Rigorous Software Engineering

Requirements

Prof. Zhendong Su

(based on slides from Prof. Peter Müller)

Spring 2019
Exercise Groups

- Please enter preferences & group information at https://goo.gl/forms/N8NpaeifFsumnSZp2 by 11:59 PM, Feb. 20th (Wed)

- Exercise groups will be announced on Feb. 22nd

- Exercise sessions start next week
What is the mission of CS?

Develop methodologies & techniques for building reliable & performant software
Will Programming Jobs Be Replaced By AI?
NATO Software Engineering Conference 1968
Fred Brooks: (Still) “No Silver Bullet”

- Software is Essentially hard to build
- Complexity is inherent
- Conformity to hardware and world
- Changeability (looks easy to change)
- Invisibility

“No Silver Bullet” ’85, Refired

“There is no single development in technology or management which alone promises a 10X gain in 10 years” is again true 30 years later
#include <stdio.h>
#include <string.h>

int main()
{
    char *word = "everest";
    char reverseword[strlen(word)+1];
    unsigned int letters_remaining = strlen(word);
    char *wordpointer = &word[strlen(word)-1];
    int i = 0;
    while(letters_remaining > 0){
        reverseword[i++] = *wordpointer--;
        letters_remaining--;
    }
    reverseword[strlen(word)] = '\0';
    printf("So the reversed word is %s\n", reverseword);
    return 0;
}
#ifndef __ASM_ALPHA_FPU_H
#define __ASM_ALPHA_FPU_H

#include <asm/special_insns.h>
#include <uapi/asm/fpu.h>

/* The following two functions don’t need trapb/excb instructions around the mf_fpcr/mt_fpcr instructions because (a) the kernel never generates arithmetic faults and (b) call_pal instructions are implied trap barriers. */

static inline unsigned long
rdfpocr(void)
{
    unsigned long tmp, ret;

    #if defined(CONFIG_ALPHA_EV6) || defined(CONFIG_ALPHA_EV67)
        __asm__ __volatile__(
            "flio $f0,%m\n\t" \
            "mf_fpcr $f0\n\t" \
            "flio $f0,%1\n\t" \
            "itof %0,$f0" \
            : "=r"(tmp), "=r"(ret));
    #else
        __asm__ __volatile__(
            "sti $f0,%m\n\t" \
            "mf_fpcr $f0\n\t" \
            "sti $f0,%1\n\t" \
            "ldt $f0,%0" \
            : "=m"(tmp), "=m"(ret));
    #endif

    return ret;
}

static inline void
wrfpcr(unsigned long val)
{
    unsigned long tmp;

    #if defined(CONFIG_ALPHA_EV6) || defined(CONFIG_ALPHA_EV67)
        __asm__ __volatile__(
            "flio $f0,%0\n\t" \
            "itof "$f0," \
            "itof "$f0," \
            "itof "$f0,"

.../linux/arch/alpha/include/asm/fpu.h [Ins] Eval: START
public Owner getOwner() {
    return this.owner;
}

protected void setVisitsInternal(Set<Visit> visits) {
    this.visits = visits;
}

protected Set<Visit> getVisitsInternal() {
    if (this.visits == null) {
        this.visits = new HashSet<Visit>();
    }
    return this.visits;
}

public List<Visit> getVisits() {
    return new HashSet<Visit>(this.visits);
}
```
#include <UIKit/UIKit.h>
#import <Foundation/Foundation.h>
#import <UIKit/UIKitDefines.h>

NS_ASSUME_NONNULL_BEGIN

typedef NS_ENUM(NSUInteger, UIDeviceOrientation) {
    UIDeviceOrientationUnknown,
    UIDeviceOrientationPortrait,  // Device orientation is portrait.
    UIDeviceOrientationPortraitUpsideDown,  // Device orientation is portrait upside down.
    UIDeviceOrientationLandscapeLeft,  // Device orientation is landscape left.
    UIDeviceOrientationLandscapeRight,  // Device orientation is landscape right.
    UIDeviceOrientationFaceUp,  // Device orientation is face up.
    UIDeviceOrientationFaceDown  // Device orientation is face down.
} __TVOS_PROHIBITED;

typedef NS_ENUM(NSUInteger, UIDeviceBatteryState) {
```

```cpp
#include <iostream>
#include <cassert>
using namespace std;
#include "foo.h"
#if ABC
void foo() {}
#endif
#define MUL(a,b) a*b

int main()
{
    cout << "2+3=" << add(2,3)
    cout << "Hello, C++" << endl;
    getchar();
    return 0;
}
```
Can we move beyond “coding”?
your wish?
Communicating the wish the hardest
function drawTree() {
    var blossomPoints = [];
    resetRandom();
    drawBranches(0, Math.PI/2, canvasWidth/2, canvasHeight, 30, blossomPoints);
    resetRandom();
    drawBlossoms(blossomPoints);
}

