3 Software technologies

• Dictionary
  – “techno-” art, skill, craft
  – “-logy” method, system
History

• In its history, software engineering changed not only the paradigms, but also technologies

• Fortran 2 + punched cards
  – technology of the past

• Swing/Java
  – technology of the present

• Services oriented architectures (SOA)
  – possible technology of the future

• Technologies used by software are accidental properties
Technology success

• Does the technology address important issue?
• Is research and development completed?
• Are there substantial residual problems? – “show stoppers”?
• Are market forces behind the technology?
• Only then, other measures may play a role
  • productivity, usability, other properties
Technology issues

• Some people call technologies “paradigms”
  – confusing overuse of the word paradigm
• Reengineering from one technology to another is often hard
• Technology and culture
  – technology can be dead for cultural reasons.
Software technology portfolio

- Programming languages and compilers
- Databases
- GUI
- Software environments
  - programmer activities
  - tools
- Version Control System
- Testing technologies
- . . .
What language we would use if C did not exist?

• obol
• Pasal
• Basi
• ++
• . . .
Compilers and linkers

Source file 1 → Object file 1

Source file 2 → Object file 2

…

Source file N → Object file N

Compiler

Object file 1

Object file 2

…

Object file N

Linker

Libraries

Executable file
Version Control System

Server
Configuration management repository

checkout
commit

checkout
commit

checkout
commit

checkout
commit
Diff and Merge

• Diff produces a patch file that identifies what changes the programmer made in the file
  – Compares the modified file X against the original file

• Merge takes original file and a patch file
  – merges the changes into the old file and thus creates the new file.
Check out and commit

- A programmer checks out the latest version of file X
- Makes all necessary changes
- Then commits the modified file X to the repository
Commit

• The commit is usually done with a help of tool diff and merge
  – diff produces a patch that identifies what changes the programmer made in the file by comparing the modified file X against the original file in the repository
  – tool merge then merges the changes into the old file and thus creates the new file
Conflict

• Programmer A and programmer B update the same file X in parallel
• Programmer B changes something in the file X that the first programmer A changed also
• Version control system does not allow the second programmer B to commit
Resolution of conflict

• Programmer B
  – undoes some changes in order to stay away from the areas that programmer A changed
  – checks out the version that was modified by programmer A and makes the changes in this modified file again
  – convinces programmer A to undo the changes which stand in the way of the changes done by programmer B
Object oriented technology

• Classes

```java
class Customer {
    // methods
    public open_account();
    public deposit(int deposit_amount);
    public withdraw(int withdrawal_amount);
    // attributes
    protected String name;
    protected String address;
    protected int account_number;
    protected int balance = 0;
}
```

• Objects

```java
Customer jacob, joseph;
```
OO code

Customer jacob, joseph;

jacob.open_account();
jacob.deposit(100);
jacob.withdraw(10);
Coding conventions

- **CamelCase**
  
  `firstCustomerWithdrawal`

- **Under_line**

  `first_customer_withdrawal`
Part-of relationship

- Composite vs. component
  - inventory and register are parts of a store
  - components of a store

```java
class Store {
    protected Inventory inv;
    protected Register reg;
}
```
Inheritance/ is-a relationship

class Person // Base type
{
    protected String name;
    protected String address;
}

class Customer: public Person
    // Derived type; Customer is-a Person
{
    protected String account;
    public open_account(int account_number);
    public deposit(int deposit_amount);
    public withdraw(int withdrawal_amount);
}
Polymorphism

• Derived type used instead of base type
  – implicit switch statement
  – late binding
    • dynamic binding
  – C++ pointers + virtual functions
Example

class FarmAnimal {
    public:
    virtual void makeSound() {};
};

class Cow : public FarmAnimal {
    public:
    void makeSound() {cout<<"Moo-oo-oo";}
};
class Sheep : public FarmAnimal {
    public:
    void makeSound() {cout<<"Be-e-e";}
};
Example (cont)

FarmAnimal* ourAnimal[3];
    ourAnimal[0] = new Cow;
    ourAnimal[1] = new Sheep;
    ourAnimal[2] = new Cow;

for(int i = 0; i < 3; i++)
{
    ourAnimal[i] -> makeSound();
}

Print out: “Moo-oo-oo Be-e-e Moo-oo-oo”
Technology diffusion

- Innovators
  - the first group of users
  - assume the risks
- Early adopters
  - draw on innovators’ experience
Technology diffusion cont.

• Majority adopters
  – advantages are proven and well understood
  – may take more than 10 years to reach this stage

• Late adopters
  – there is no longer any significant doubt

• Laggards
  – they have no other option
Technology diffusion example

Automobile in Detroit

• Innovators
  – Henry Ford, others

• Early adopters
  – wealthy, purchase from innovators

• Majority adopters
  – model T made it accessible

• Late adopters
  – still used streetcars, but then saw benefits

• Laggards
  – streetcar tracks torn up, no other option
Hype cycle

- Peculiar behavior of innovators
- Technology trigger
  - product launch or other event generates significant interest
- Inflated expectations
  - a frenzy of publicity generates unrealistic expectations
- Disillusionment
  - technologies fail to meet expectations and become unfashionable
  - normal diffusion resumes after disillusionment bottoms out
Hype cycle

J. Fenn, A. Linden. “Gartner’s Hype Cycle Special Report for 2005”. Gartner Research, Aug., 2005