open source development tools?
- Bug tracking tools?
- Testing tools?
- Packaging tools?
- Automation tools?

Open-source Software Development

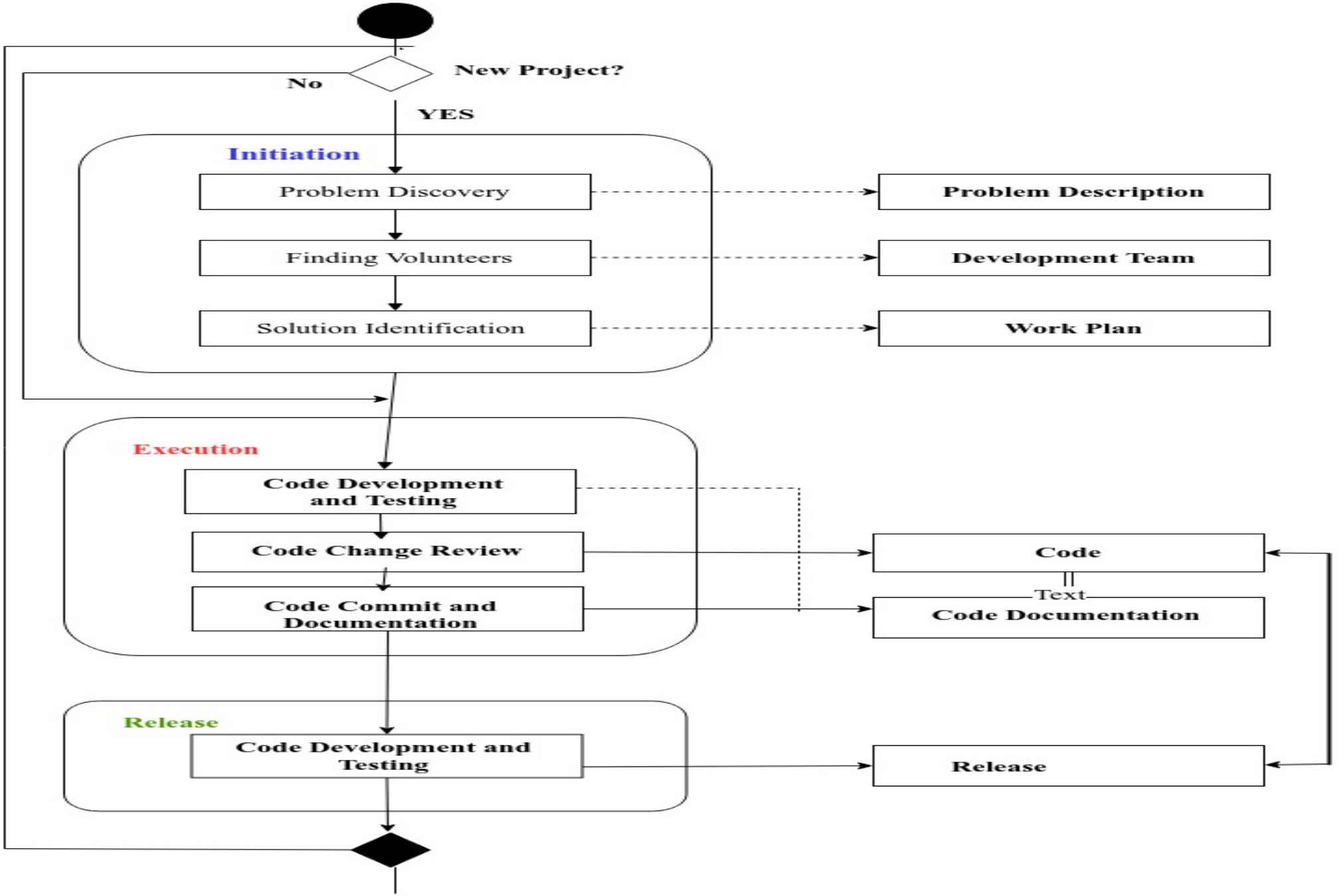
- Open-source development is the process by which open source software (similar software whose source is publicly available) is developed
- Open source products “available with its **source code** and under an **open source license** to use for any **purpose, study,, change & improve its design**”
- Ex: *Mozilla Firefox, OpenOffice Suite* etc.
- Open source software development is unstructured, because no clear development tools, phases etc. have been defined like DSDM
- Every project ha sits own phases, but there are generalities between Open source projects

