Singapore Institute of Management
BSc in Computing & Information Systems
CS 210 Tutorial 14

Why is software so hard to maintain?

Answer :

Software is so difficult to maintain because :

- ***Complexity*** *- specification domain and solution domain are complex and the process by which it is constructed is complex. Humans find it hard to manage complexity.*

- ***Flexibility*** *- Customers and developers expect software to be flexible in their environment to change, adapt and evolve to meet new requirements. If systems fail to meet these goals, the software developers have to work hard to regain 'credibility' and that of their system.*

- ***Interactivity*** *- Software is interactive. Typically even mechanical interactive systems require more maintenance effort than their non-interactive counterparts. Interaction with user changes the user's perception of the system and their context in which it operates. User interfaces require maintenance.*

- ***Immaturity*** *- Software engineering is a new and demanding discipline. Much theory and practice needs to be established and techniques remain undiscovered. Few ground rules exist and there is little international recognition for the importance and depth of the subject.*

Define backward slicing. Give an example of how this will help in understanding a program.

Answer :

A (static, backward) program slice s, of a program p, is constructed with respect to a slicing criterion (V,i) where V is a set of variable identifiers and I is a program point. Statements in p which cannot affect the values of V when the next statement to be executed is at point I, may be removed from p to form s.

Given the below eg.

```
read(n);
s:=0;
p:=1;
while n>0 do
   begin
            s:=s+n;
            p:=p*n;
            n:=n-1;
   end;
```

Backward slicing is useful when trying to evaluate the computations which may affect the value of the variable s at line number 9. Backward slicing removes statements which cannot affect the value of this variable at that point.

This is useful because large programs often contain many intertwined computations and so the human may be swamped by the amount of information that has to be considered. If this aspect can be denoted by a slicing criterion, then slicing will allow the human to avoid consideration of parts of the program which are irrelevant..

As a maintainer of programs, what helps you to understand the programs? What is program transformation? When do you use it to help in maintenance of programs? What are some major problems encountered?

Answer :

Comments in codes, and documentation at system level are useful for understanding programs during maintenance. Automated support is also required to tell maintainers what an existing system does, what its interdependencies are and what the effect of changing a component of the system will be - achievable through the use of software tools such as CASE tools.

Program transformation is a technique for manipulating existing system code to re-engineer it. This activity involves altering the structure of the code without altering its functionality.

Transformation can also be used as a tool for program comprehension as it allows a maintainer to view a system through different syntactic views, all of which preserve the semantics of the system.

The reason for performing transformation is to simplify the existing codes so that the new codes are easy to prove correct and more easily understood and easier to maintain then the old codes.

Right transformation strategies are lacking in the industry. A transformation strategy is a collection of transformation tactics applied in an algorithmic fashion to achieve some form of code-level improvement. For e.g. the strategy might be to remove side effects from the code, to remove go-to statements or it might be more complex; to remove some side effects but only if this could be achieved with some (specified) tolerable increase in overall program length.

What factors should you take into consideration when designing software to reduce maintenance costs? Explain.

Answer :

Factors to consider include :

- ***Design for change*** *- system designer should 'expect the unexpected'. That is developer should spend some time on the question of "what could change that I have assumed will be fixed". For e.g. one may have assumed phone numbers to be of fixed length of 7 digits and designed system to be thus. But later changes were introduced to increase the length to 8 digits! Thus in the original design of the system, the developer should considered and designed the field length to be 15 digits to cater for future expansion and maybe to take into consideration overseas numbers as well!*

- ***Design for Testing*** *- Changing the system requires testing. Thus should consider designing system for easy testing. For e.g. as coding time is inexpensive, include debugging facility with debugging facility. This can be switched on and off providing useful information to the tester of the system. Can even include assertions into the code to clarify connection between design and implementation.*

- ***Design for maintenance*** *- Maintainer has to understand the existing system before attempting changes. Due to possibility of ripple effects of changes, maintainer need to understand the system at more levels of abstraction than a developer does. So should design system that makes maintenance comprehension as straightforward as possible.*

Over engineer *- Design of system assumes more demanding requirements than those which are strictly necessary and therefore committing more resources to the design and implementation. This may be experienced by a developer who allocates more memory than strictly required.*

User perception *- General public have unrealistic view of the software development process. The process is expected to be expedient, fault-free and precise while its products are expected to be flexible, powerful and easy to use. This view if remains unchanged then the software engineering profession risks a huge backlash from a dissatisfied public.*

Under what circumstances would you use dynamic slicing to debug programs? Explain with example.

Answer :

Dynamic slice is constructed with respect to a specific input and is only valid for that input. A dynamic slicing criterion consists of a set of variables, a line number and a sequence of input.

Dynamic slicing is useful for debugging programs when for a particular input, an incorrect result was produced by the program in question and the developer is to discover the statement of statements which cause the error to occur. Useful to only consider lines of the program which do not contribute to the result in error based on the input.

For e.g. for the code fragment :

```
scanf ("%d", &x);
scanf("%d",&y);
if (x>y)
   z= 1 ;
else
  z=2 ;
printf("%d",z);
```

if the input was <1,1>, then line 6 would have been executed but line 4 will not be. A dynamic slice constructed with respect to the criterion ({z},<1,4>) will not contain line 4. The idea could have been to print z = 1 instead of 2, as the condition was to be x>=y and not x>y. This would help to debug the program.

You have been tasked to change a piece of code by amending the computation formula for a particular variable. Suggest the type of method you would use to check impact of the code change. Justify your choice of method.

Answer :

Will use forward Slicing to check impact.

Forward slicing is to look forward in a program to see which statements can be affected by the slicing criterion of (V,i) where V is the set of variable identifiers and I is a program point. Thus we are finding statements from point I forward which would be affected were we to introduce an erroneous computation at the slice point i.

For e.g. the following program :

```
year = 98;
Time  = 10;
Years = invoice-year;
If (Years > Chase_Value)
{
    printf("Due: ");
   printf("%d, Years);
   printf(" years ago at ");
   printf("%dpm",time);
}
```

So for a static slicing criterion ({year},1), The forward slice contains the statements which are affected by change to the value of year at line 1.

The forward slice looks like :

```
year = 98;

Years = invoice_year;
If (Years> Chase_value);

    Printf("%d",Years);
```

This helps in understanding what is the effect to the rest of the program should the value of a variable be changed at a particular point In the program due to the new requirements.