The Software Development Process

A personal view

Dr Robert Brady

Brady ltd

Science Park

Cambridge

r.brady@bradytrinity.com

Key reference:
"Debugging the Development Process"
S Maguire
Microsoft Press

The three most important things in software development

1. Bugs

2. Bugs

3. Bugs

My agenda today:-

- Why are bugs inevitable?
- Why do the best companies treat their developers as if bugs were not inevitable?
- How do you structure a development process to get an acceptable level of bugs in practice?

How to lose $70bn

In the late 1980s, IBM lost $70 billion of stock-market value, and gave an entire market away to a previously small company called Microsoft.

According to the popular book “Big Blues”, this was, amongst other things, because it couldn’t write software effectively.

But IBM “did it right”. It followed all the standard rules taught in computer science courses at the time:

- Get the design right before you write the code
- Write complete documentation
- Get it right first time
- Use formal methods, design walk-throughs etc. to satisfy yourself that the code is bug-free

So what went wrong?

Size is important

Bytes

100b-1kb	Typical punch-card program (The IBM development method was probably developed for this type of program)
2kb-10kb	Typical software module Typical computer science project(?)
16kb	Operating system of Sinclair Spectrum
200Kb	Our first software product – 1986
18 Mb	Human Genome estimate (estimate - 30k genes * protein size 800)
64Mb	Xbox RAM
100Mb	Our current software product (code)
300Mb	Typical database used with our product
1Gb – 2 Gb	Windows 2000 with associated products
8Gb	Permanent storage on Xbox

How size affects the basic assumptions

	Punch-card program	2kb of code	Large program
Complete the design in advance	Almost essential	Difficult	Too complex - not possible
Complete the documentation in advance	Highly desirable	Difficult	Too complex - not possible
Prove it is bug-free	Very difficult mathematical challenge	Too complex - not possible	Too complex - not possible
“Right first time”	A worthy goal	Too complex - not possible	Too complex - not possible

Conclusion: bugs are inevitable

(but do not let this become an excuse)

The consequences of Bug denial

Until recently, computer science courses used to teach that you must design in advance, document in advance, and write bug-free code first time. ...

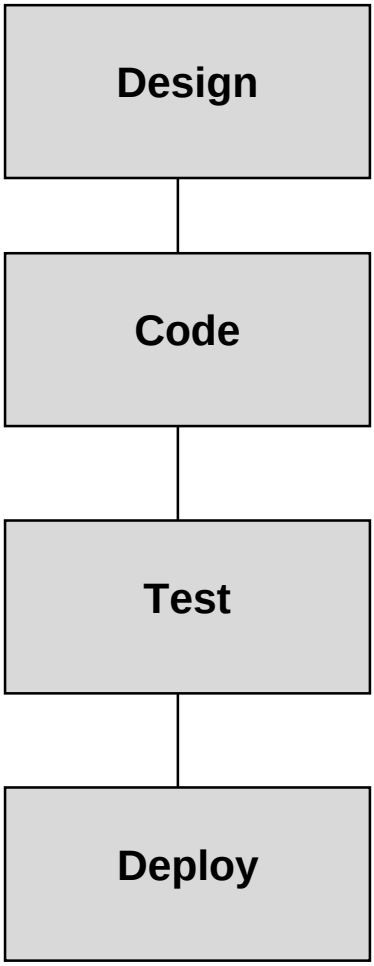
… they were in denial that bugs are inevitable …

… and so were the students we used to hire …

… The students believed that management was ineffective or stupid because its policies differed from those taught by the university professor …

… and my management used to have a policy not hiring people who had graduated in Computer Science at a university because it “just didn’t work”