Open-source Software Development

- Open source philosophy revolves around the opportunities for a **new software development model**
- Open source is about **sharing ideas** and **spreading the effort** of creating software over a large number of interested developers
- Sharing can lead to better software in **shorter development** *times* and also allows for **better feedback** directly to the developers than the traditional development methods
- Major companies like *Apple* (Darwin), *IBM, Microsoft*(MS Visual Code) and other companies recognized this method is a significant one
- IBM donated **software** and **patents** to open source groups

Open-source Software Development

- Open source software development can be divided into several phases, broadly initiation, execution and releasing
- Steps in Open source software development
  - Starts with a choice between the adopting of an **existing project** or starting of a **new project**
  - If an existing project is adopted, the process goes directly to the execution phase. Otherwise goes with problem description, identifying a collaborative development team, preparing a view plan followed by the execution phase
  - In the **execution phase**, if code changes are required then incorporated and then enters the release phase after code commit
  - Software will be **released** after developing and testing along with source code


New Project?
No
YES
Initiation
Problem Discovery
Finding Volunteers
Solution Identification
Problem Description
Development Team
Work Plan
Execution
Code Development and Testing
Code Change Review
Code Commit and Documentation
Code
Text
Code Documentation
Release
Code Development and Testing
Release

History: Open Source Development

- 1997, Eric S Raymond, in his essay "Cathedral and the Bazar"
- As per Raymond,
  - The traditional software development is like a cathedral "carefully crafted by individual wizards or small bands of mages working in splendid isolation"
  - bazar model "great babbling bazaar of different agendas and approaches"

- Cathedral model
  - **central planning, tight organization** and **one process** from start to finish
  - In closed source software development, programmers are often spends a lot of time dealing with and creating bug reports as well as handling feature requests, not on actual development
  - Also, involves on management related tasks such as deadlines, budget, technical issues of the software; where as in open source development users do this

Types of Open Source Projects

- Standalone Programs (pieces of code)
  - Some program might dependent on other open source projects
  - Serve a specified purpose and fill a definite need
  - Ex: Linux kernel, Firefox, OpenOffice tool etc.
- Distributions
  - Collections of software, published from the same source with a common purpose and there are in large number
  - Ex: Debian, Fedora Core, Mandriva, RedHat etc.(Linux kernel with apps); ActivePerl, OpenCD, Cygwin for Windows
- Books/document projects- not shipped as part of the product- Linux Documentation project

Starting an Open Source Project

- Several ways
  1. An individual who senses the need for a new project announces the intent to develop the project to the public
  2. A developer working on a limited but working code-base, release it to the public as the 1st version of an open source program and continue to work on improving it and possibly is joined by other developers
  3. The source code of a mature project is released to the public, after being developed as proprietary software or in-house software
  4. A well established open-source project can be forked by an interested outside party. Several developers can then start a new project, whose source code then diverges from the original
- It's a common mistake to start an own project when contributing to an similar existing project, would be more effective (NIH syndrome)
- To start a successful project, it is very important to ***investigate*** what's already there

