



Estimating the Number of Remaining Bugs

When should we stop testing

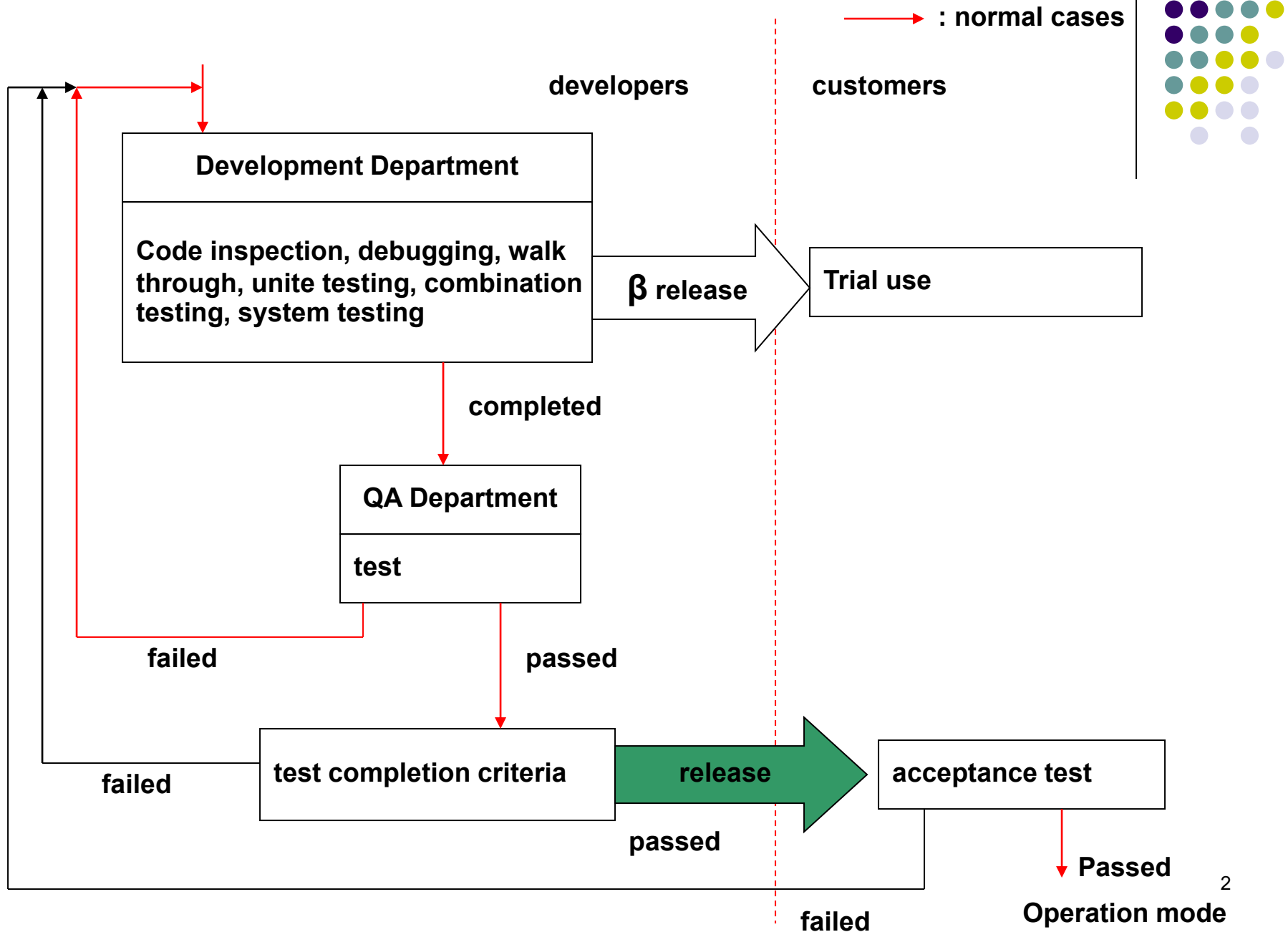
Charoen Vongchumyen

Email : charoen.vo@kmitl.ac.th

05/2021

Thank to Tsuneo Yamaura

Steps towards release

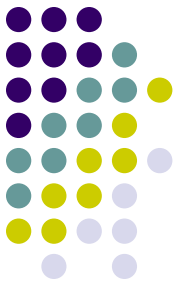


Test completion criteria / release criteria (part 1)




- Ideally, 'We release the software because it reached the quality level that we planned'
- In reality, 'We release the (premature) software because the release date has come'
 - ⇒ Quality is the pride of the engineers! This is not the right attitude of the engineers!
- Example of compromise
 - β release (trial use)
 - ⇒ 'Well, this program has bugs because this is not an official release.'
 - Partial release (a part of functions are released)
 - ⇒ 80 % of the functions are provided by 20 % of the modules
 - ⇒ Most likely, the customers do not tell 'We need the remaining 20 % immediately' ?

Test completion criteria / release criteria (part 2)



- **Ideally test completion criteria :**
 - (1) All the test cases were executed.**
 - (2) All the detected bugs have been fixed.**
 - (3) All the detected bugs have been analyzed to check similar bugs.**
 - (4) The bug growth model (S – shaped curve) came to saturation.**
 - (5) Passed the stress testing and the continuous operation testing.**
- **The worst test completion criteria :**
 - (1) The release date has come.**

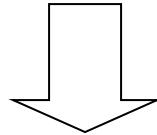
Compromise in test completion criteria :

- 
- (1) All the test cases were executed.**
 - (2) All the serious bugs were fixed.**
 - (3) MTBF (Mean Time Between Failures) was long enough.**

Test completion criteria / release criteria (part 3)

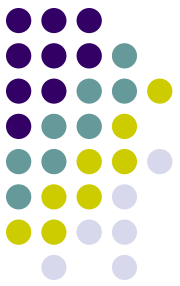


- **When you have to violate car parking regulations.**
 - (1) You have to recognize that you are violating parking regulations.
 - (2) You have to measure parking hours, and understand the environment.