Waterfall model

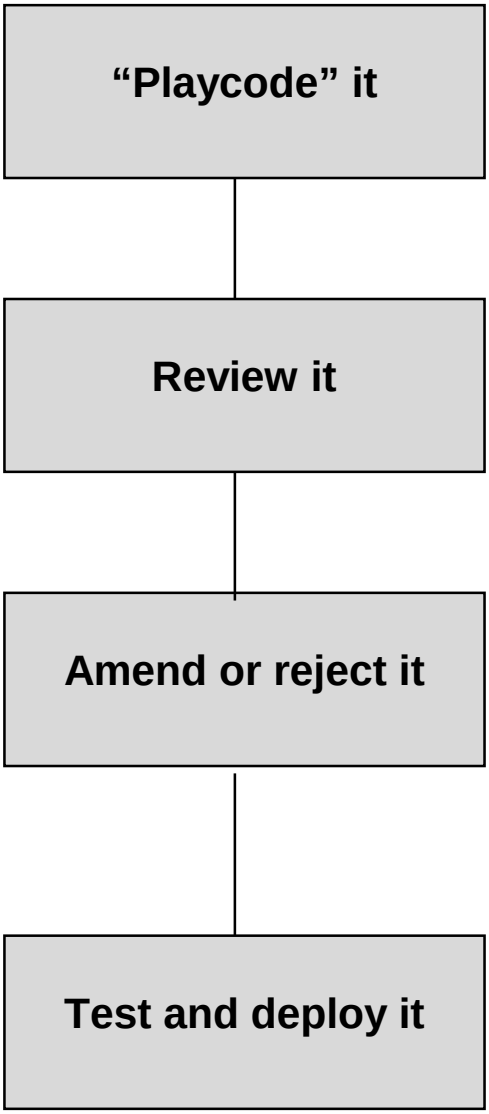


Strengths and pitfalls of the Waterfall Model

Good for small modules or sub-units, particularly if you can have simple and well-specified interface.

- IBM implemented this model by having DIFFERENT people in each stage. This gave people posh-sounding job titles (“Analyst” etc.), but caused very bad communication that killed their projects.
- Like Microsoft, we have a policy: “We do not have any programmers” We have developers. They are responsible for seeing the whole thing through.

