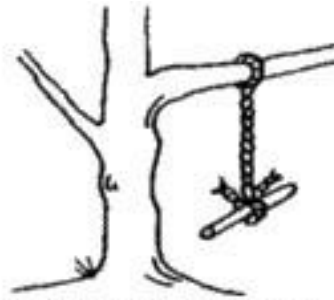


# Documentation & Maintenance

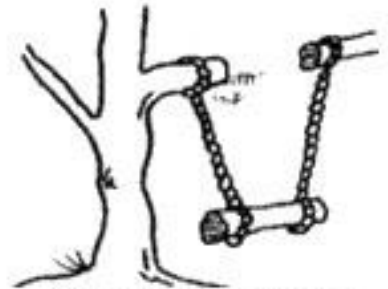
Documentation

# Why documentation?

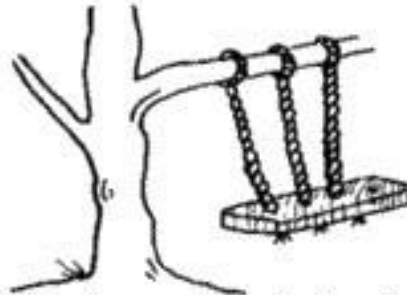
1. Facilitates communication
  - Within the development team itself
  - Between the development team and the project management
  - With customer
2. Records contracts and agreements