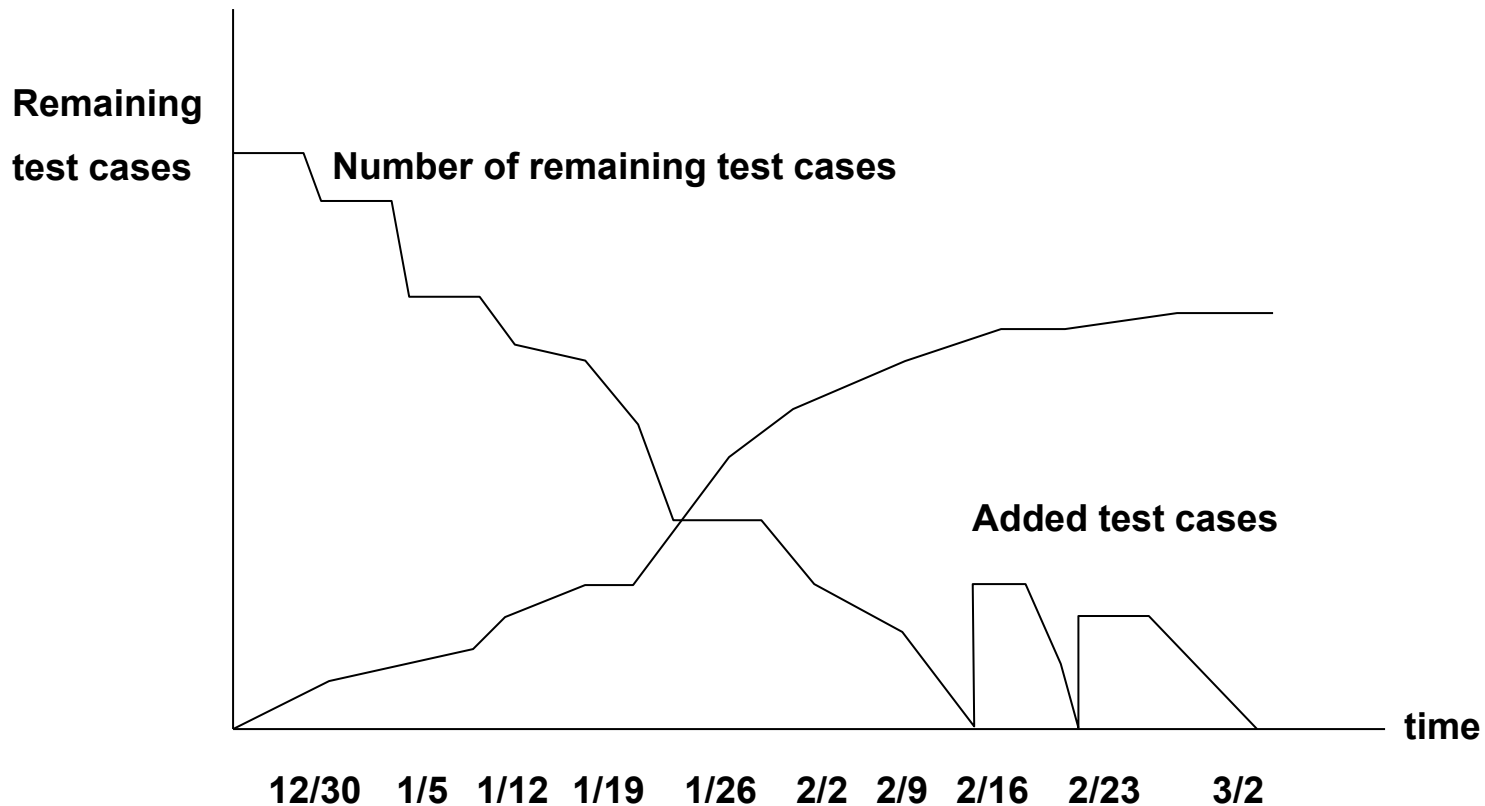


- **When you have to release software whose quality has not yet come to the shipment criteria.**
 - (1) You have to quantitatively measure the quality
 - (2) You have to be able to quantitatively estimate that how many person – months will bring what quality level.
 - ⇒ Estimate from the S – shaped model of “test case execution” and “bug growth”

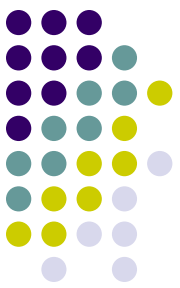
The Quality Data You Should Collect (part 1)



- **Test case execution curve (Gompertz curve)**
⇒ The curve that tells how many test cases executed when.
- **Bug growth curve (Gompertz curve)**
⇒ The curve that tells how many bugs detected when.



The Quality Data You Should Collect (part 2)



- **Bug Data**

(1) Symptoms / behavior of bug, Who and when detected

(2) Magnitude of bug

⇒ light (like misspelling) ⇔ catastrophic (environment destroying)

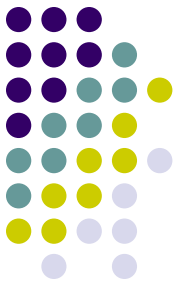
(3) Reproducing conditions

⇒ 90 % is solved if you can pinpoint the conditions that trigger the bug.

(4) When you fix the bug, you have to specify.

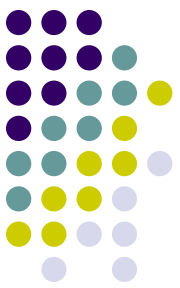
- The module where the bug resided
- Information of “before fixing” (e.g., source code)
- Who fixed when
- Who approved it when

Estimating the Number of Remaining Bugs



- (1) Capture / recapture model (planting artificial bugs)**
- (2) Sampling**
- (3) Gompertz Curve**
- (4) MTBF (Mean Time Between Failures)**
- (5) Previous statistical data**

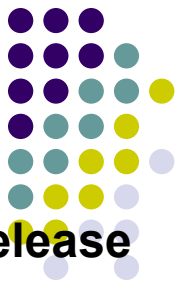
(1) Capture / recapture model



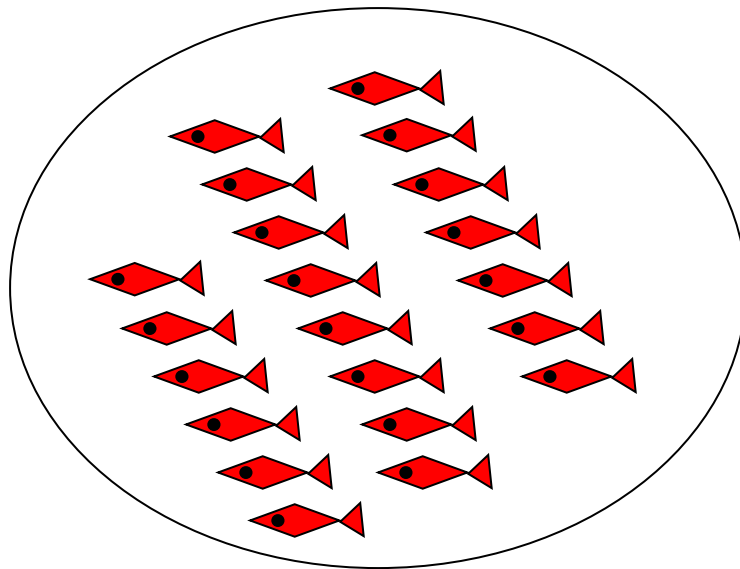
- Also called ‘fish – in – the pond’ model.
- A zoological method to estimate the population of animals living in a certain area.
- If you want estimate the number of fish in the pond, follow the steps below :
 - (1) Capture F_{n0} pieces of fish, paint them red, and release them.
 - (2) If a week later you recapture F_{n1} , and Fr pieces of fish are red, you can estimate the population of fish in the pond (F_p) by the below formula :