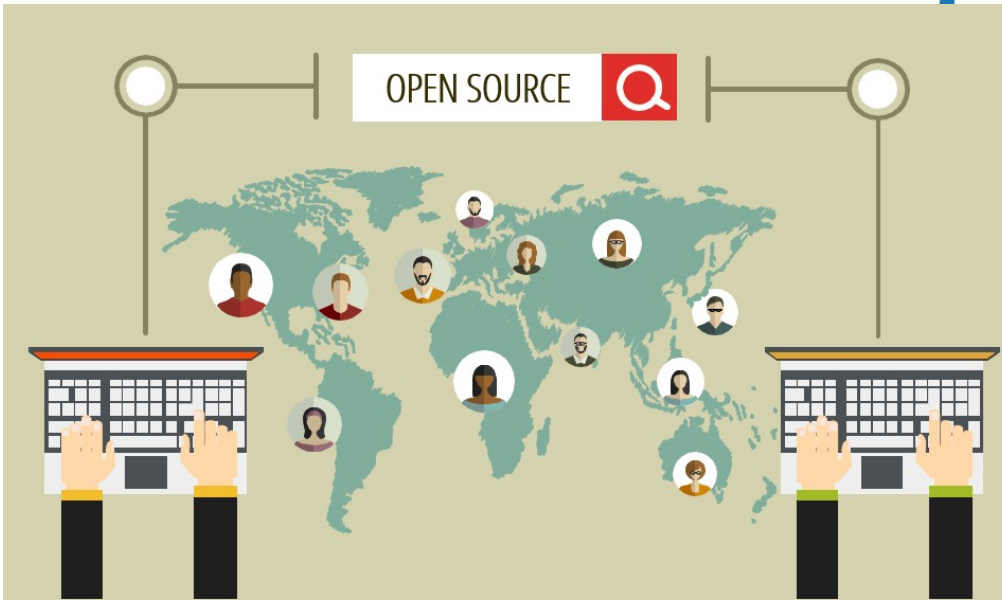
Who are the participants in OSS Projects?

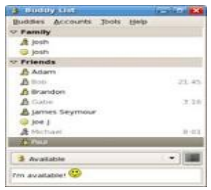
- Core- developers who modify the primary code that constitutes the project
- Peripheral- users who control the report bugs, submit fixes and suggests changes
- Various type of participants

1. **Project leaders** – core team, involves in coding, release
2. **Volunteer developer** – (core/peripheral) do actual coding for the project. Include senior members with broader overall authority, peripheral developer producing and submitting code fixes, occasional contributors, maintainers.
3. **Every day users**- perform testing, identify & report bugs etc.
4. **Posters** -participate frequently in newsgroups and discussions, but do not do any coding

Tools used for Open Source Development

- **Open source community**- Free Software Commons, Co-Developer Community, Developer-Developer Community, User Community
- All the people who are working on the same project need not be in proximity
- Communication Channels
  - E-mailing lists
  - Instant messengers like IRC. Riot Pidgin, Tox, etc
  - Web forums–(problems, solutions)
  - Wikis - (developer & users)
  - Videoconferencing – Open Meeting, Jitsi, BBB, MatterMost etc
  - Blogs, Micro blogs