function drawBranches(i, angle, x, y, width, blossomPoints) {
    ctx.save();
    var length = tween(i, 1, 62, 12, 3) * random(0.7, 1.3);
    if (i === 0) { length = 107; }
    ctx.translate(x, y);
    ctx.rotate(angle);
    ctx.fillStyle = "#000";
    ctx.fillRect(0, -width/2, length, width);
    ctx.restore();

    var tipX = x + (length - width/2) * Math.cos(angle);
    var tipY = y + (length - width/2) * Math.sin(angle);

    if (i > 4) {
        blossomPoints.push([x, y, tipX, tipY]);
    }
    if (i < 6) {
        drawBranches(i + 1, angle + random(-0.15, -0.05) * Math.PI, blossomPoints);
        drawBranches(i + 1, angle + random(0.15, 0.05) * Math.PI, blossomPoints);
    }
    else if (i < 12) {
        drawBranches(i + 1, angle + random(0.25, -0.05) * Math.PI, blossomPoints);
    }
}
```javascript
var length = tween(1, 1, 62, 12, 3) * random(0.7, 1.3);
if (i == 0) { length = 107; }

ctx.translate(x, y);
ctx.rotate(angle);
ctx.fillRect(159, -width/2, length, width);
ctx.restore();

var tipX = x + (length - width/2) * Math.cos(angle);
var tipY = y + (length - width/2) * Math.sin(angle);

if (i > 4) { blossomPoints.push(x, y, tipX, tipY); }

if (i < 6) {
  drawBranches(i + 1, angle + random(-0.15, -0.05) * Math.PI);
  drawBranches(i + 1, angle + random(0.15, 0.05) * Math.PI);
} else if (i < 12) {
  drawBranches(i + 1, angle + random(0.25, -0.05) * Math.PI);
}

function drawBlossoms(blossomPoints) {
  var colors = [
    "#f5c9d4", "#f7c9f3", "#ebb4cc", "#f9d0f9", "#e2d6f9", "#f8e5ef", "#e5e4ef" ];
  ctx.globalAlpha = 0.60;

  for (var i = 0; i < blossomPoints.length; i++) {
    var p = blossomPoints[i];
    for (var j = 0; j < 16; j++) {
      var x = lerp(p[0], p[2], random(0,1)) + random(-10,10);
      var y = lerp(p[1], p[3], random(0,1)) + random(-10,10);
      ctx.fillStyle = colors[Math.floor(random(0, colors.length))];
      ctx.fillCircle(x, y, random(2,5));
    }
  }
```
Bret Victor: Inventing on Principle
function drawTree() {
    var blossomPoints = [];
    resetRandom();
drawBranches(0, -Math.PI/2, canvasWidth/2, canvasHeight, 30);
    resetRandom();
drawBlossoms(blossomPoints);
}

