The software development process
A personal view

Robert Brady

robert.brady@cl.cam.ac.uk

17 October 2013

Software development

Why any management at all?

‘Just recruit great developers’

- They are 10-50 times more productive than average developers
- who are 10-50 times more productive than poor developers
- Management will just get in the way

1980's – successful anti-management revolution

According to 'Big Blues: the Unmaking of IBM'

- In the late 1980's, IBM lost $70 billion of stock value and gave an entire market away to a small company called Microsoft
- Mainly because it couldn't write software effectively.

1980's – successful anti-management revolution

According to 'Big Blues: the Unmaking of IBM'

- In the late 1980's, IBM lost $70 billion of stock value and gave an entire market away to a small company called Microsoft
- Mainly 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, before release
- Regard other methods (eg Microsoft's) as "hacking"

1980's – successful anti-management revolution

According to 'Big Blues: the Unmaking of IBM'

- In the late 1980's, IBM lost $70 billion of stock value and gave an entire market away to a small company called Microsoft
- Mainly 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, before release
- Regard other methods (eg Microsoft's) as "hacking"

So what went wrong?

IBM method based on fundamental misconception

Evolution of size

0.1-1kb	Typical punch-card program The IBM development method was probably developed for this type of program

IBM method based on fundamental misconception

Evolution of size

0.1-1kb	Typical punch-card program
	The IBM development method was
	probably developed for this type of program
2kb-10kb	Typical software module/class
	Typical computer science project

IBM method based on fundamental misconception

Evolution of size

0.1-1kb	Typical punch-card program
	The IBM development method was
	probably developed for this type of program
2kb-10kb	Typical software module/class
	Typical computer science project
16kb	Operating system of Sinclair Spectrum

IBM method based on fundamental misconception

Evolution of size

0.1-1kb	Typical punch-card program
	The IBM development method was
	probably developed for this type of program
2kb-10kb	Typical software module/class
	Typical computer science project
16kb	Operating system of Sinclair Spectrum
200kb	Our first software product – 1986

IBM method based on fundamental misconception

Evolution of size

0.1-1kb	Typical punch-card program
	The IBM development method was
	probably developed for this type of program
2kb-10kb	Typical software module/class
	Typical computer science project
16kb	Operating system of Sinclair Spectrum
200kb	Our first software product – 1986
18 Mb	Human Genome – active code
	(30k genes * protein size 800)
	Number varies from year to year

IBM method based on fundamental misconception

Evolution of size

0.1-1kb	Typical punch-card program
	The IBM development method was
	probably developed for this type of program
2kb-10kb	Typical software module/class
	Typical computer science project
16kb	Operating system of Sinclair Spectrum
200kb	Our first software product – 1986
18 Mb	Human Genome – active code
	(30k genes * protein size 800)
	Number varies from year to year
20Mb	Our current software (one product)

IBM method based on fundamental misconception

Evolution of size

0.1-1kb	Typical punch-card program
	The IBM development method was
	probably developed for this type of program
2kb-10kb	Typical software module/class
	Typical computer science project
16kb	Operating system of Sinclair Spectrum
200kb	Our first software product – 1986
18 Mb	Human Genome – active code
	(30k genes * protein size 800)
	Number varies from year to year
20Mb	Our current software (one product)
750Mb	Human genome - including rubbish code
	(3 x 109 base-pairs)

IBM method based on fundamental misconception

Evolution of size

0.1-1kb	Typical punch-card program
	The IBM development method was
	probably developed for this type of program
2kb-10kb	Typical software module/class
	Typical computer science project
16kb	Operating system of Sinclair Spectrum
200kb	Our first software product – 1986
18 Mb	Human Genome – active code
	(30k genes * protein size 800)
	Number varies from year to year
20Mb	Our current software (one product)
750Mb	Human genome - including rubbish code
	(3 x 109 base-pairs)
4Gb	Windows Vista

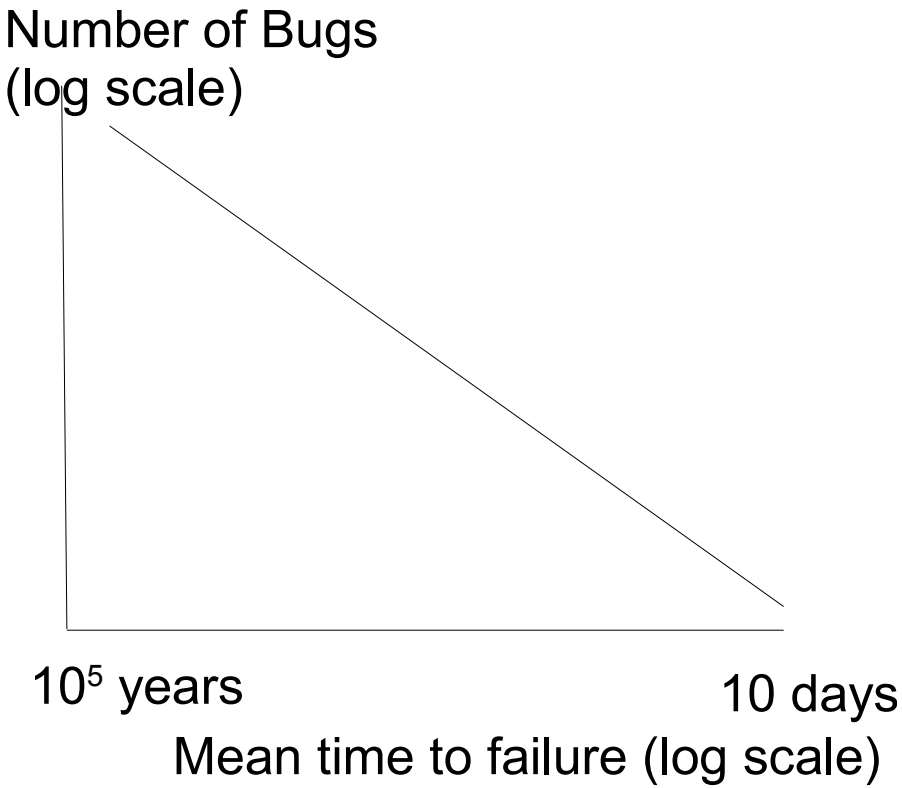
IBM method based on fundamental misconception