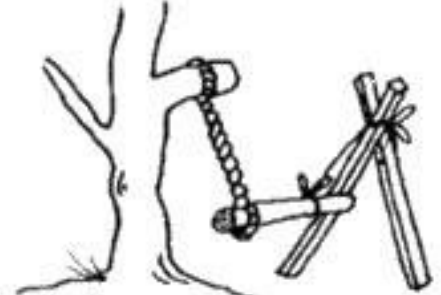
What the user asked for



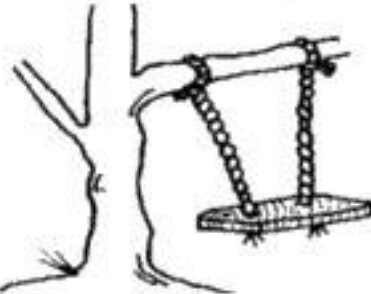
How the analyst saw it



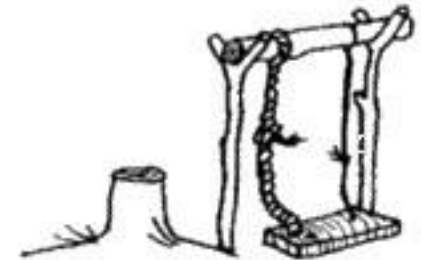
How the system was designed



As the programmer wrote it



What the user really wanted



How it actually works

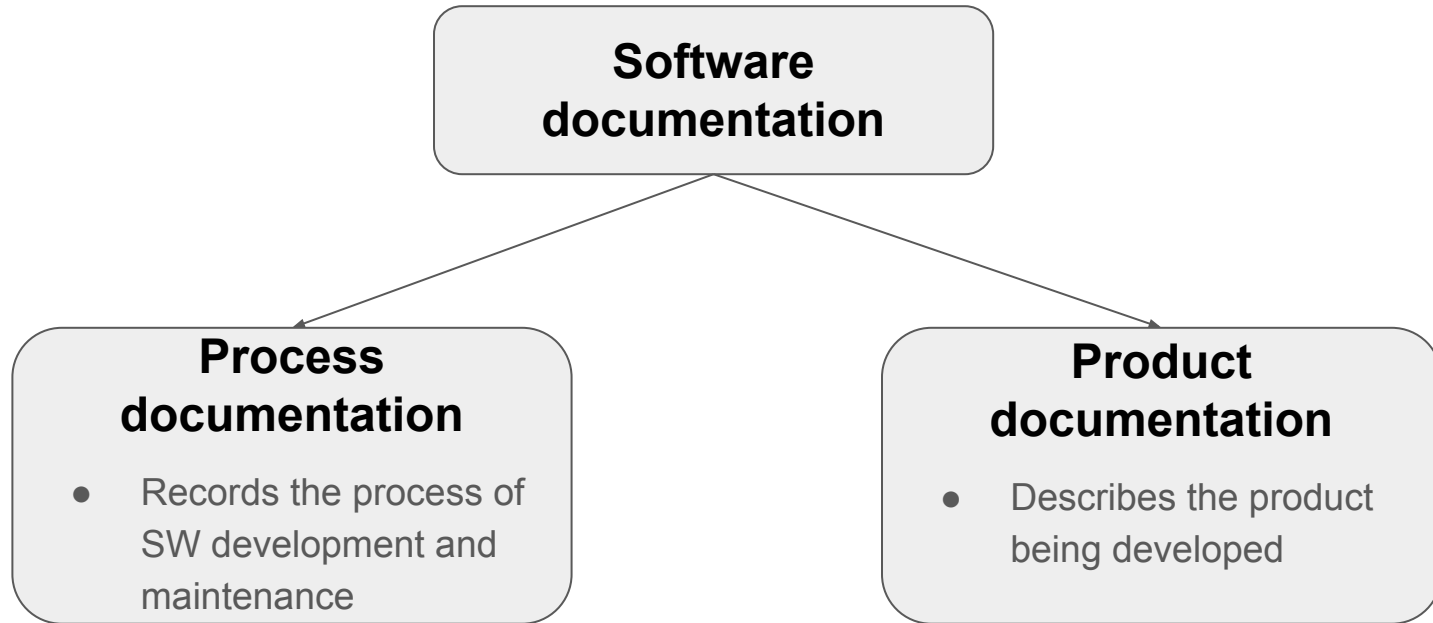
# Why documentation?

## 3. Provides information

- For users and system administrators
- For future maintenance
- For project management



# Classification



# Process documentation:

-> Records the process of SW development and maintenance

- PM documents
  - Predict and control the SW process
  - E.g., schedules, plans, estimates
- Reports
  - Report resource usage during the SW process
  - E.g., amount of man-days consumed, deadlines satisfaction
- Standards
  - Set out how the SW process is to be implemented
  - E.g., coding standards, documentation standards
- Communication documents
  - Record the details of everyday communication
  - E.g., memos, e-mail discussions, meeting minutes, version history

➤ These documents  
becomes quickly  
**out of date!**

# Product documentation

- These documents have **longer** lifespan

-> Describes the product being developed

## 1. **System documentation** -> Describes how the system works

- Requirements
- Architecture & design
- Source code listings (commented)
- Validation & verification documents (testing,..)
- Maintenance documents (List of Known Bugs, HW and SW dependencies, ..)

## 2. **User documentation** -> Tells user how to use the SW product

- Should take into account all relevant user classes
  - E.g., end users vs. system administrators
- Should take into account various levels of expertise
  - E.g., beginners -> screenshots, tutorials, use cases/scenarios
  - advanced users -> function reference, detailed description

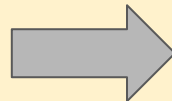
- Where does API documentation belong?
- Is it easy to change it after publishing?

# Document form

- Documents - Office, PDF, text, HTML,...
- Diagrams - UML, ..
- Wiki
- Document/content management systems
- E-mail messages
- Bug/issue reports
- Version history
- ...



Each (product) document provides a separate view of the system and these views overlap



It is important to keep all documents up-to-date and mutually consistent !



# Automatically generated documentation

## JavaDoc

- Generates documentation from Javadoc comments within the code
- Classes, fields, methods

## OpenAI / Swagger

- JSON / YAML specification format
- Can be generated from comments / annotations within the code
- Can be visualized
- [PetStore](#) example

Other: Doxygen, DocFX, ..