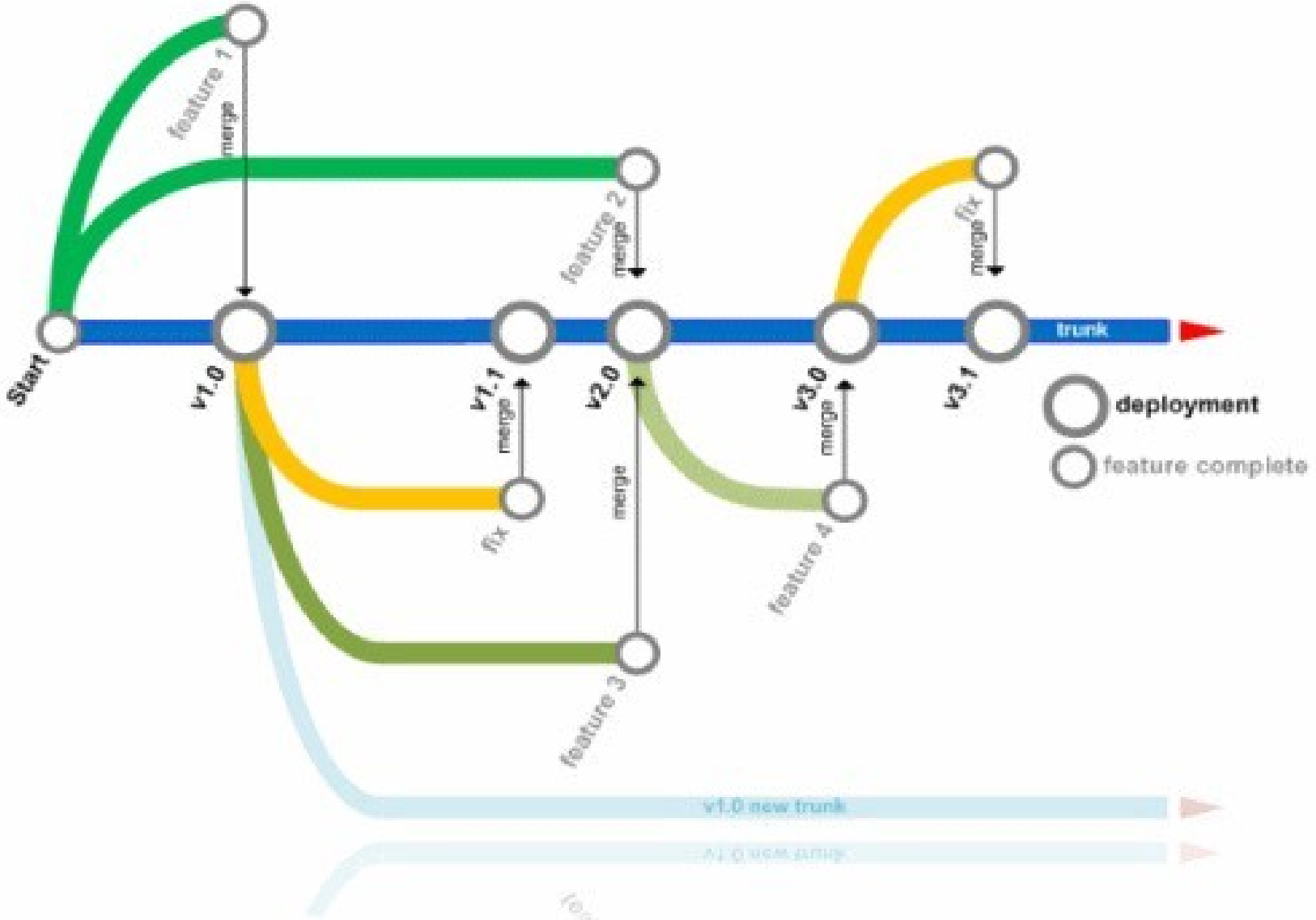


OSSD- Software Engg. Tools

- Version Control Systems (VCS)-
  - Allow developers to work simultaneously
  - Do not overlap each others changes
  - Maintain history of every version
- Some OSS VCSs
  - CVS-Concurrent Versions System
  - SVN Subversion Revision Control
  - RCS- Revision Control System
  - Git, GitHub, GitLab
  - CVSNT
  - Mercurial SCM
  - GNU Bazar etc.
- Proprietary VCSs
  - MS Azure, Teams, IBM Rational, AWS etc.

git commit -a

Branch and Merge




github
SOCIAL CODING
1.8m repositories
source
forge
260k projects
Google code
250k projects
bitbucket
108k repositories
launchpad
26.5k projects
CodePlex
30k projects
Java.net
RUBYFORGE
9.5k projects
The Central Repository
Apache
LINUX
mozilla
250 projects

OSSD-Bug Tracking Tools and Tasks

- Most large-scale projects require a bug tracker (web or other internet based)
- Simple text files are not sufficient, because they have many such bubs and because they wish to facilitate reporting and maintenance of bugs by users and secondary developers
- Some popular bug trackers are:
  - Bugzilla – a sophisticated web-based bug tracker from Mozilla
  - Mantis – a web-based PHP/MySQL bug tracker
  - Trac – bug tracker with Wiki and an interface to the CVS
  - Request Tracker -
  - GNATS-
  - SourceForge -bug tracker part of its services,
  - libreOffice