The “Prototype” Model

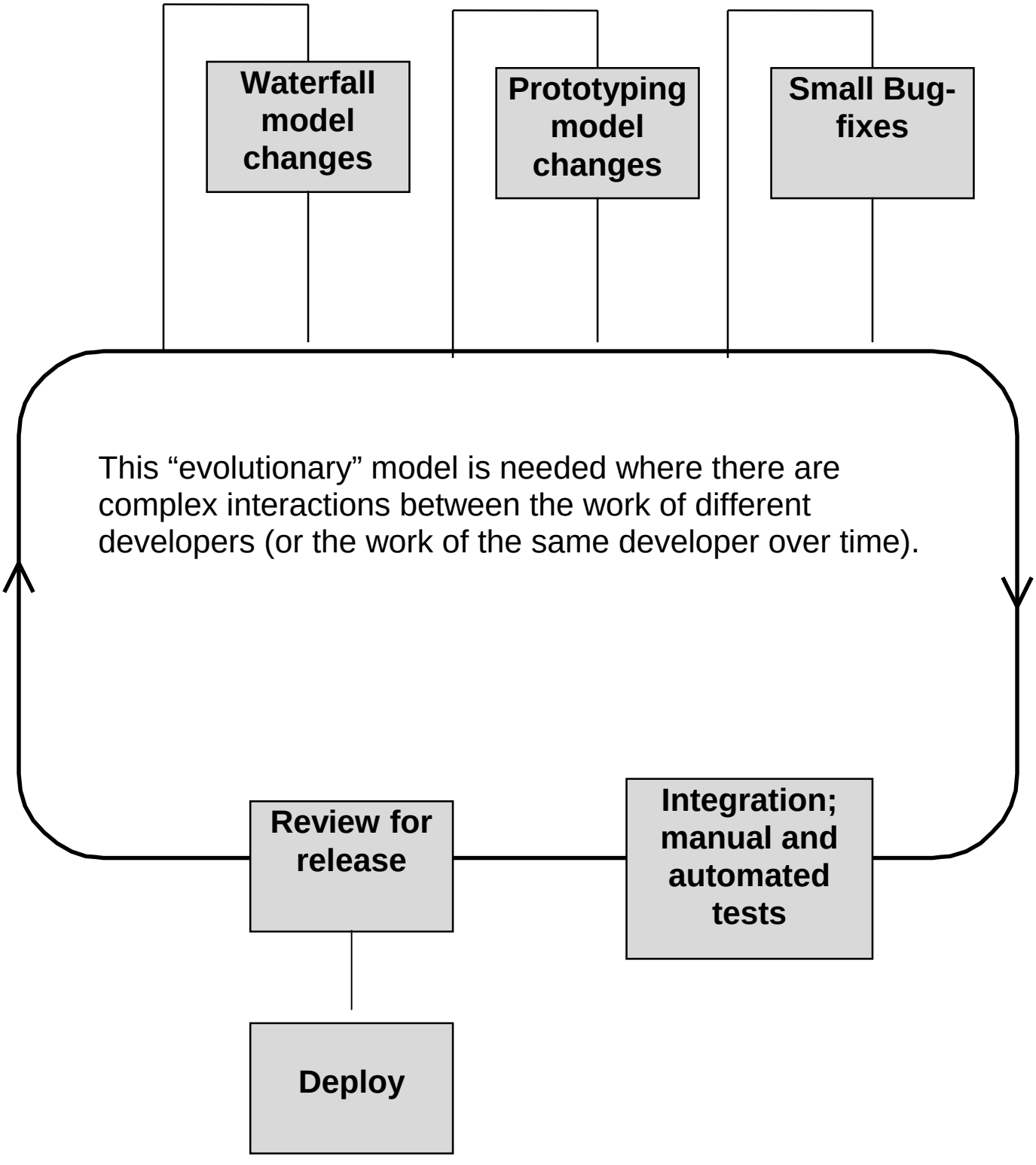


Strengths and pitfalls of the “Prototype” model

- Good where there are significant project risks or unknowns - e.g. external software, new techniques or methods, or can’t decide between alternatives.

- Not very predictable (a big problem in contracted developments)

The “evolutionary” model
- what everyone does in practice
(whatever they call it)



A quiz

You are the manager of a small (2 person) software development/test team. They come to you with a problem and a proposed solution. Do you approve it?

Problem

- We need to implement 10 features. We have reviewed the designs, we now need to code and test them.

- Time is very tight. We will have to pull out all the stops to do it by the contracted deadline of next month

- John (the developer) is the best person to do the coding

- Richard (the test engineer) is the best person to do the testing

Proposed solution

- John and Richard work closely together to accelerate the development phase

- John codes the features and makes quick releases to Richard during development

- Richard provides testing feedback during development

- After this development phase, the software goes into the normal release cycle for testing/bugfix

If you approve the plan

- You will send a message to your developers that bugs don't matter – you can "throw them over the wall" and someone else will find them for you

- You will accelerate developers who produce sloppy code and slow down developers who produce good code

- The process will be inefficient, eg
  - the developer has a rough idea which areas will be buggy, he can home in on these
  - The developer has tools ("debuggers") to find bugs which the tester doesn't have
  - The developer will have to constantly communicate with the tester on what's changed, this slows them both down
  - The tester will be inefficient because silly bugs will stop him running his automated tests

- When you get to the original deadline
  - your project will probably have all the features
  - but the product probably won't work well enough to run the automated tests, so you cannot ship
  - You won't be able to advise the customer of the new ship date, because the automated tests don't work and they might (or might not) uncover something when they do run
  - It will be too late to take corrective action

If you reject the plan (developer has to test his code before release)

- Your team will be forced to make the hard project decisions, eg
  - Go back to the design stage for feature number 3 – can we implement it more simply?
  - Cut feature number 6 – it's not strictly in the specification
  - Advise the customer there is a risk. Does he want a delay or does he want feature number 7 in a later release?
  - Request more resources (a long shot…)

- Your team will work more efficiently
  - The tester will always work on code that is basically stable (so he can develop his regression tests etc.)
  - The developer will be rewarded for producing quality code, not for producing features that destabilise the product

- Your team will be able to plan the project
  - If a feature is in the product then it will "basically work"
  - The team (and you) can now monitor progress
  - You can get test results and customer feedback early on the features you have implemented
  - Management make a decision to ship, with an shorter freeze-time for stabilisation

Developers, Bugs, Attitude and Organisation

- Convince your developers to be affronted if someone finds a bug in their code

"I believe that the final bug in T_EX was discovered and removed on November 27, 1985. But if, somehow, an error still lurks in the code, I will gladly pay a finder's fee of $20.48 to the first person who discovers it"

Donald Knuth

- Developers must fix their own bugs

1. Developers get feedback early on, and can change their development process to produce better code earlier on
2. Developers realise that bugs are important
3. Slows down buggy developers
4. Speeds up good developers
5. Lets you plan projects