Evolution of size

0.1-1kb	Typical punch-card program
	The IBM development method was
	probably developed for this type of program
2kb-10kb	Typical software module/class
	Typical computer science project
16kb	Operating system of Sinclair Spectrum
200kb	Our first software product – 1986
18 Mb	Human Genome – active code
	(30k genes * protein size 800)
	Number varies from year to year
20Mb	Our current software (one product)
750Mb	Human genome - including rubbish code
	(3 x 109 base-pairs)
4Gb	Windows Vista
300Gb	Storage on my laptop

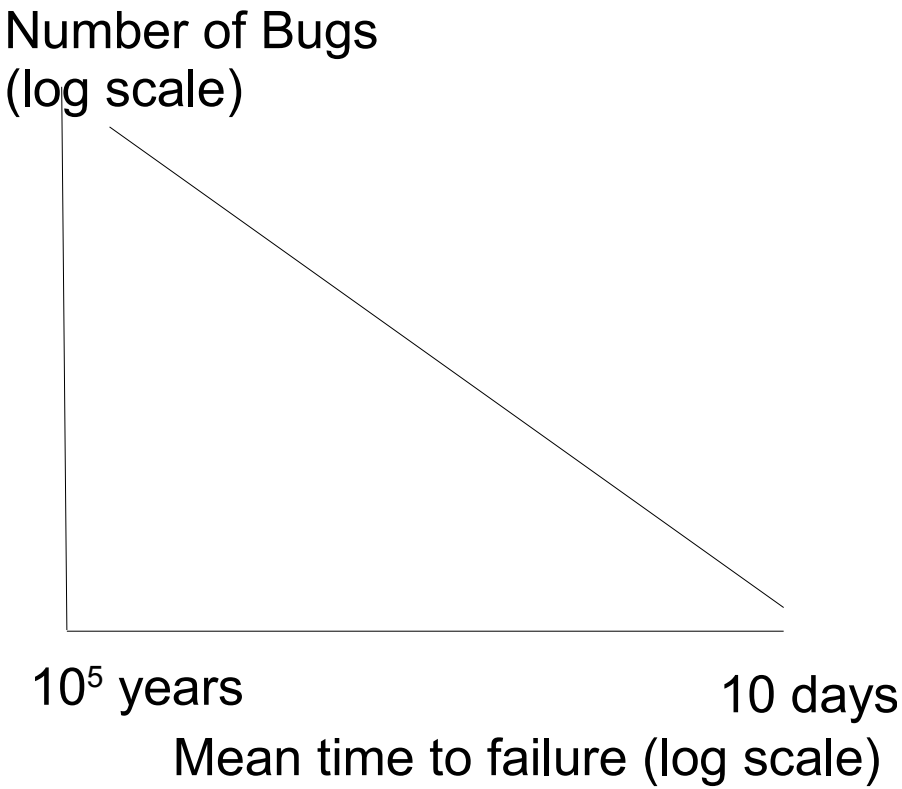
Software development

IBM seminal measurements 1984



Adams E. N., Optimising preventive maintenance of software products, IBM Journal of Research & Development, Vol. 28, issue 1 pp 2–14 (1984)

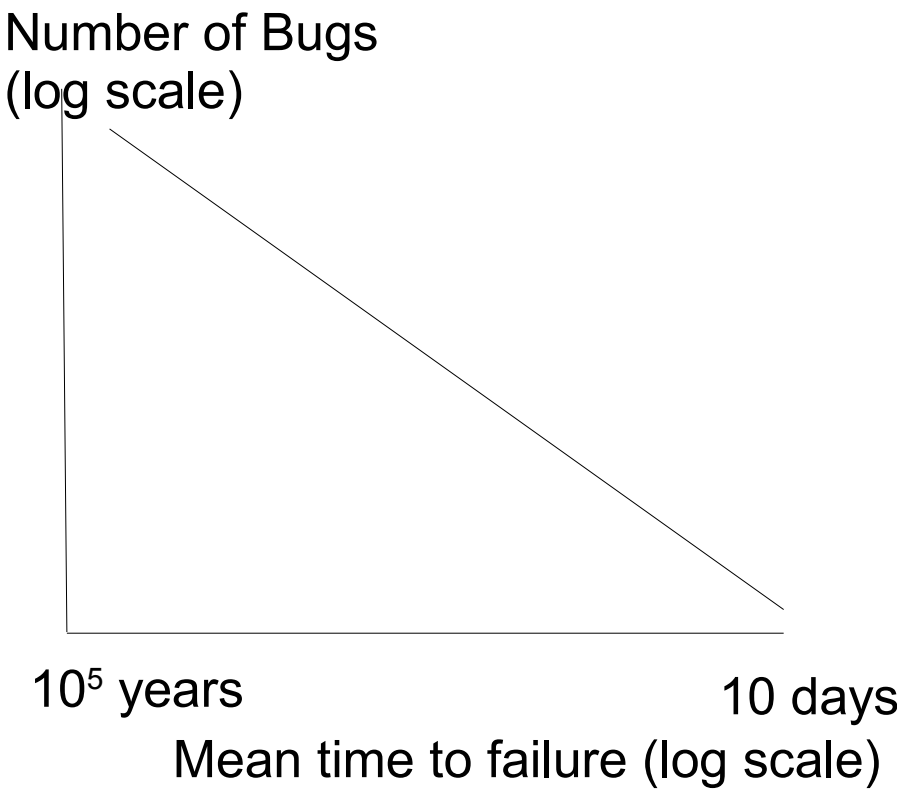
IBM seminal measurements 1984



Period	average	bugs	mttf
10–20d	15d	1	15d

Adams E. N., Optimising preventive maintenance of software products, IBM Journal of Research & Development, Vol. 28, issue 1 pp 2–14 (1984)