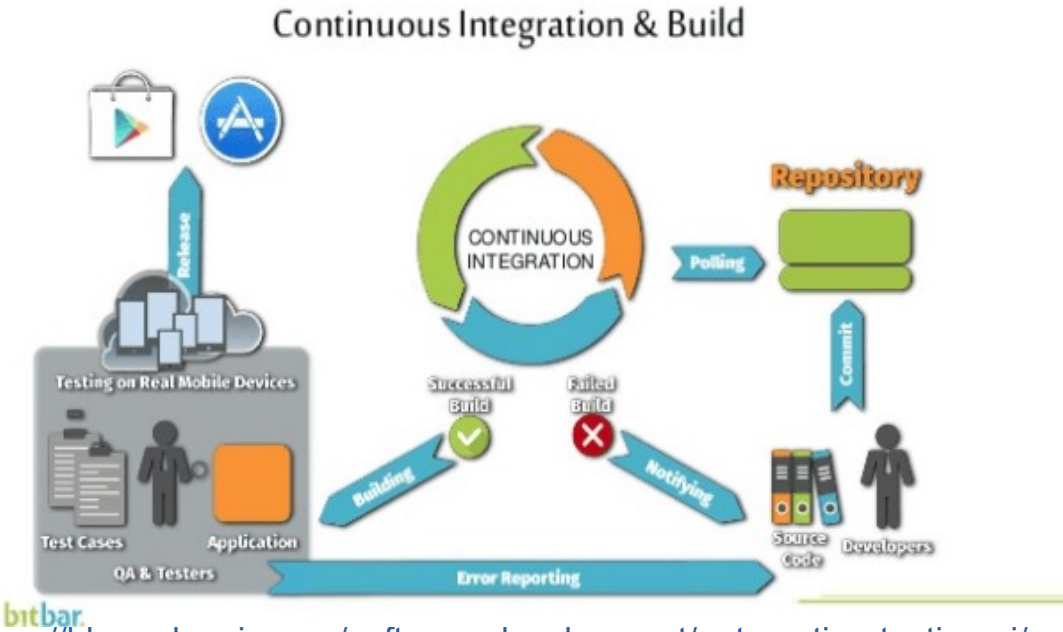
OSSD- Testing Tools

- Open-source projects undergo frequent integration
- Tools helps to automate testin during system integration
- Ex: Tinderbox
  - enables participants to detect errors during system integration
  - Runs a continuous build process and informs users about the parts of sourc code that have issues and on which platform(s) these issues arise
- Debugger- GNU gdb
- Memory leak tool or memory leak debugger

https://ec30.org/skills/selenium-testing/

Commercial and Open-source automation Testing Tools

- Selenium -2000 on wards, cross-platform
- Katalon Studio -powerful automation tool works with mobile, web services and applications
- UFT- Unified Functional Testing- comprehensive, mobile, GUI and API
- IBM RFT- wide range of applications- Ajax, Flex, SAP, .Net and Java
- Continuous Integration Tools
  - **DevOps**
  - **Jenkins-** Opensource Oraclae Sun Microsystems
  - **TeamCity**
  - **Travis CI**
  - **Go CD**



https://blog.eduonix.com/software-development/automating-testing-ci/

Package Management and Automatic Tests

- Package Management
  - A package management tool is a collection of tools to automate the process installing, upgrading, configuring and removing software from a computer
  - Ex: Redhat Package Manager(rpm), Advanced Packaging Tool(apt), Debian Package Manager (dpkg), Yellowdog Updater (yum)/DNF, Ubuntu Software Center etc.
- Automation Tests
  - Software testing is an integral part of software development
  - One can write scripts and programs that run the software and try to check whether it behaves correctly or not
  - Extream program based packages includes automated tools
  - Most of OSS provides command based or API to test

