SOFT2412

WEEK 1

**Agenda**

– Software Engineering, Software Development, SD process, SDLC, SDLC/SD models

– Agile, agile software development, software development methods (waterfall, agile, spiral)

– Agile principles, agile practices

Software **Engineering**-what and why?

What:

* Software is using everywhere : power, communication, edu, health, etc.
* Improving working efficiency and creating economy
* Emerging challenge: security, tech innovation

Why:

* Need to build **high-quality software** systems under **resource constraints**
* Social: user need, convenience
* Economical: cost
* Time: deliver of product

**Engineer vs developer:**

Engineer: design architecture, provide methods, managing development process& complexity,

Developer: write codes.

**SD process**

– Software Development Process

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

– Defines how various activities are to be done, and in what order

– It defines a Lifecycle for a Software Development project

– Includes processes, a set of tools, definitions of the artefacts, etc.

* universally applicable SD process:

– **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

– planning

– Plan-driven (plan-and-document heavy-weight)

– Agile processes (light-weight)

**Software Process Models(SDLC)**

– **Waterfall Model** – Development process activities as process phases

– **Spiral Model** – Incremental development risk-driven

– **Agile Model** – Iterative incremental process for rapid software development

– **The Rational Unified Process** (RUP or UP) – Bring together elements of different process models – Phases of the model in timer (dynamic perspectives), process activities (static perspective), good practices (practice perspective)

**Waterfall**: strictly defined separate phases, passing from team to team.

Pros: easy to understand and implement, used for large systems engineering projects

Cons: difficulty of accommodating change as phases too strict

**Spiral**: each loop represent a phase, successive iteration.

Pros: better iteration, risk driven approach

Cons: not practical

**Rational unified process**: Iterative and incremental(like agile)

**Agile Development Model**

Agile intend to develop systems **more quickly** with **limited time spent on analysis and design**

Why agile important:

• constant introduction of new technology

• New players enter the market,

• New requirements are added

• “Small is Beautiful”

• listening to the customer reduce failure by a smaller, more flexible competitor

• reduce maintenance costs

Agile process:

– **Light-weight**

– **People-based** rather than Plan-based

WEEK 2 Tools and Technologies for Controlling Artefacts(GIT)

**Agenda**

– Software Development – **Artefacts**

– Software **Configuration Management**

– **Version Control Systems**

– Version Control with **Git**

**Artefacts**

– items produced that represent work done during a software development project

– artefacts **evolution** as being developed

– The artefacts have **value** and need to be preserved, communicated, maintained, protected

– High value -> need for **management**

**Requirements Artefact**: user requirement. in agile, as “User stories”. We need to track changes and versions of it.

**Code**: the code is spread over different files and directories, having a hierarchy structure.

**Dependencies**: some can depend on the state of others. Problems will rise if dependency not counted.

**Software Configuration Management**- Software version and **version management**

**Configuration Management (CM)** 配置管理

– Configuration management is concerned with the **policies, processes and tools** for **managing changing software systems**

– CM is essential for team projects to control changes made by **different developers**

Configuration Management Activities

– **Version management**: Keeping track of the multiple versions of system components and ensuring that changes made to components by different developers do not interfere with each other.

– **System building**: the process of assembling program components, data and libraries, then compiling these to create an executable system.

– **Change management**: keeping track of requests for changes to the software from customers and developers, working out the costs and impact of changes, and deciding the changes should be implemented.

– **Release management**: preparing software for external release and keeping track of the system versions that have been released for customer use.

**Version Management (VM)** 版本管理

– The process of **keeping track of different versions** of software components or configuration items, and making sure the **changes made by different developers don’t interfere**.

– **Codeline**: a sequence of versions of source code with **later versions** **derived from earlier versions**

– **Baseline**: a definition of a **specific system** that specifies the component, libraries used, configuration files, etc. baseline is important because you often have to **recreate a specific version of a complete system**.

**Semantic Versioning (SemVer)** 语义化版本

– Set of rules and requirements that determine how version numbers should be assigned and incremented for software being developed 给与版本号特殊含义

– **Managing dependencies**: the bigger your system grows and the more packages/libraries/plugins you integrate into your software

– Given a version number **MAJOR.MINOR.PATCH**, increment the:

1. **MAJOR** version when you make incompatible **API changes**, eg 1.0.0

2. **MINOR** version when you **add functionality i**n a backwards-compatible manner, eg 1.1.0

3. **PATCH** version when you make backwards-compatible **bug fixes**. Eg 1.0.1

**Version Control and Version Control system**

– A method for **recording changes to a file** or set of files over time so that you can **recall specific versions** later

**Version Control System (VCS)**

– Category of software tools that help software teams to manage changes to source code over time

– It allows developers to keep track of every modification to the code in a special kind of storage (repository)

WEEK 3 Tools and Technologies for Controlling Artefacts

**Agenda**

– Working with **Remote Repository** 、

– **GitHub**

– **Distributed Git**

– **Remote Branches**

– **Collaboration**

– **Workflows**

**Remote (Hosted) Repository**

– A **remote** repository is generally a **simple** repository ->the contents of your project’s **.git directory** and nothing else

– **Team-based (collaboration)** projects

– Remote repo team members (collaborators) can **access anytime**

– More reliable **common repo** (rather own local repo)

– All team members can **push and pull**

– Need to have some **coordination and permission control**

– Typical server set-ups using the configured protocols

|  |  |  |
| --- | --- | --- |
| Protocol | Pros | Cons |
| File system | simple, support access control | public share is difficult to setup |
| SSH | easy to setup (most systems provide ssh tools), fast (compress data), support authenticated write access | no anonymous access (even read access) |
| HTTP | unlikely to be blocked | Can become difficult to setup |
| Git | Fastest protocol, allow anonymous public access | Difficult to setup, lack of authentication, use non standard port (9418) which can be blocked |

**Hosted Git Service – GitHub**

**Organizations**

– Organization’s members can have owners or member roles:

– Owner: have complete administrative access to the organization (often a few people in the organization should be assigned as owner roles

– Member: default role for everyone else

– Owners can manage members’ access to the organization’s repos. and projects with fine-grained permission controls

– Can add collaborators from outside of the organization (consultant) to have access to one or more organization repos. without bring a member of the organization

**Distributed Git - Remote Branches**