$$F_p = F_{n0} \cdot \frac{F_{n1}}{Fr}$$

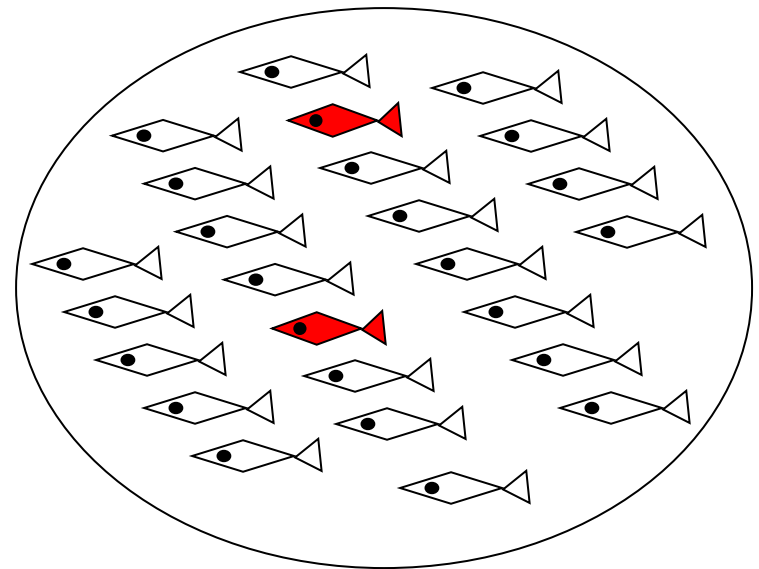
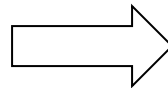
(1) Capture / recapture model



(ex.) You caught, 100 pieces of fish, painted them in red, and release them. A week later, you caught 200 pieces of fish, and there were 5 red. The red 100 are supposed to be $5 / 200$, and the population of the fish is : $100 * (200 / 5) = 4,000$



100 caught and painted in red



5 out of 200 were red

(1) Capture / recapture model



- **Bug planting model**

The capture / recapture model is applied to estimated the number of the remaining bugs.

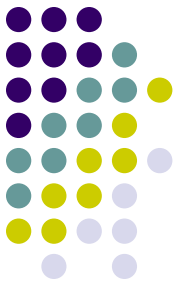
- The step to estimate the number of the remaining bugs are :

(1) A project manager plants $Ea0$ of artificial bugs to the source code ($Ea0$ of artificial bugs are red fish).

(2) If in the debugging phase a programmer detects $E1$ bugs, and there are $Ea1$ artificial bugs, the population of the bugs are estimated as :

$$Ep = Ea0 \cdot \frac{E1}{Ea1}$$

(1) Capture / recapture model



- **Disadvantages of capture / recapture model**
 - (1) For the engineer's mentality, planting artificial bugs is hard to accept (There is psychological opposition because engineers look for quality).**
 - (2) It is not easy to fake the bugs with reality.**
 - (3) Programmers can easily smell – out artificial bugs.**
 - (4) A new bug may sneak into the source code when artificial bugs are fixed.**
 - (5) The estimated number of the remaining bugs are always under – estimated.**

(1) Capture / recapture model (2 – team model)



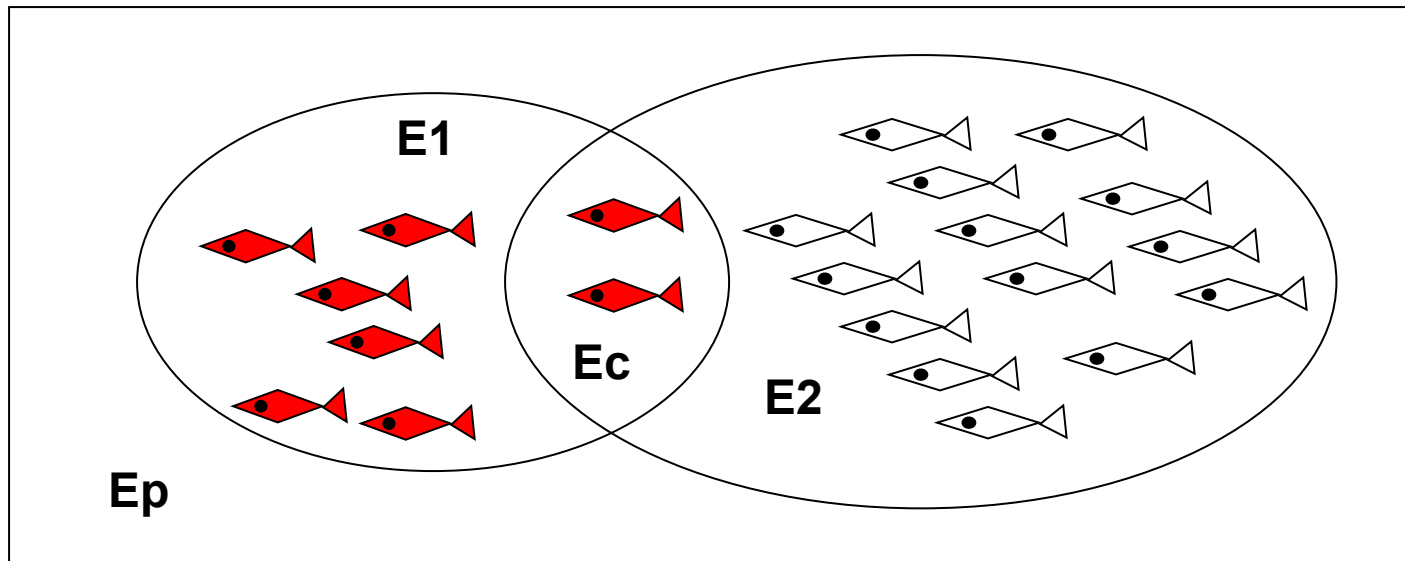
- Capture / recapture model revised

⇒ 2 – team model

(1) 2 team independently test the same software.