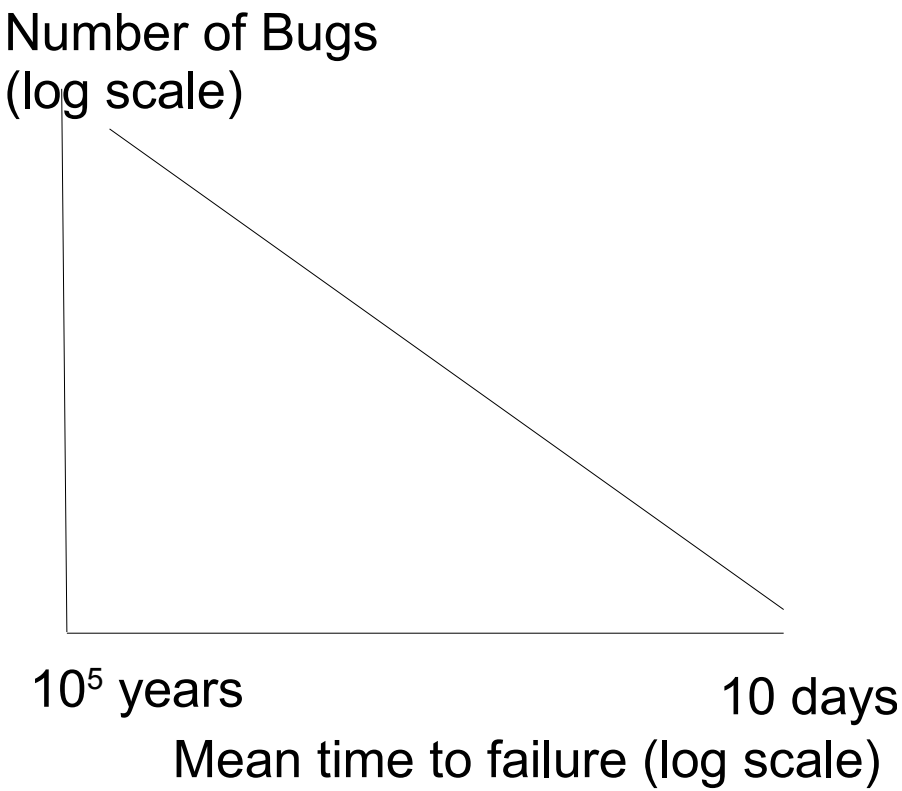
IBM seminal measurements 1984



Period	average	bugs	mttf
10–20d	15d	1	15d
20–40d	30d	2	15d

Adams E. N., Optimising preventive maintenance of software products, IBM Journal of Research & Development, Vol. 28, issue 1 pp 2–14 (1984)

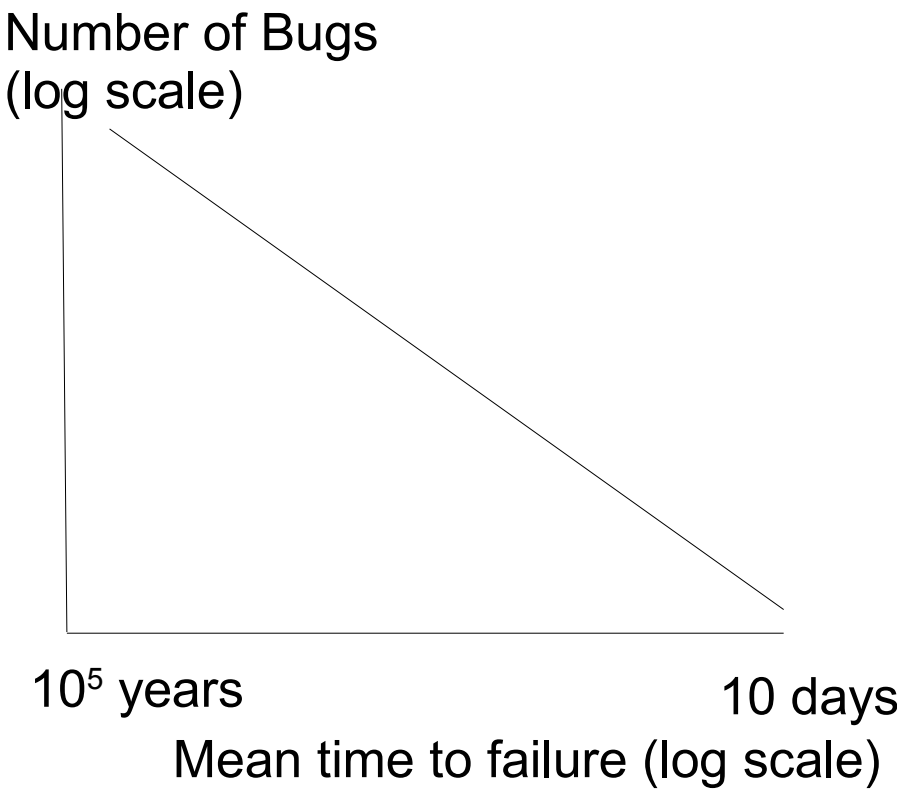
IBM seminal measurements 1984



Period	average	bugs	mttf
10–20d	15d	1	15d
20–40d	30d	2	15d
40–80d	60d	4	15d

Adams E. N., Optimising preventive maintenance of software products, IBM Journal of Research & Development, Vol. 28, issue 1 pp 2–14 (1984)

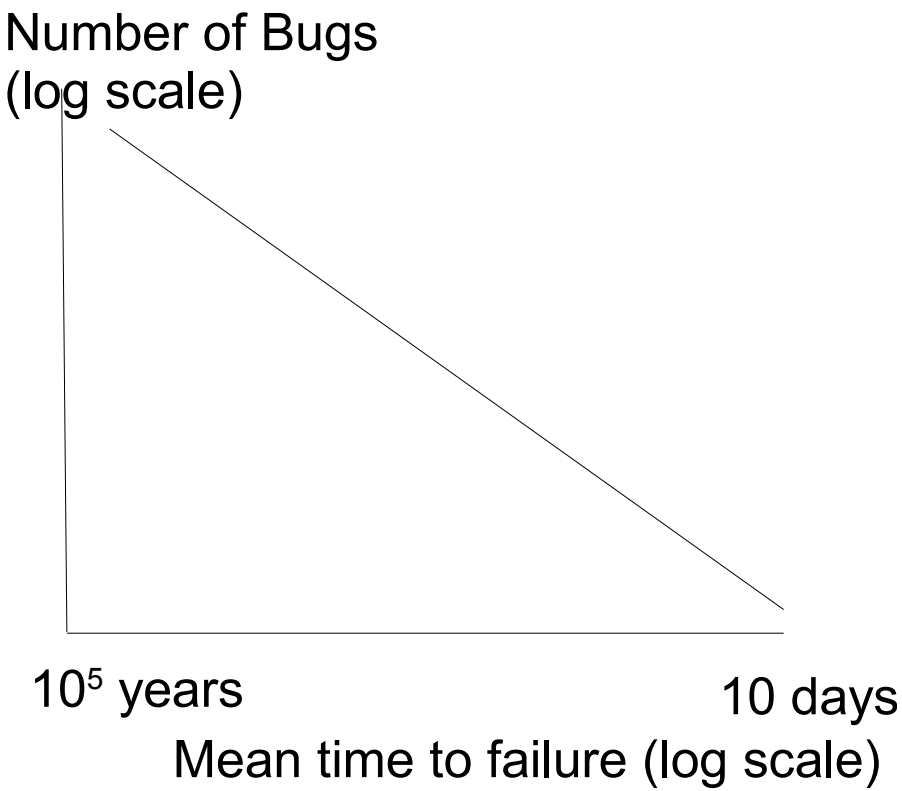
IBM seminal measurements 1984



Period	average	bugs	mttf
10–20d	15d	1	15d
20–40d	30d	2	15d
40–80d	60d	4	15d
80–160d	120d	8	15d