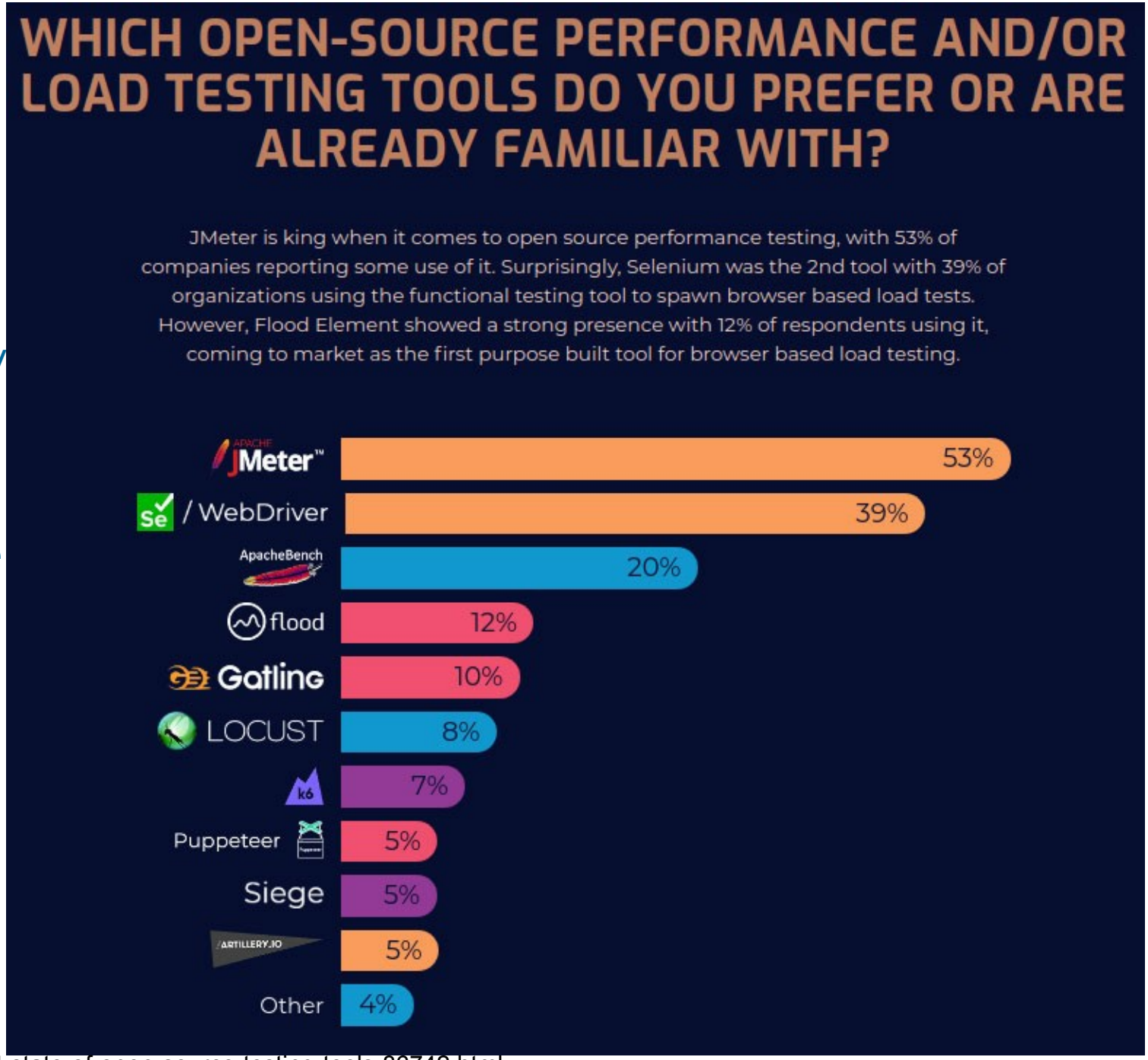
apt-get

yum
yellowdog updater modified

DNF

Current status of OSS testing tools

- By Germain Jack M. July 7, 2020 article, based on the survey reports of Tricentis and three survey organizations
- 30% of the software companies survey on on open source testing tools
- Most of the people with lack of skills
- 65% on premises
- 35% in pipeline



https://linuxinsider.com/story/the-current-state-of-open-source-testing-tools-86742.html

Approximately 37% of organizations had functional test automation integrated into their build pipeline, to assume some functional testing was completed with each and every build. When paired with the number doing functional test automation with each sprint (40%), more than 75% of organizations were doing functional automation testing at least every 2 weeks.