(2) If Team 1 detects $E1$ bugs, Team 2 finds $E2$ bugs, and E_c bugs are in common, the estimated number of the remaining bugs E_p is :

$$E_p = \frac{E1 \cdot E2}{E_c}$$

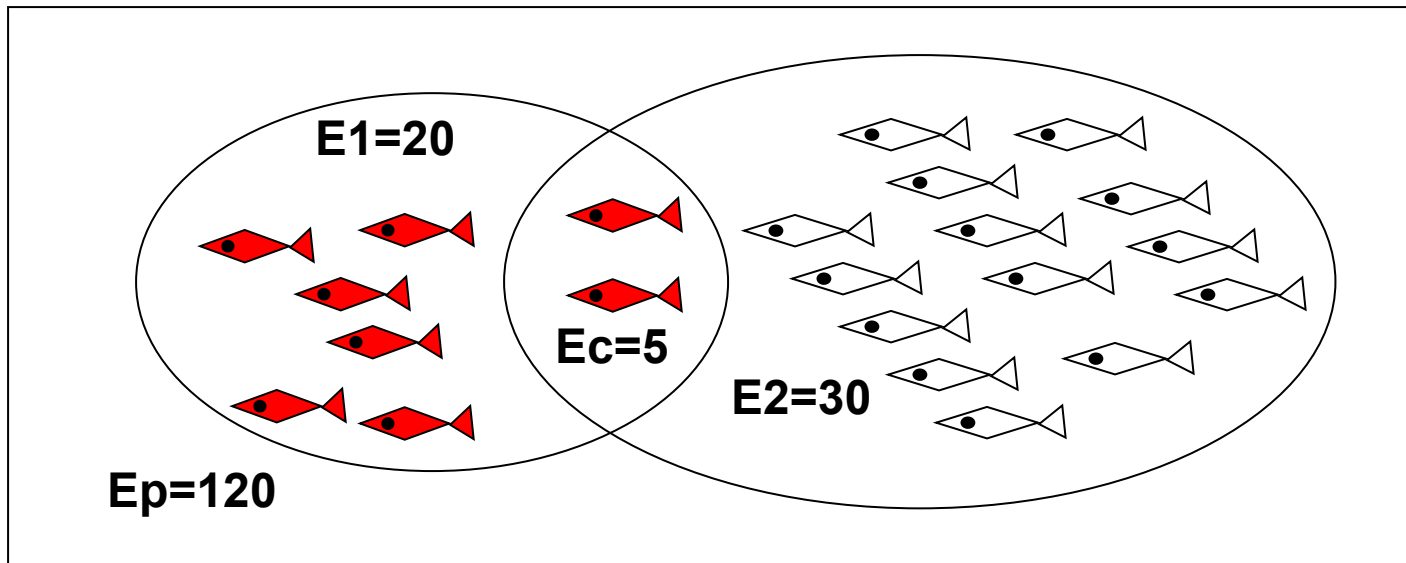


(1) Capture / recapture model (2 – team model)

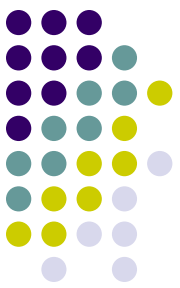


If Team1 detects 20 bugs, Team2 finds 30 bugs, and 5 bugs are in common, the estimated number of the remaining bugs E_p is :

$$E_p = \frac{20 \times 30}{5} = 120$$



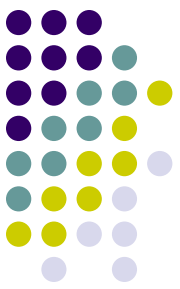
(1) Capture / recapture model (2 – team model)



- **Advantages of 2 – team model**
 - (1) You do not have to plant artificial bugs.**
 - (2) You can detect actual bugs.**

- **Disadvantages of 2 – team model**
 - (1) You have to test all the functions to estimate the bug population.**
 - (2) “2 – team model” takes time.**
 - (3) It is difficult to detect a bug overshadowed by another.**
 - (4) The estimated value is always underestimated.**

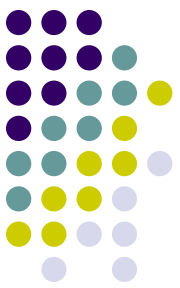
(2) Estimating the Bug Population with Sampling Method (part 1)



- After you designed and generated the test cases, pick up 5 % ~ 10 % of the test cases as a sample
- The estimation steps of the bug population are :
 - (1) Pick up T_s test cases from T_t test cases as a sample.
 - (2) If T_s test cases are executed to detect E_s bugs, the population of the remaining bugs E_p is estimated as :

$$E_p = E_s \left[\frac{T_t}{T_s} \right]$$

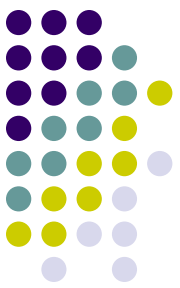
(2) Estimating the Bug Population with Sampling Method (part 2)



Two types of Sampling Test

- **One shot sampling**
 - **Pick up a sample just once to estimate the number of the remaining bugs.**
 - **You can tell the programmers what test cases were sampled.**
- **Iterative Estimation**
 - **Pick up a sample, and not let the programmers inform what test cases have been selected.**
 - **You can repeat the estimation at a milestone of testing and debugging**

(2) Estimating the Bug Population with Sampling Method (part 3)

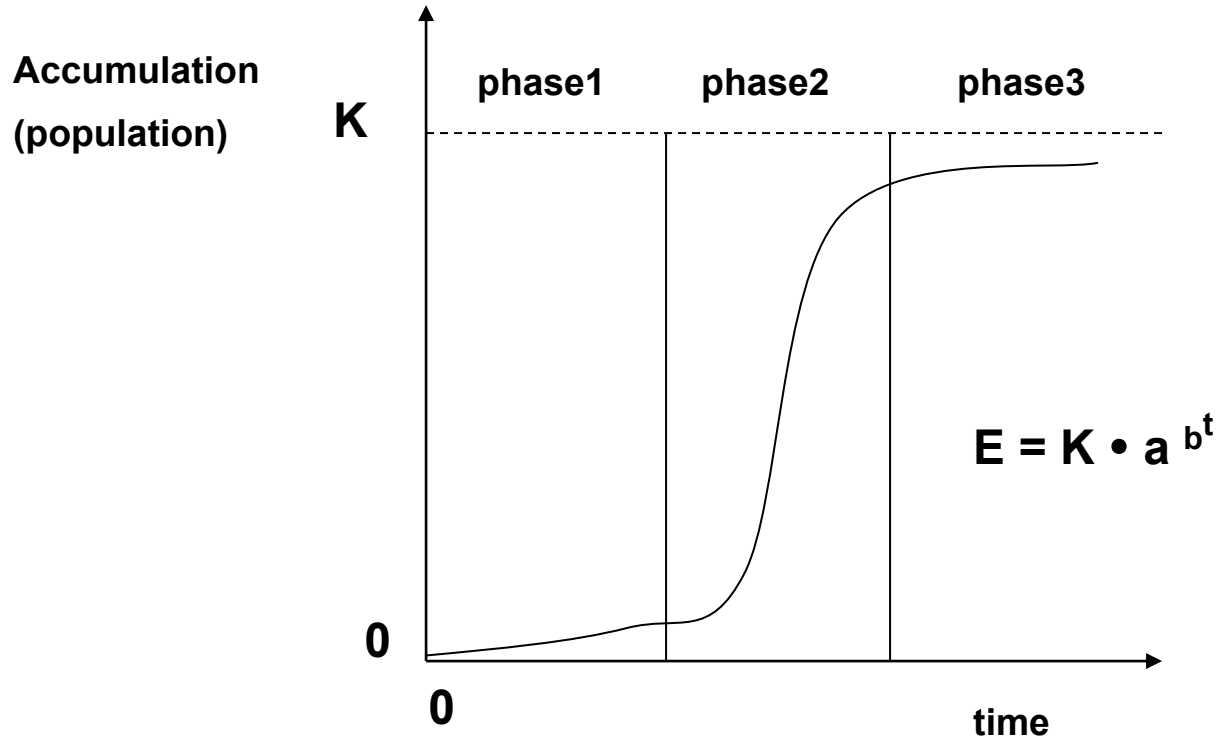


- **Advantages of the Sampling Method**
 - (1) Does not take time to estimate if the sampled test cases are made to test suite (automatic execution).**
 - (2) Estimated bug population is relatively precise**