function drawBranches(i, angle, x, y, width, blossomPoints) {
    ctx.save();
    var length = tween(i, 1, 62, 12, 3) + random(0.7, 1.3);
    if (i == 0) { 'length = 107'; }
    ctx.translate(x, y);
    ctx.rotate(angle);
    ctx.fillStyle = "#0000";
    ctx.fillRect(0, -width/2, length, width);
    ctx.restore();
    var tipX = x + (length - width/2) * Math.cos(angle);
    var tipY = y + (length - width/2) * Math.sin(angle);
    if (i > 4) {
        blossomPoints.push([x, y, tipX, tipY]);
    }
    if (i < 6) {
        drawBranches(i + 1, angle + random(-0.15, 0.05) + Math.PI, 30);
        drawBranches(i + 1, angle + random(0.15, 0.05) + Math.PI, 30);
    } else if (i < 12) {
        drawBranches(i + 1, angle + random(0.25, -0.05) + Math.PI, 30);
    }
}
Goal is the **object**, not the code
Can we directly manipulate & explore the object to express the “wish”?
Can we directly manipulate & explore the object to express the “wish”?

Perhaps via visualization & virtual reality?
Main Activities of Software Development

- Requirements Elicitation
- Design
- Implementation
- Validation

These activities may overlap and are typically executed iteratively.
How the customer explained it

How the project leader understood it

How the analyst designed it

How the programmer wrote it

How the consultant described it

How the project was documented

What was installed by operations

How the customer was billed

How the project was supported

What the customer really needed
Software – a Poor Track Record

- 68% of all software projects are unsuccessful
  - Cancelled
  - Late, over budget, fewer features than specified
- The average unsuccessful project
  - 179% longer than planned
  - 154% over budget
  - 67% of originally specified features
Why IT-Projects Fail

- Top 5 reasons measured by frequency of responses by IT executive management

- Failure profiles of yellow projects (44%)
  1. Lack of User Input 12.80%
  2. Incomplete Requirements 12.30%
  3. Changing Requirements 11.80%
  4. Lack of Executive Support 7.50%
  5. Technology Incompetence 7%

- Failure profiles of red projects (24%)
  1. Incomplete Requirements 13.10%
  2. Lack of User Involvement 12.40%
  3. Lack of Resources 10.60%
  4. Unrealistic Expectations 9.90%
  5. Lack of Executive Support 9%
2. Requirements Elicitation

2.1 Requirements

2.2 Activities
Requirements

- Definition
  
  *A feature that the system must have or a constraint it must satisfy to be accepted by the client*

  [Brügge, Dutoit]

- Requirements engineering (RE) defines the requirements of the system under construction
Requirements

- Describe the user’s view of the system
- Identify the what of the system, not the how

Part of requirements
- Functionality
- User interaction
- Error handling
- Environmental conditions (interfaces)

Not part of requirements
- System structure
- Implementation technology
- System design
- Development methodology
Types of Requirements

- **Functionality**
  - What is the software supposed to do?

- **External interfaces**
  - Interaction with people, hardware, other software

- **Performance**
  - Speed, availability, response time, recovery time

- **Attributes (quality requirements)**
  - Portability, correctness, maintainability, security

- **Design constraints**
  - Required standards, operating environment, etc.
Functionality

- Relationship of outputs to inputs
- Response to abnormal situations
- Exact sequence of operations
- Validity checks on the inputs
- Effect of parameters
External Interfaces

- Detailed description of all inputs and outputs
  - Description of purpose
  - Source of input
  - Destination of output
  - Valid range, accuracy, tolerance
  - Units of measure
  - Relationships to other inputs/outputs
  - Screen & window formats
  - Data and command formats
Performance

- **Static numerical requirements**
  - Number of terminals supported
  - Number of simultaneous users supported
  - Amount of information handled

- **Dynamic numerical requirements**
  - Number of transactions processed within certain time periods (average and peak workload)
  - Example: 95% of the transactions shall be processed in less than 1 second
Constraints (Pseudo Requirements)

- Standard compliance
  - Report format, audit tracing, etc.

- Implementation requirements
  - Tools, programming languages, etc.
  - Development technology and methodology should not be constrained by the client. Fight for it!