Adams E. N., Optimising preventive maintenance of software products, IBM Journal of Research & Development, Vol. 28, issue 1 pp 2–14 (1984)

IBM seminal measurements 1984



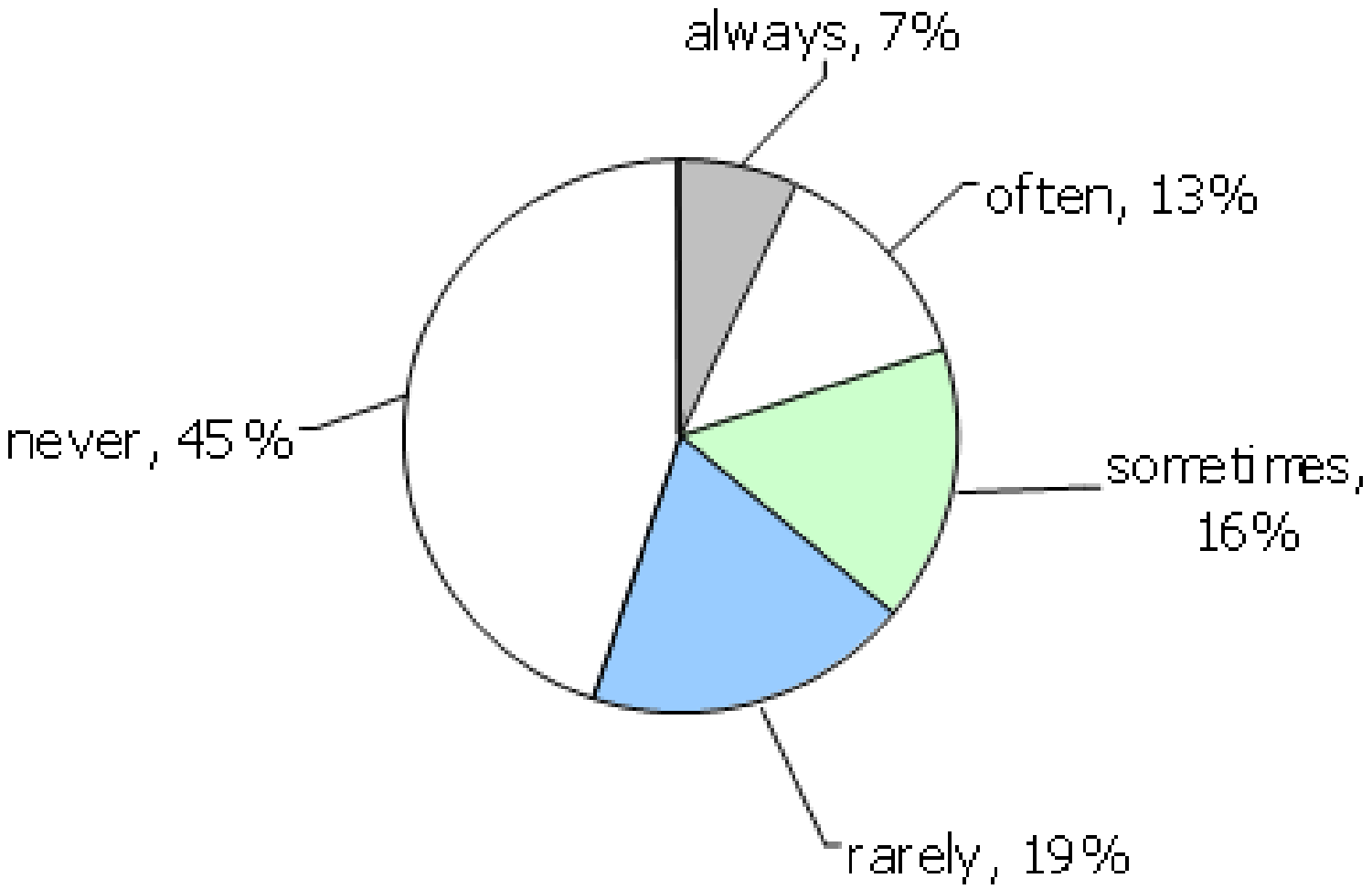
Period	average	bugs	mttf
10–20d	15d	1	15d
20–40d	30d	2	15d
40–80d	60d	4	15d
80–160d	120d	8	15d

Hard-to-find bugs dominate

Adams E. N., Optimising preventive maintenance of software products, IBM Journal of Research & Development, Vol. 28, issue 1 pp 2–14 (1984)

Statistics of large projects (1)