(3) Estimating Bug Population by Gompertz Curve



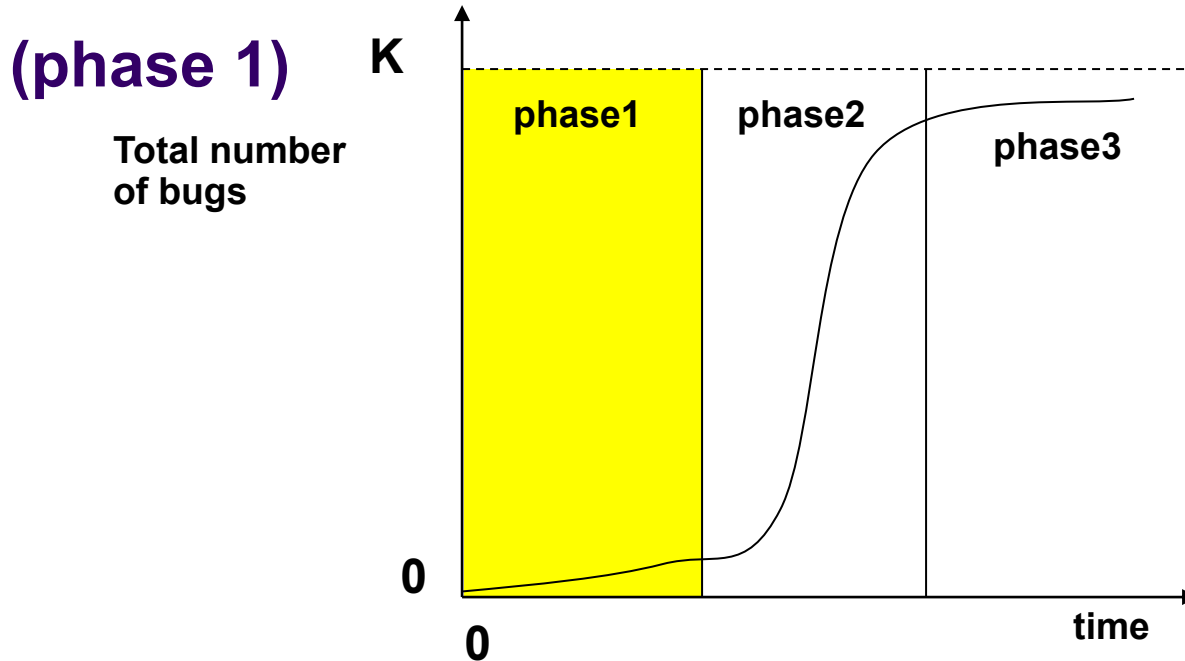
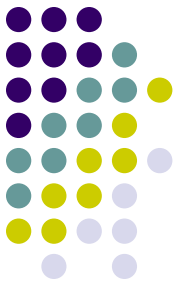
Overview of Gompertz Curve



Gompertz curve is S-shaped formulated curve that present phenomena that gradually grows like population increase.

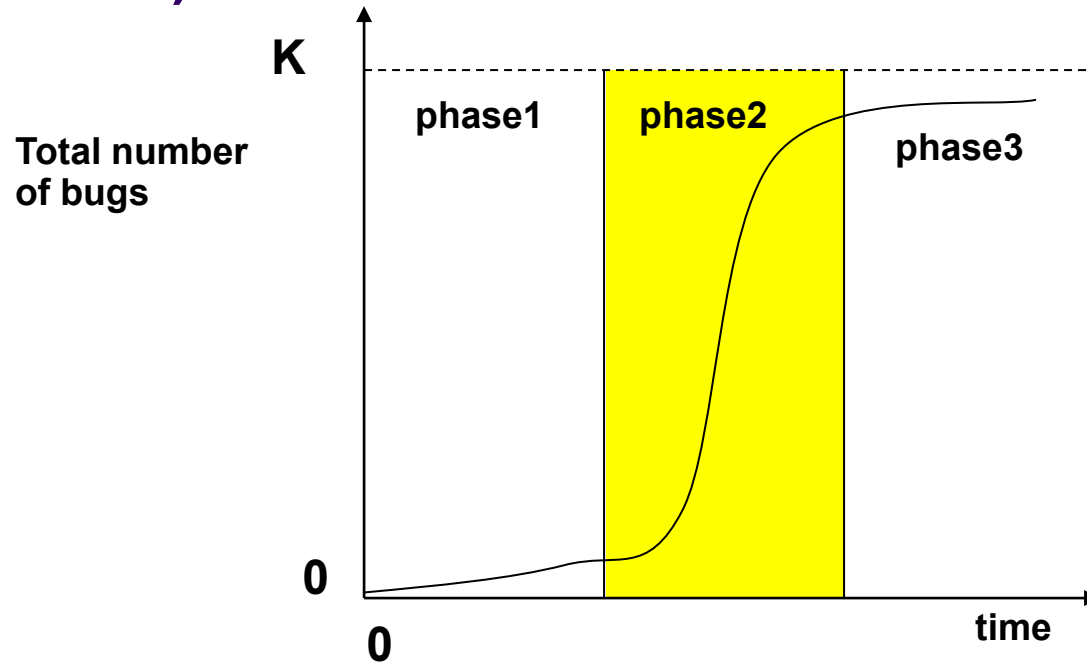
- phase1 : preparation period
- phase2 : rapid growing period
- phase3 : saturation period

(3) Estimating Bug Population by Gompertz Curve



- Early stage of the test period. Many bugs remain undetected
- The 'Main Street' of the basic functions does not go through. Thus errors of basic functions are detected (e.g., missing function). It takes time to fix the bugs.
- A bug must be fixed to execute a test case : it takes time to successfully run a single test case
- A bug overshadows another. A correction of a bug may reveal another behind : it takes time to fix a bug