4%
Never

9%
Rarely e.g. only in response to particular issues

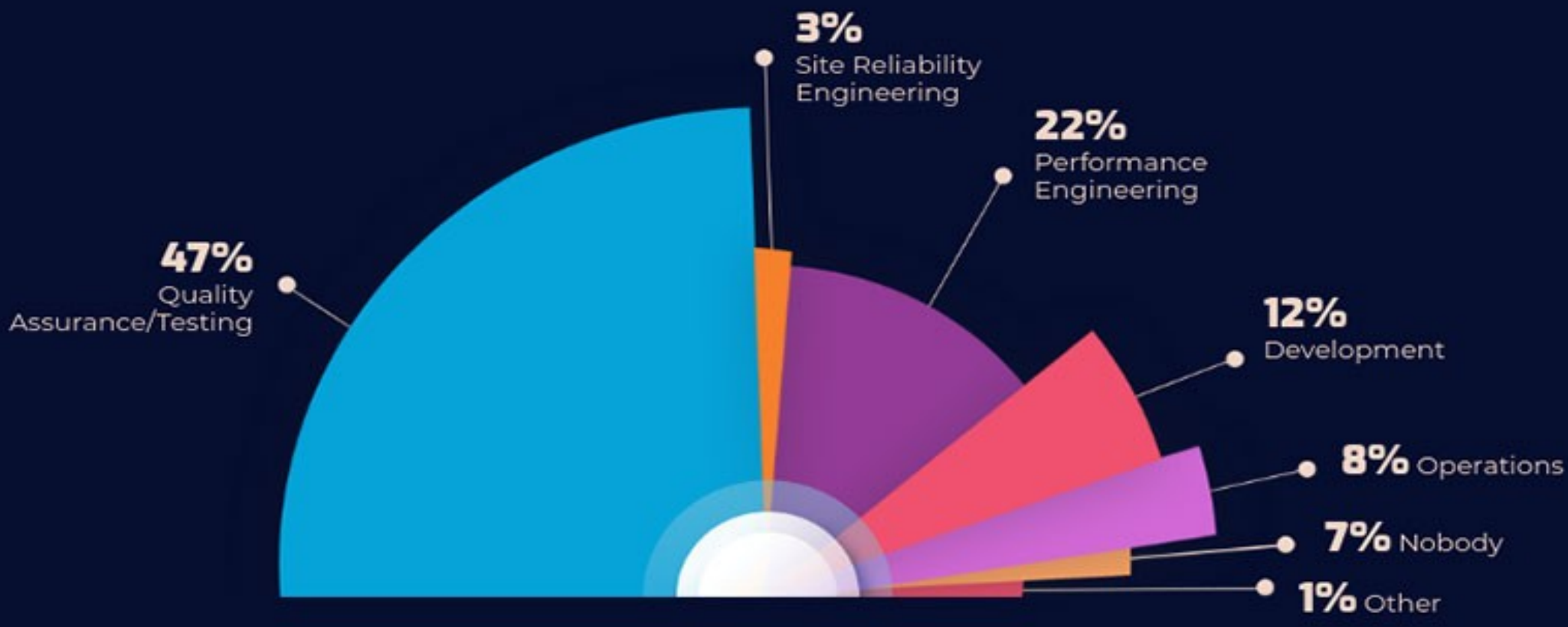
10%
Infrequently e.g. linked to scheduled test efforts

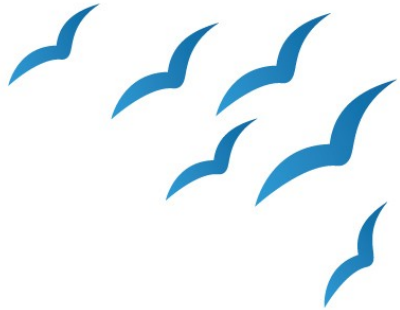
37%
Very frequently e.g. linked to continuous integration

40%
Frequently e.g. linked to agile methodologies

WHO IS RESPONSIBLE FOR PERFORMANCE AND/OR LOAD TESTING IN YOUR ORGANIZATION?

Interestingly, only 33% of companies surveyed had specifically tasked performance engineers (or similar Operations, Site Reliability Engineering) with doing performance testing. More often, Quality Assurance and Testers were responsible for functional testing as well as performance testing. A higher proportion of developers were reported to be responsible for performance testing than for functional testing, which was a large surprise.





References

- https://www.gnu.org/philosophy/free-sw.html
- https://opensource.com/open-source-way
- https://www.theregister.com/2020/08/18/aws_toyota_alliance/
- https://www.politico.eu/wp-content/uploads/2020/05/Public-Statement-Siri-recordings-TLB.pdf
- https://blog.google/products/google-nest/partnership-adt-smarter-home-security/
- https://backlog.com/blog/git-vs-svn-version-control-system/
- https://www.tricentis.com/