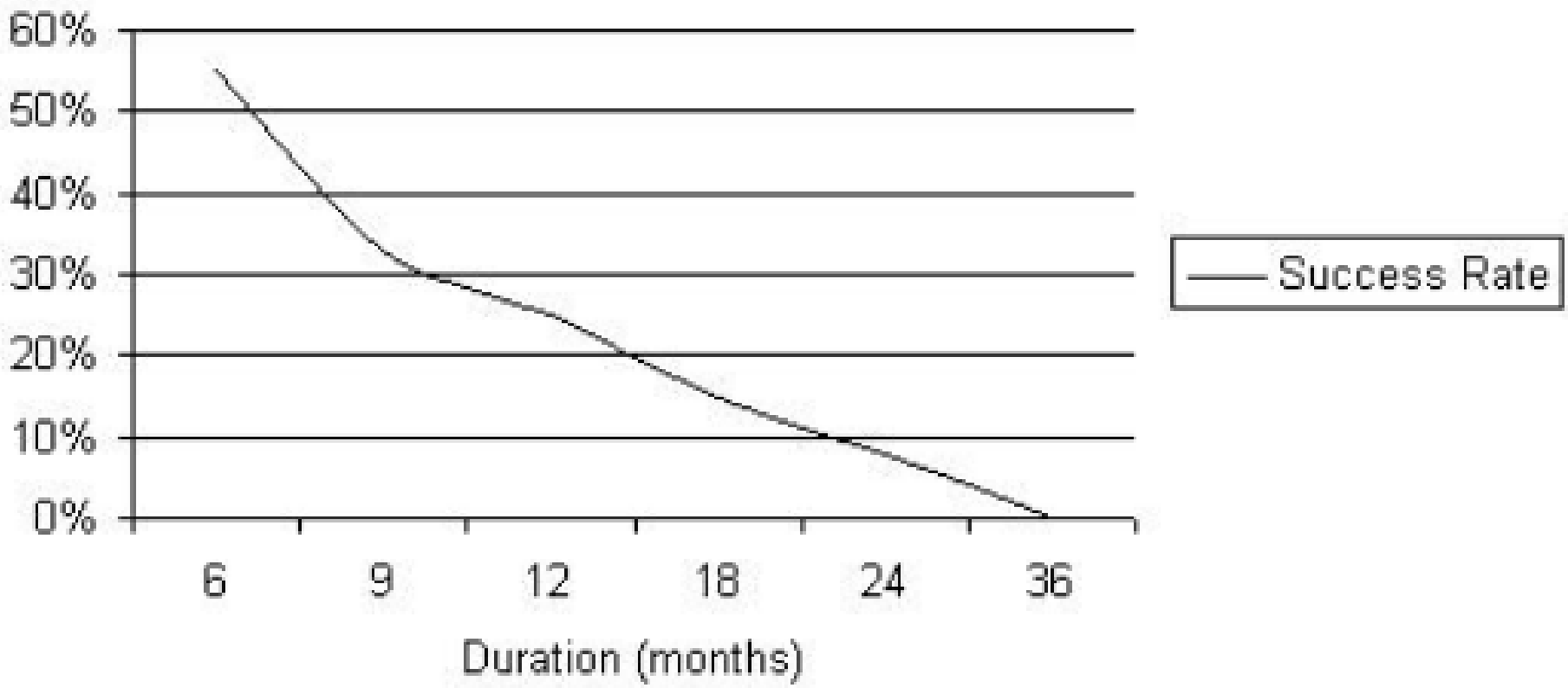
Actual use of requested features (Johnson 2002)



Only a viable ‘business’ in public contracts (eg NHS patient records)

Statistics of large projects (2)

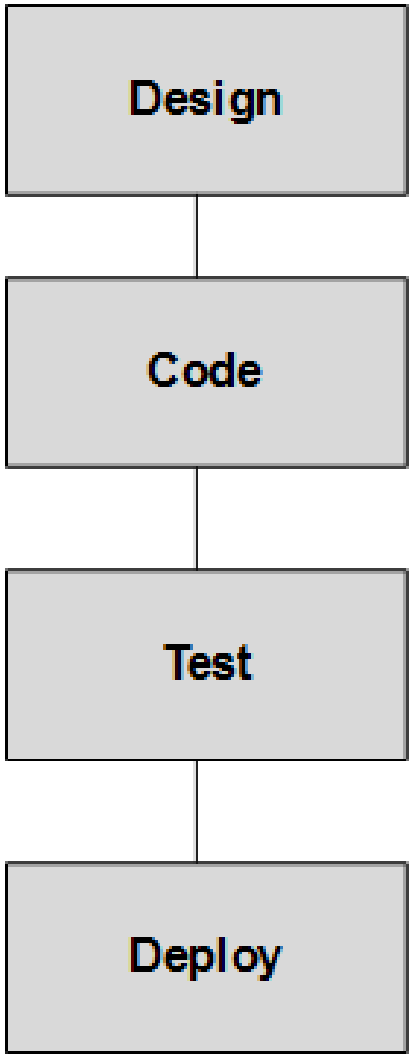
Success rate of projects (Johnson 1998)



Good business for litigation lawyers, not for small software houses

Software development

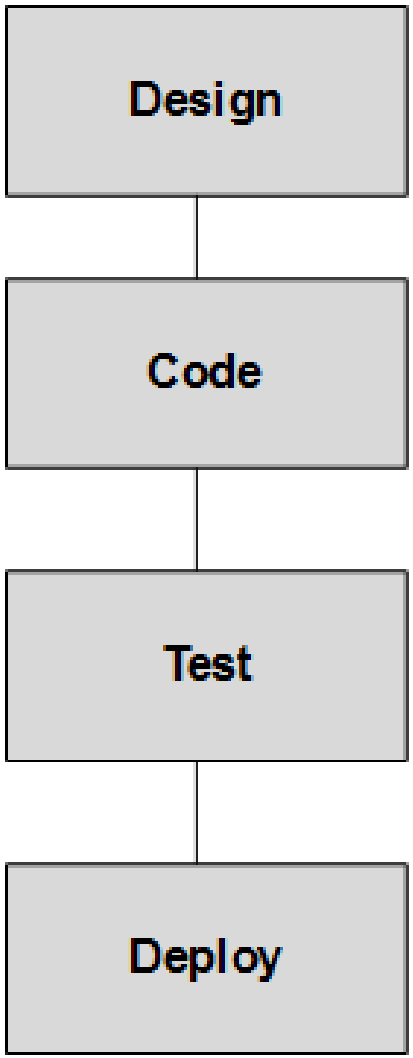
Waterfall model



Mainstay of development process

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

Waterfall model



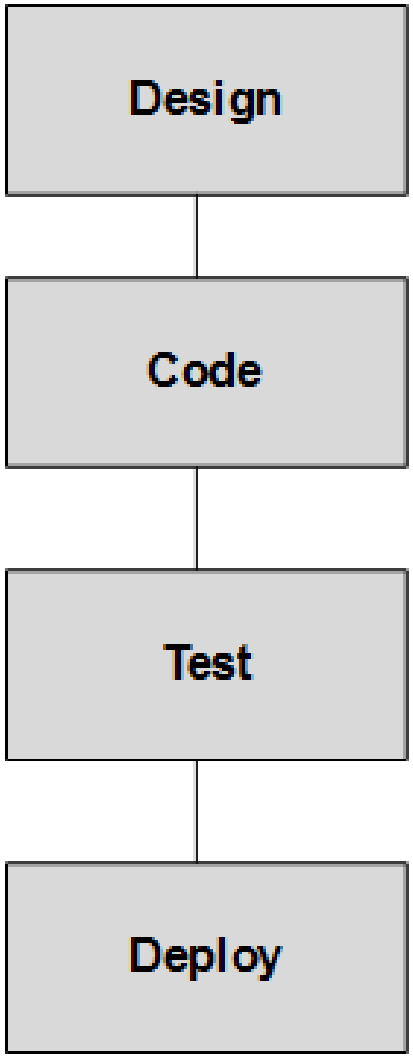
Mainstay of development process

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

Danger

- Different people for each stage
- lost information = failure
- ‘We don’t have programmers, we have developers’