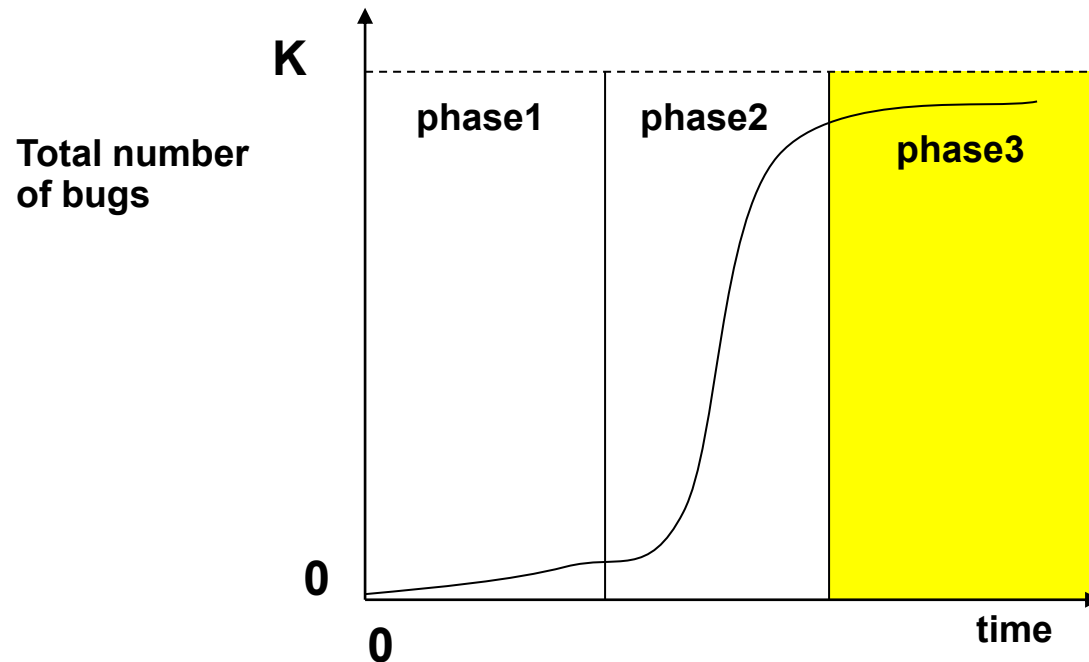
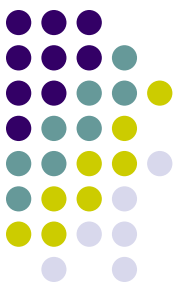
(3) Estimating Bug Population by Gompertz Curve (phase 2)



- 'Basic functions' work without problems.
- Many bugs exist on the border and limitation. Since such bugs can be easy to reproduce, it does not take time for fixing.
- When many bugs are fixed there are chances of generating new bugs. It is highly recommended to run regression test periodically.

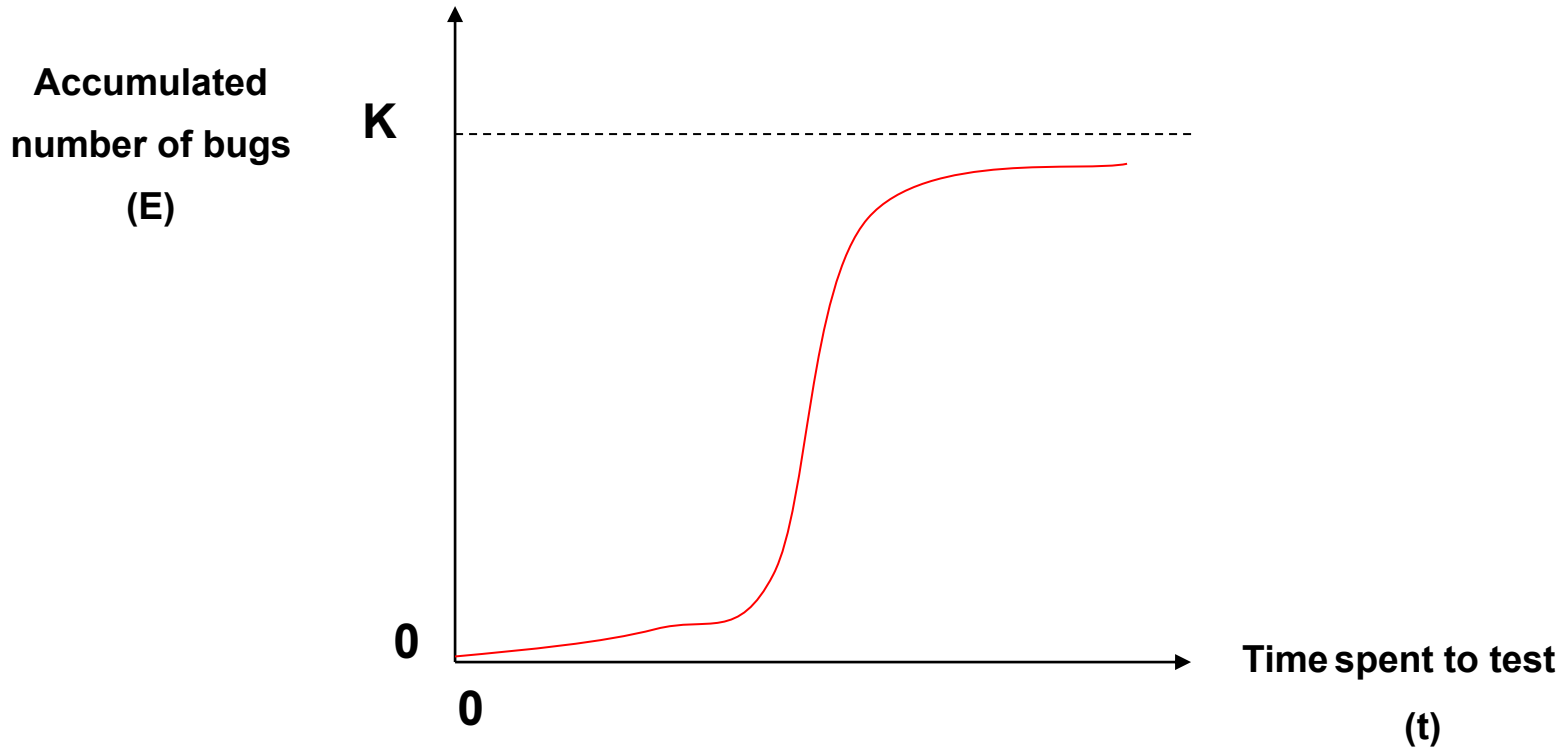
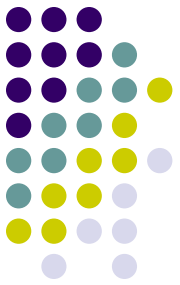
(3) Estimating Bug Population by Gompertz Curve

(phase 3)



- **Basic functions and error cases work properly**
- **Bugs related to special cases and combinational cases (e.g., timing errors), and performance bugs will be detected. Such bugs are hard to reproduce, and very difficult to pinpoint what triggers the bugs**