# Documentation management

- Amount of documentation grows quickly
- It is inevitable to manage the documentation efficiently
  - Use predefined document templates
  - Specify location for each type of document clearly
  - Use document (content) management systems
  - Use version control

## Recommended minimal system documentation:

- Requirements specification, architecture/design documents, commented source code



- It is better to provide a minimal but up-to-date and consistent documentation than a comprehensive but poorly maintained documentation

# Maintenance & Operations

# Software maintenance

= Modifying a system after it has been put into use

- ✓ Modifies existing components
- ✓ Adds new components to the system
- ✗ (Normally) does not significantly change the system's architecture



## Maintenance is not only bug fixing!

But also: adapting the software to changing requirements, changing environment, ...

- In fact, corrections represent only about 25% of all maintenance tasks

# Why software maintenance?

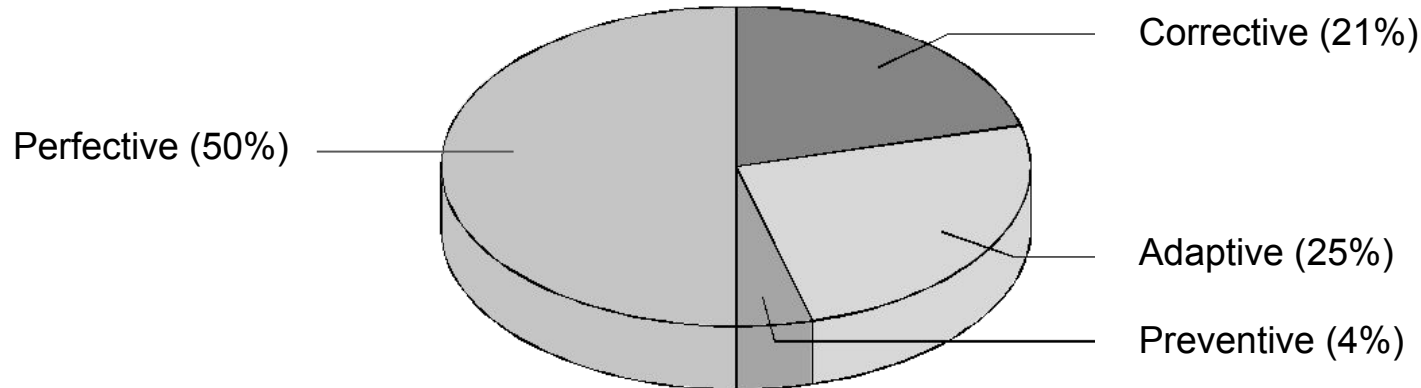
Environment of system operation changes in time

- > Requirements on system changes in time (and new requirements emerge)
- > Systems **MUST** be maintained if they are to remain useful in an environment

Maintenance may significantly overlap with development in agile approaches !

# Types of maintenance (ISO/IEC 14764)

1. **Adaptive:** Modifying the system to cope with environment changes (computer, OS, etc.)
2. **Perfective:** Modifying the system to satisfy new or modified requirements
3. **Corrective:** Correcting discovered problems
4. **Preventive:** Detecting and correcting latent faults before they become effective faults



# Why does software maintenance cost so much?

- It is usually more expensive to add functionality after a system has been developed rather than design this into the system
- Expensive activity: To figure out
  - WHAT part of code to modify and
  - HOW to modify it
- Overall maintenance costs:
  - Usually 2\* to 100\* greater than development costs (Ian Sommerville, 2000)
  - Affected by both technical and non-technical factors



# Factors affecting maintenance cost

- Team stability
  - Costs are reduced if the same staff are involved for some time
  - In case of staff turnover, "cultural" knowledge of the software is lost
- Contractual responsibility
  - The developers of a system may have no contractual responsibility for maintenance so there is no incentive to design for future change
- Staff skills
  - Maintenance staff are often inexperienced and have limited domain knowledge
  - Maintenance is generally considered as an unglamorous task and is typically assigned to the team newcomers
- Inadequate configuration management
  - Different representations of a system are out of step



# Factors affecting maintenance cost

- Inadequate documentation
  - Insufficient, incomplete, inconsistent or out-of-date documentation makes it more difficult to understand the system
  - Reverse engineering may help
- Inflexible design/architecture
  - The architecture and/or design of the system is too rigid to allow for simple implementation of requested changes
  - Costs are increased especially in case that significant changes in original software design are not allowed
- Program age and structure
  - Programs are poorly structured already during the initial development
  - As programs age, their structure is degraded and they become harder to understand and change (e.g., old languages, compilers, programming styles, design patterns)
  - Maintenance corrupts the software structure so makes further maintenance more difficult.