Waterfall model



Mainstay of development process

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

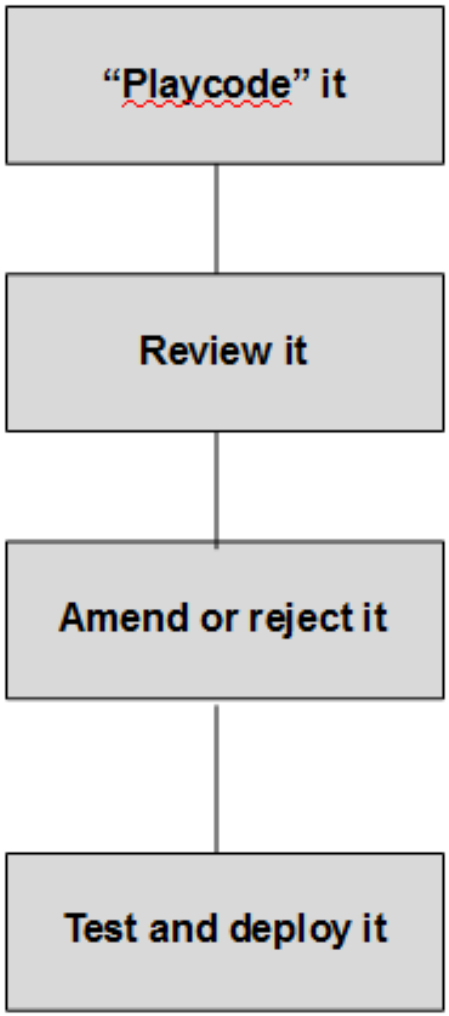
Danger

- Different people for each stage
- lost information = failure
- ‘We don’t have programmers, we have developers’

Communication and developer induction

- Sit with a user
- Agree small issues/problems
- Fix some yourself (nobody else)

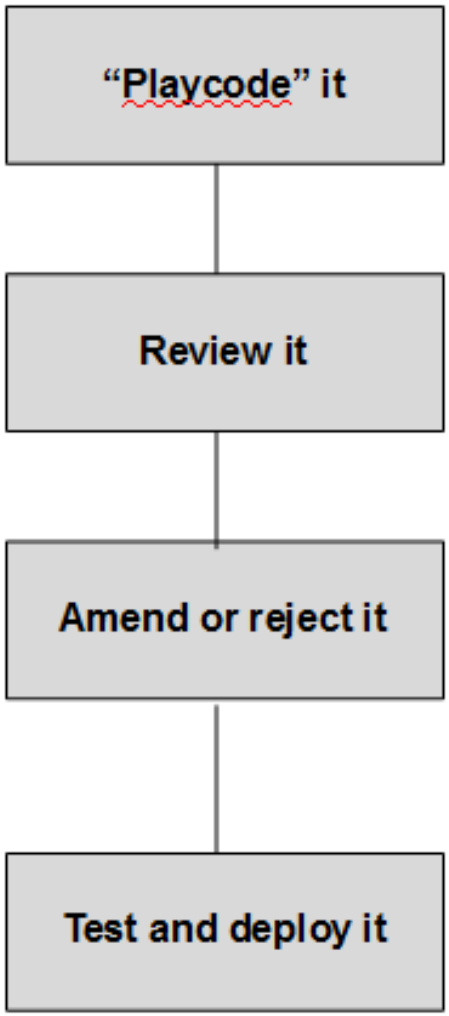
Prototyping



Good where there are significant project risks or unknowns

- external software
- new techniques or methods
- can’t decide between alternatives

Prototyping



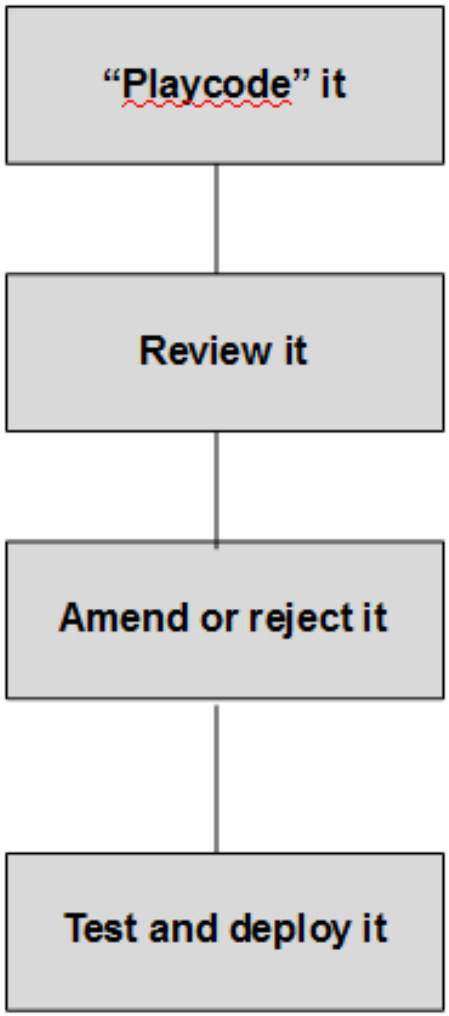
Good where there are significant project risks or unknowns

- external software
- new techniques or methods
- can’t decide between alternatives

Not very predictable

- a big problem in contracted developments

Prototyping



Good where there are significant project risks or unknowns

- external software
- new techniques or methods
- can’t decide between alternatives