(3) Estimating Bug Population by Gompertz Curve



$$E = K \cdot a^{b^t}$$

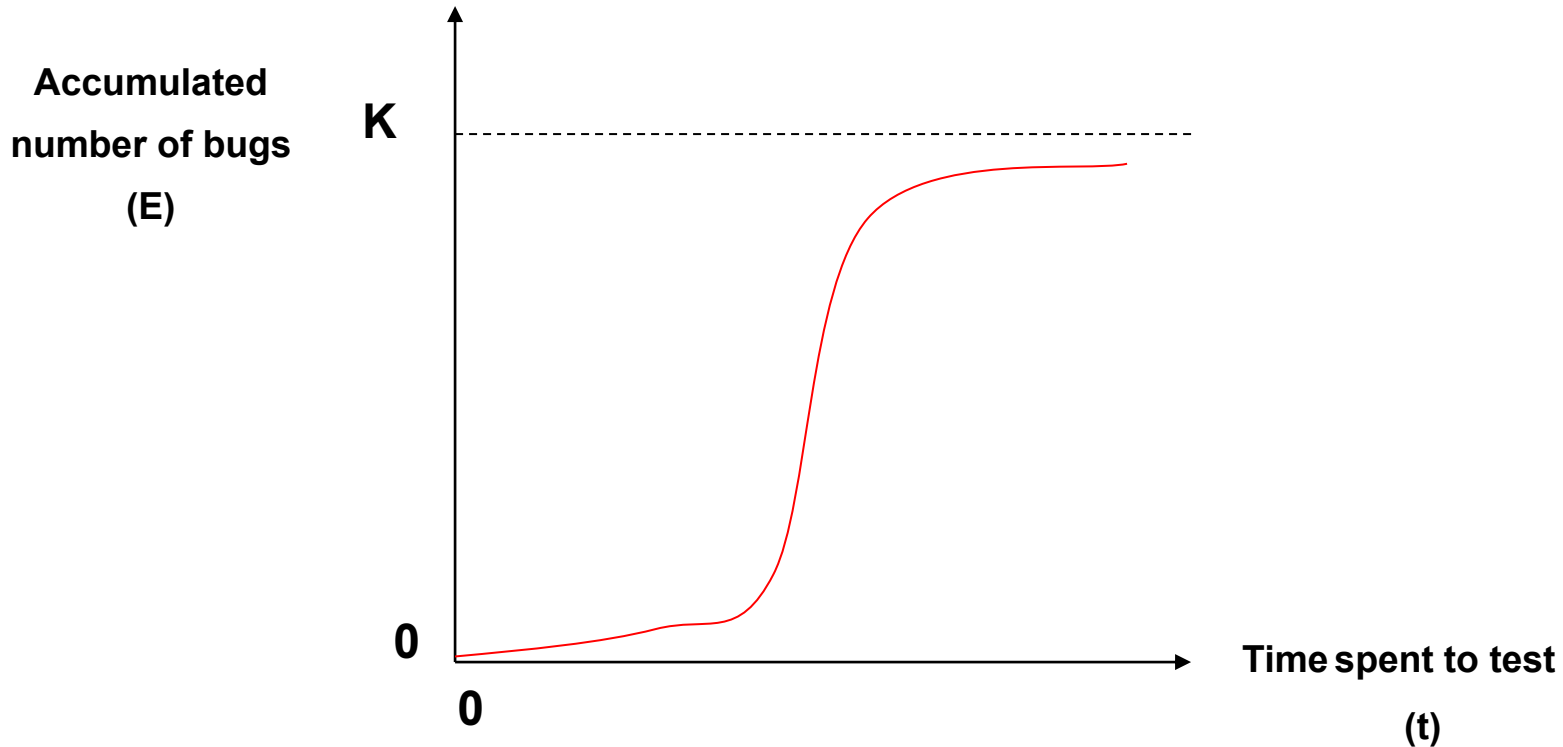
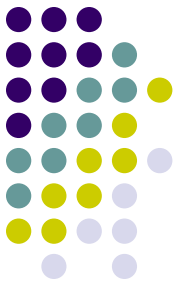
E : Accumulated number of bugs

K : Saturated value at $t \rightarrow \infty$

a, b : parameters larger than 1

Calculated from the
actual values

(3) Estimating Bug Population by Gompertz Curve



Steps to estimate the number of the bug population

(1) Convert $E = K \cdot a^{bt}$ to a polynomial formula $D = At + B$

(2) Get A and B by calculating :

$$D = \log \left[\frac{dE}{dt} \cdot \frac{1}{E} \right], \text{ and } A = \log b, B = \log (\log a \cdot \log b)$$

(3) Estimate a, b, E from A, B

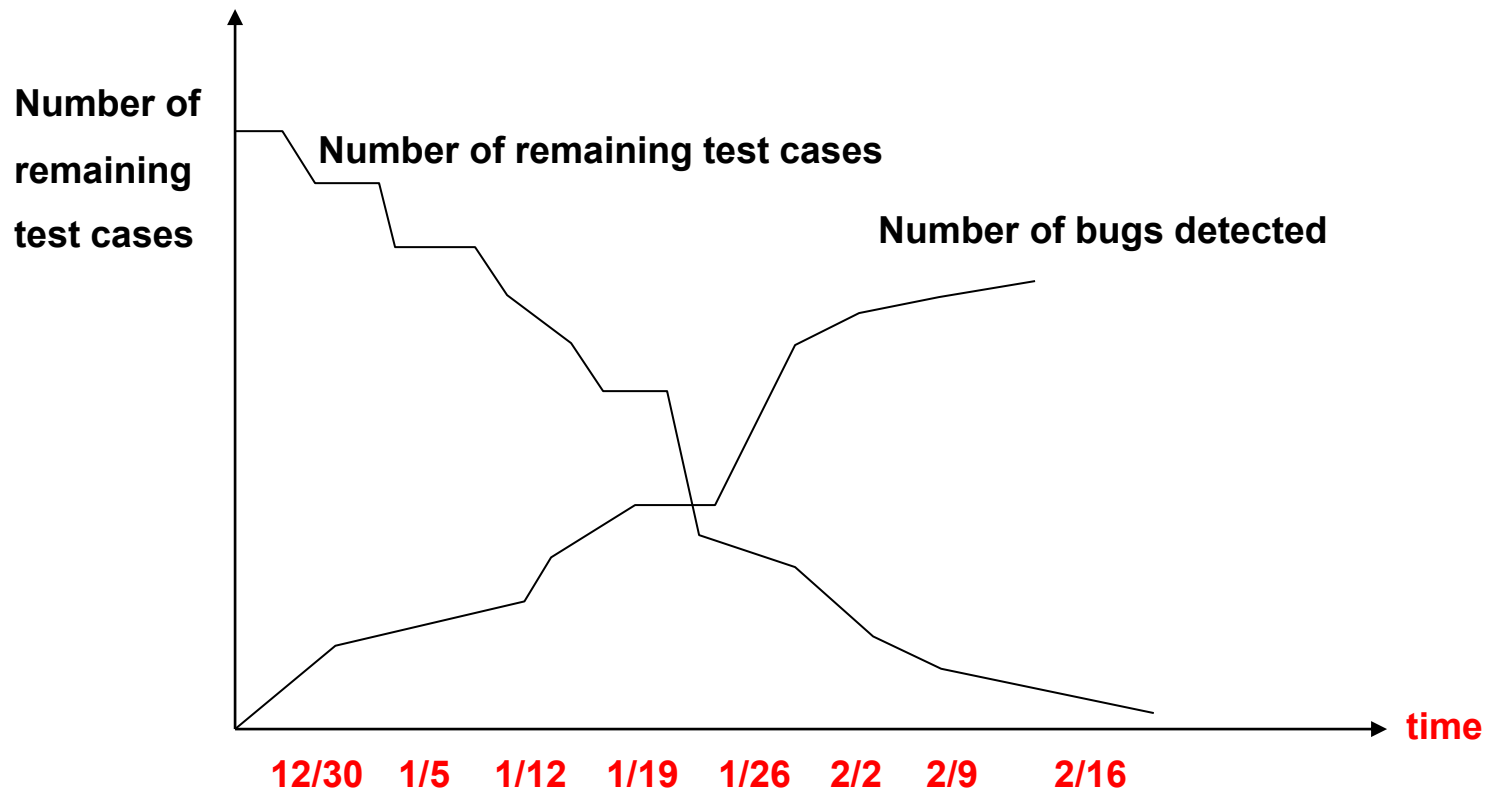
(3) Estimating Bug Population by Gompertz Curve