# Well-known maintenance examples

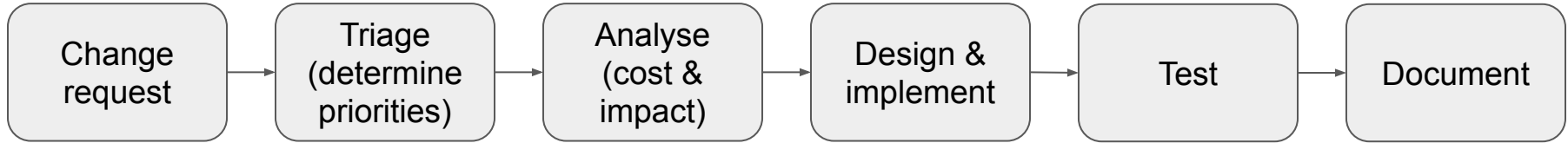
- Y2K (1.1.2000) - worldwide
- SKK -> EUR (1.1.2009) - Slovakia

→ Many systems had to be updated

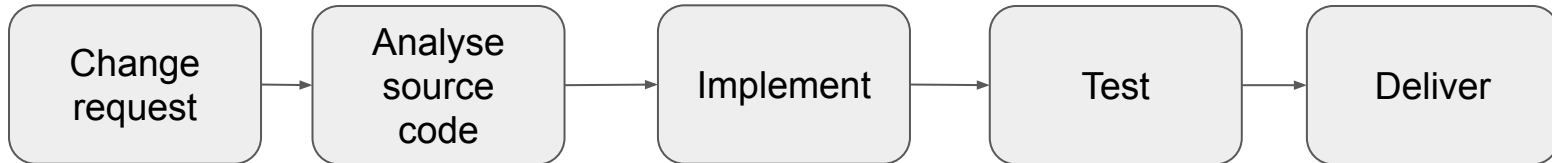
→ In both cases, complex analysis was needed  
(find where changes need to be made)

# Maintenance process

- Complex and varied (depends on type of maintenance)
- In general - structured maintenance:



- Unstructured maintenance (emergency repair):



Clearly structured maintenance is a more reliable and (usually) a more efficient process - unfortunately, it's not always possible

# Maintainable software

- Good initial design & architecture
- Understandable software structure
- Accurate documentation
- Good configuration management
- Use of standards (design, language, coding, etc.)

# Operations

= managing, deploying, maintaining, and supporting software systems in a production environment

Traditional setup:

- Separate development and operations
- Developers (Dev| hand off completed code to operations (Ops) for deployment
- Often caused delays and inefficiencies

## Infrastructure as a Code (IaaS)

Code is used both for SW implementation and for infrastructure management (Terraform, Ansible, Kubernetes, ..)

Modern CI/CD pipelines

- Code is tested and deployed continuously, via automation tools
- Development and deployment processes overlap:
  - Developers need to understand operations (like infrastructure management) to build deployable code
  - Operations teams need to understand the application logic to solve issues and optimize performance

## DevOps

= a set of principles that aim to integrate and automate the work of development and IT operations

# IaaS example (e-commerce store)

GitHub (same VCS platform)

|                   |                                         |
|-------------------|-----------------------------------------|
| — frontend/       | (React, Vue, Angular, etc.)             |
| — backend-api/    | (Node, Python, Java, etc.)              |
| — infrastructure/ | (Terraform, Pulumi, CDK, K8s manifests) |
| — shared-lib/     | (utilities shared across app/services)  |