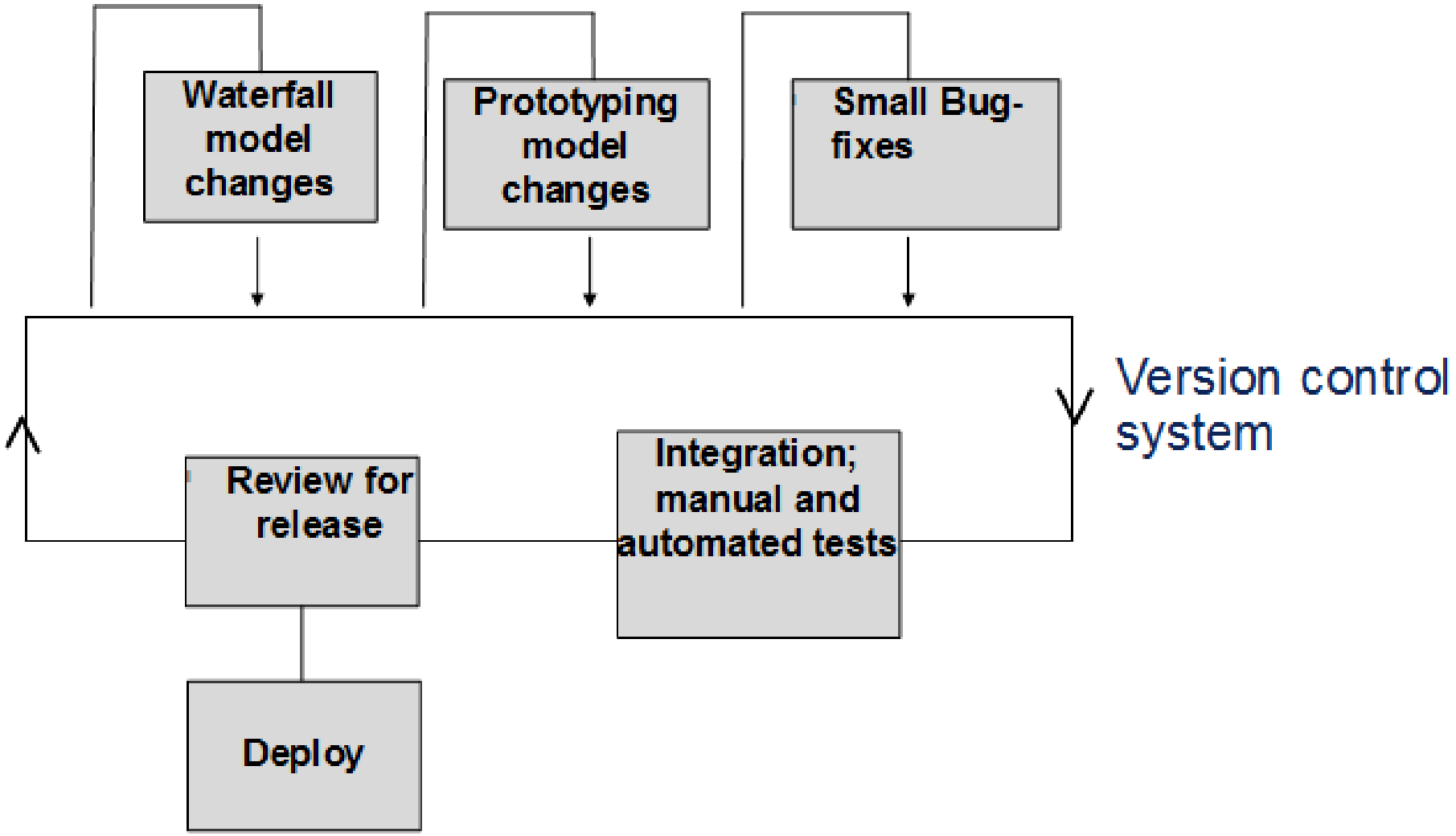
Not very predictable

- a big problem in contracted developments

Extreme programming

- The biggest unknown is the user!
- (less fashionable now)