How should we handle the *time*?

⇒ You may do only 1 hour of testing one day, and 18 hours next day. We cannot assume both days are the same if we look for precise estimation.

⇒ We have to normalize the data, but it will be time - consuming



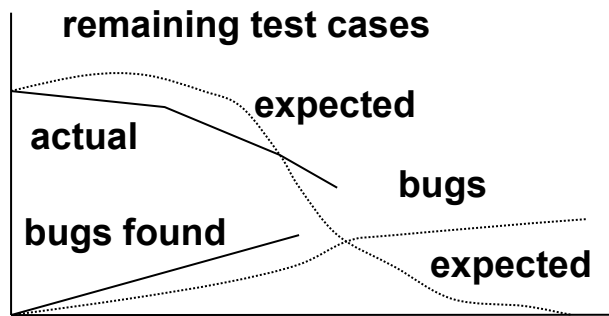
(3) Estimating Bug Population by Gompertz Curve



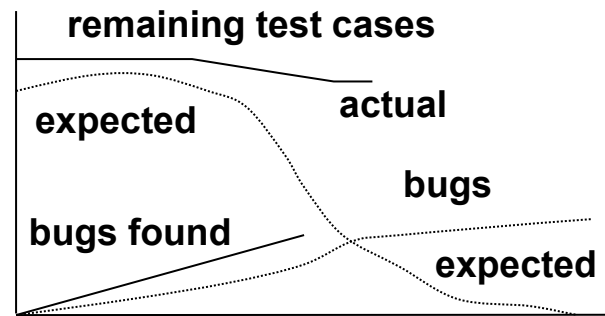
- **Estimated bug population is just estimation : Do not assume it as absolute value.**

⇒ **Make loose estimation with using the upper value and the lower value.**

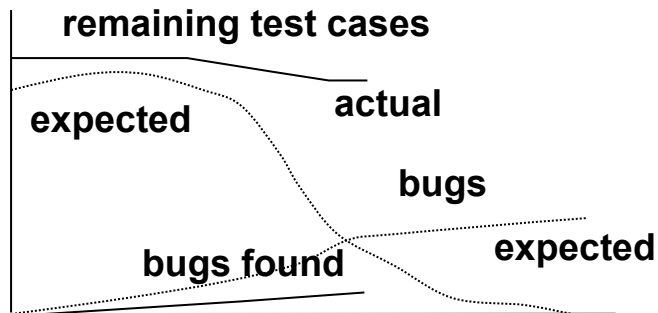
⇒ **Estimate the quality from the pattern of the curves.**



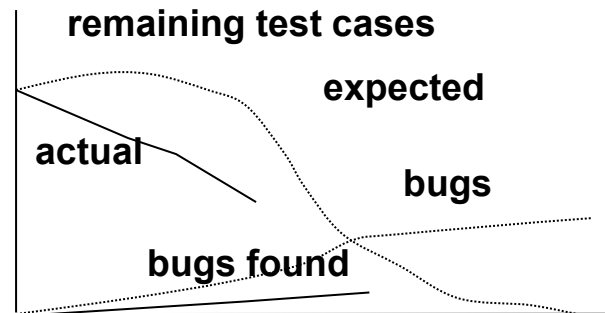
best case



low quality in previous process



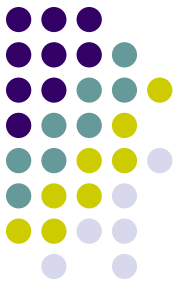
inexperienced testers



low test case quality

(4) Estimating Bug Population by MTBF

(Mean Time Between Failure)



- **MTBF is a average time between one bug is detected and nest one comes out. MTBF is widely used in the reliability theory.**
- **MTBF can be applied when :**
 - (1) In the system testing phase (At least “Main Street” of the software must be open)**
 - (2) You can generate random test data (MTBF assumes actual operation).**
- **Generate and execute many random data, and get the probability of system failure.**

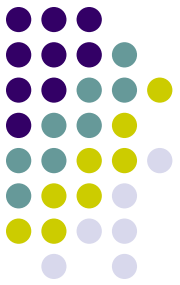
(4) Estimating Bug Population by MTBF (Mean Time Between Failure)



Warning to Random Testing

- **Use random numbers to generate and execute random data.
Is called “monkey operation.”**
- **Usually, test cases are designed to detect targeted bugs (e.g., I will detect bugs on the borders). The random test, which does not assume particular bugs, is not very effective to detect bugs.**
- **Almost all the test data is ignored as error data when executed.**
- **If an actual bug is detected, reproduction is not easy.
⇒ The random testing may not be suitable for analyzing bugs and quality.**

(5) Estimating Bug Population from the Previous Statistical Data



- Use statistical data (bug density) of the project (e.g., the bug density of the previous project was 4.37 / KLOC).
- If the current project developed 80 KLOC, the estimated bug population
Almost all the test data is ignored as error data when executed.

Ep is :

$$\begin{aligned} E_p &= 4.37 \times 80 \\ &= 349.6 \end{aligned}$$

- Easy to calculate, but is relatively accurate in particular if you have the statistical data of the same project of the same domain