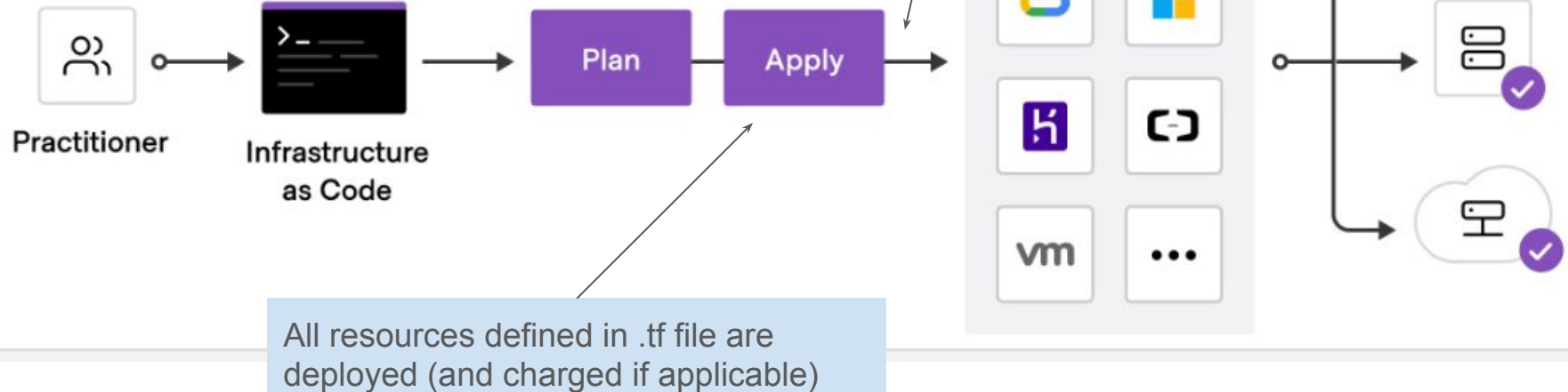
infrastructure -> [main.tf](#) (Terraform IaaS)

Description of infrastructure (resources)

- Provider (aws, google,..)
- VPC / subnets
- Security groups
- RDS (Relational Database Service) PostgreSQL
- ECS (container orchestration)
- Services (run in containers)
- ...

# Terraform workflow

Terraform communicates with API provided for resource management. It can be either cloud API or any other API available for resources to be managed.  
Credentials need to be provided for cloud API.



```
resource "aws_ecs_service" "frontend_service" {  
  desired_count = 2  # current  
}
```



```
resource "aws_ecs_service" "frontend_service" {  
  desired_count = 3  # updated  
}
```

Desired count - how much copies of the container ECS runs at all times

We want to change it from 2 to 3

1. terraform plan → Shows: “update ECS service: desired\_count from 2 to 3
2. terraform apply → Updates ECS service without touching other resources



# References

## Documentation

- Ian Sommerville: Software documentation, In Software Engineering, Vol 2: The Supporting Processes. R. H. Thayer and M. I. Christensen (eds), Wiley-IEEE Press.[Book chapter], 2002.
- ISO/IEC/IEEE 15289:2015: Systems and software engineering - Content of life-cycle information items (documentation), 2015
- Google Style Guides: <https://github.com/google/styleguide>

## Maintenance

- Ian Sommerville: Software Engineering (10th edition), 2015
- Pigoski, Thomas. Chapter 6: Software Maintenance (PDF). SWEBOK. IEEE. 2001, [http://sce.uhcl.edu/helm/SWEBOK\\_IEEE/data/swebok\\_chapter\\_06.pdf](http://sce.uhcl.edu/helm/SWEBOK_IEEE/data/swebok_chapter_06.pdf)
- Lientz B., Swanson E.: Software Maintenance Management. Addison Wesley, Reading, MA, 1980
- ISO/IEC 14764:2006: Software Engineering - Software Life Cycle Processes - Maintenance, 2006
- IEEE STD 1219-1998: Standard for Software Maintenance, 1998
- [What is DevOps](#) - online at aws.amazon.com
- Wikipedia: [Infrastructure as a Code](#)
- [What is Infrastructure as Code with Terraform?](#)