Evolution

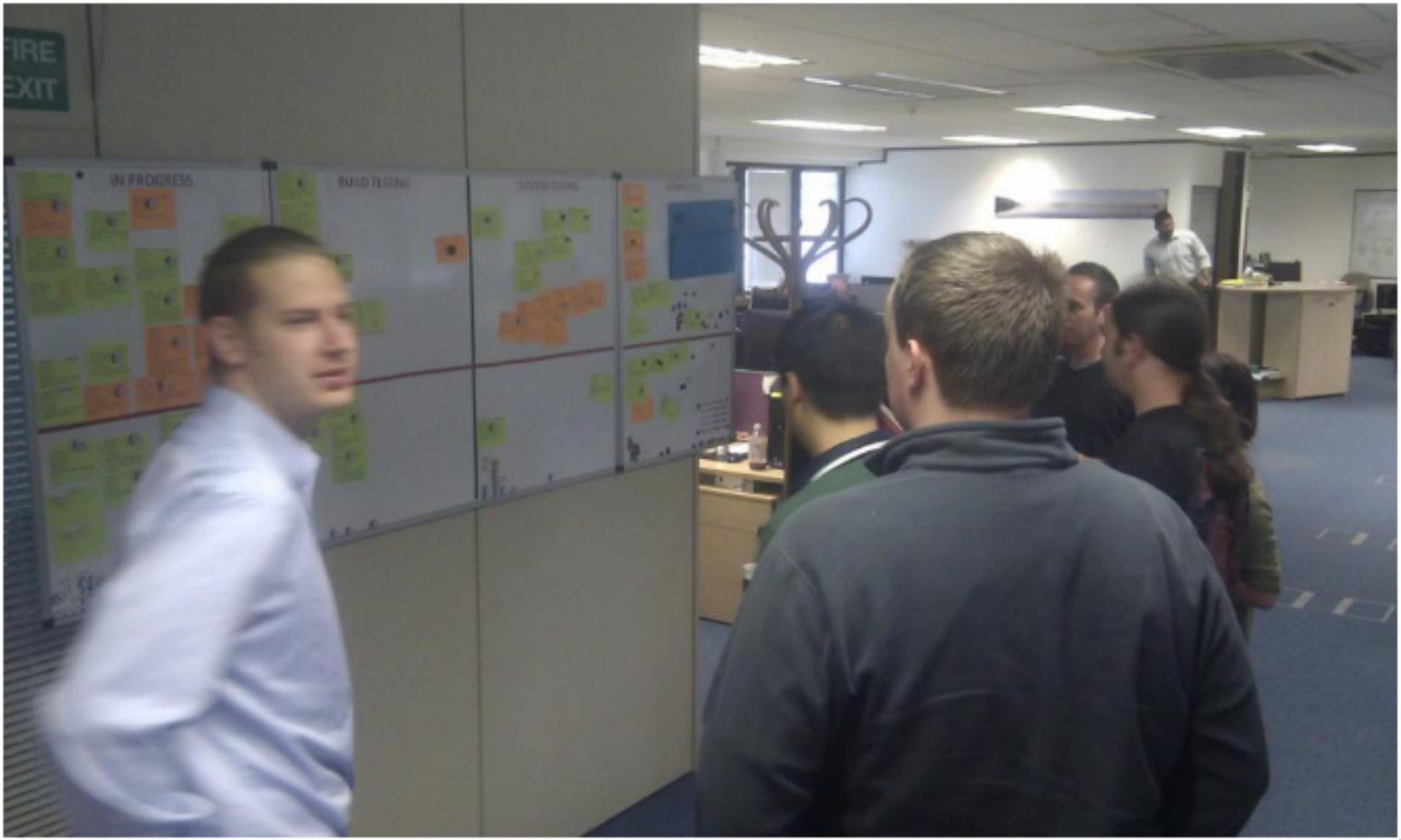


Get a version control system, eg Bitbucket is free

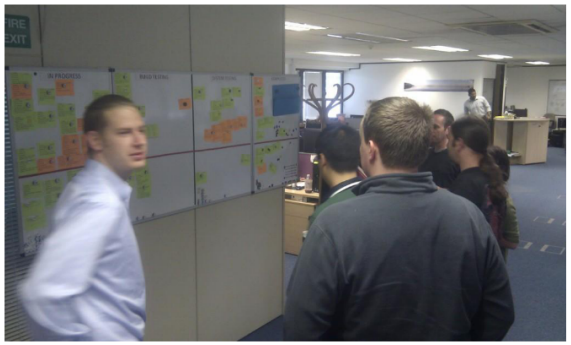
Software development

Agile development

A scrum in Brady plc



Scrum



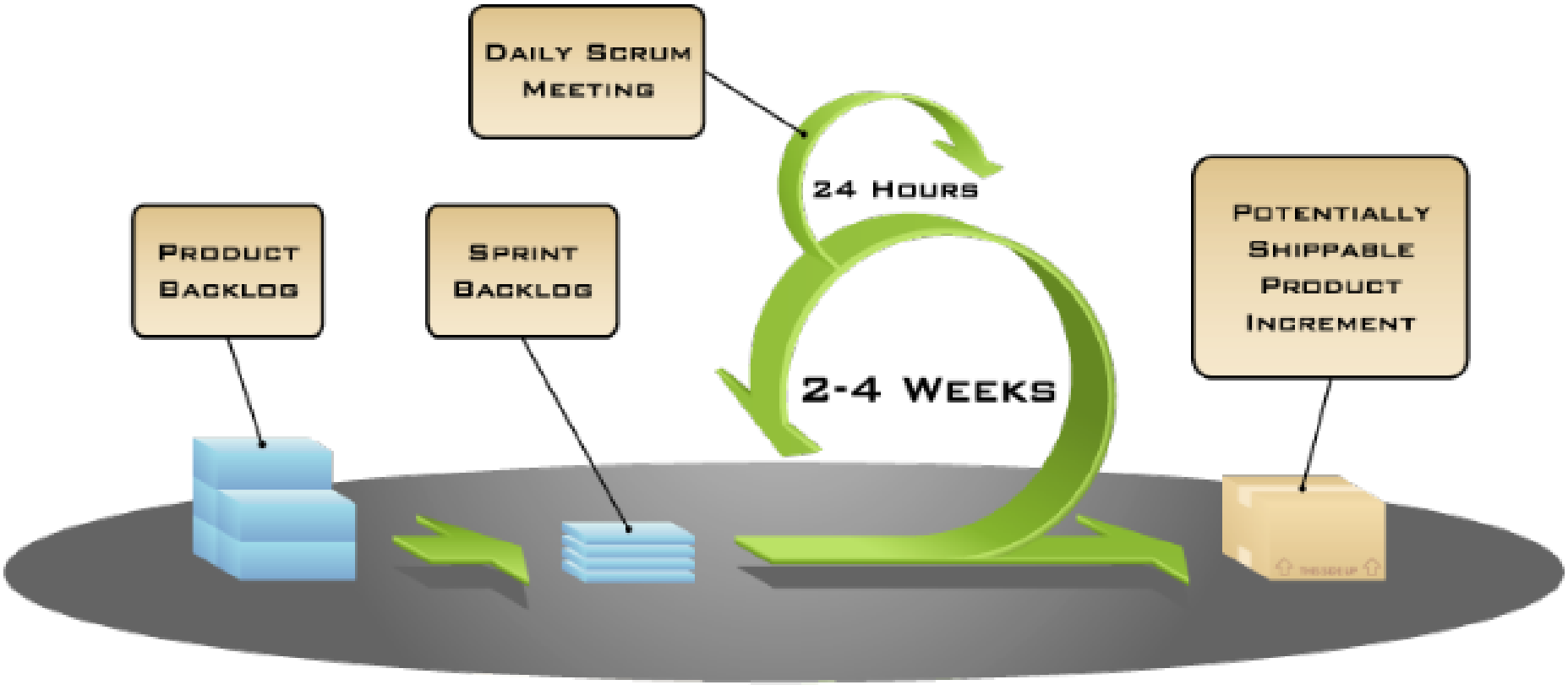
Parameters

- Daily
- 15-minutes
- Stand-up

Not for problem solving

- Whole world is invited
- Only team members, ScrumMaster, product owner, can talk
- Helps avoid other unnecessary meetings

Agile methodology



Atlassian Jira

- Very full software control for this
- With customer bug reporting, feature requests etc.
- Free to use for small projects

Software development

Test-oriented development

Requirements – design – develop – test

- miscommunication throughout the chain
- testing at the end so it suffers most

Requirements - test – design – develop

- test engineer is part of the development team
- Tests run automatically with each daily build

Satisfaction reported

- Remains to be proven in practice
- (I think it is only part of the answer)

Service Oriented Architecture

Use carefully defined interfaces to cut down the size of projects

Service Oriented Architecture

Use carefully defined interfaces to cut down the size of projects

Real-world example

- Two teams - Cambridge and Geneva
- Cambridge have mathematical models
- Geneva have a ‘physical’ system (tracks customer’s metal)
- Customer asked for valuation models

Service Oriented Architecture

Use carefully defined interfaces to cut down the size of projects

Real-world example

- Two teams - Cambridge and Geneva
- Cambridge have mathematical models
- Geneva have a ‘physical’ system (tracks customer’s metal)
- Customer asked for valuation models

Our first major SOA implementation

- C produced a valuation service in late 2011 (cross-platform ‘restful’ service)
- getting worried about lack of feedback from Geneva
- Just went successfully into production

Software development