- Operations requirements
  - Administration and management of the system

- Legal requirements
  - Licensing, regulation, certification
Quality Criteria for Requirements

Correctness
Requirements represent the client’s view

Completeness
All possible scenarios are described, including exceptional behavior

Consistency
Requirements do not contradict each other

Clarity
Requirements can be interpreted in only one way
Quality Criteria for Requirements (cont’d)

**Realism**
Requirements can be implemented and delivered

**Traceability**
Each feature can be traced to a set of functional requirements

**Verifiability**
Repeatable tests can be designed to show that the system fulfills the requirements
Quality Criteria: Examples

- “System shall be usable by elderly people”
  - Not verifiable, unclear
  - Solution: “Text shall appear in letters at least 1cm high”

- “The product shall be error-free”
  - Not verifiable (in practice), not realistic
  - Solution: Specify test criteria

- “The system shall provide real-time response”
  - Unclear
  - Solution: “The system shall respond in less than 20ms”
Relative Cost to Fix an Error

- The sooner a defect is found, the cheaper it is to fix

[Boehm 1981]
Requirements Validation

- A quality assurance step, usually after requirements elicitation or analysis

- **Reviews** by clients and developers
  - Check all quality criteria
  - Future validations (testing)

- **Prototyping**
  - Throw-away or evolutionary prototypes
  - Study feasibility
  - Give clients an impression of the future system
  - Typical example: user interfaces
2. Requirements Elicitation

2.1 Requirements

2.2 Activities
Requirements Elicitation Activities

- Identifying Actors
- Identifying Scenarios
- Identifying Use Cases
- Identifying Nonfunctional Requirements
Identifying Actors

- Actors represent **roles**
  - Kind of user
  - External system
  - Physical environment

- **Questions** to ask
  - Which user groups are supported by the system?
  - Which user groups execute the system’s main functions?
  - Which user groups perform secondary functions (maintenance, administration)?
  - With what external hardware and software will the system interact?
Scenarios and Use Cases

- Document the behavior of the system from the users’ point of view
- Can be understood by customer and users

**Scenario**
- Describes common cases
- Focus on understandability

**Use Case**
- Generalizes scenarios to describe all possible cases
- Focus on completeness

- A scenario is an instance of a use case
Scenarios

- **Definition:**
  
  A narrative description of what people do and experience as they try to make use of computer systems and applications

  [M. Carroll, 1995]

- **Different Applications during the software lifecycle**
  - Requirements Elicitation
  - Client Acceptance Test
  - System Deployment
Scenario Example

When Alice wants to borrow a book, she takes it to the checkout station. There she first scans her personal library card. Then she scans the barcode label of the book. If she has no borrowed books that are overdue and the book is not reserved for another person, the systems registers the book as being borrowed by her and turns off the electronic safety device of that book. Several books can be checked out together. The checkout procedure is terminated by pressing a ‘Finished’ key. The system produces a loan slip for the books that have been borrowed.

[Adapted from Glinz 2000]
Identifying Scenarios: Questions to Ask

- What are the tasks the actor wants the system to perform?
- What information does the actor access?
- Which external changes does the actor need to inform the system about?
- Which events does the system need to inform the actor about?
Sources of Information

- **Client**
  - User manuals
  - Procedure manuals
  - Company standards
  - etc.

- **Users**
  - Speak to the end user, not just to the client

- **Existing documentation**

- **Task observation**
Use Cases

- A list of steps describing the interaction between an actor and the system, to achieve a goal

- A use case consists of
  - Unique name
  - Initiating and participating actors
  - Flow of events
  - Entry conditions
  - Exit conditions
  - Exceptions
  - Special requirements
Use Case Example: Event Flow

**Actor steps**
1. Scans library card
3. selects ‘Borrow’ function
5. scans label of book to be borrowed
7. presses ‘Finish’ key

**System Steps**
2. validates the card; returns the card; displays user data; displays ‘Select function’ dialog
4. displays ‘Borrow’ dialog
6. identifies book; records book as borrowed, unlocks safety label; displays book data
8. prints loan slip; displays ‘Finished’ message

Also specify alternative flows and exceptional cases
Identifying Nonfunctional Requirements

- **Nonfunctional** requirements are defined together with functional requirements because of dependencies
  - Example: Support for novice users requires help functionality

- Elicitation is typically done with check lists

- Resulting set of nonfunctional requirements typically contains conflicts
  - Real-time requirement suggests C or assembler implementation
  - Maintainability suggests OO-implementation