1. Why bother managing the development process?

2. Software development before 2001

3. Agile software development (from 2001)

4. Building the team

5. Thank you
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
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 – the Hacking 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 (later taught in computer science courses)

- 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 – the Hacking 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 (later taught in computer science courses)
- 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

Complete design up front ...

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

Complete design up front ...

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

Complete design up front ...

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 – 1982
**IBM method based on fundamental misconception**

Complete design up front ...

<table>
<thead>
<tr>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1-1kb</td>
<td>Typical punch-card program</td>
</tr>
<tr>
<td></td>
<td>The IBM development method was probably developed for this type of program</td>
</tr>
<tr>
<td>2kb-10kb</td>
<td>Typical software module/class</td>
</tr>
<tr>
<td>16kb</td>
<td>Typical computer science project</td>
</tr>
<tr>
<td>200kb</td>
<td>Operating system of Sinclair Spectrum – 1982</td>
</tr>
<tr>
<td></td>
<td>Our first software product – 1986</td>
</tr>
</tbody>
</table>
IBM method based on fundamental misconception

Complete design up front ...

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 – 1982

200kb Our first software product – 1986

18 Mb Human Genome – active proteins
**IBM method based on fundamental misconception**

Complete design up front ...

<table>
<thead>
<tr>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1-1kb</td>
<td>Typical punch-card program</td>
</tr>
<tr>
<td></td>
<td>The IBM development method was probably developed for this type of program</td>
</tr>
<tr>
<td>2kb-10kb</td>
<td>Typical software module/class</td>
</tr>
<tr>
<td></td>
<td>Typical computer science project</td>
</tr>
<tr>
<td>16kb</td>
<td>Operating system of Sinclair Spectrum – 1982</td>
</tr>
<tr>
<td>200kb</td>
<td>Our first software product – 1986</td>
</tr>
<tr>
<td>18 Mb</td>
<td>Human Genome – active proteins</td>
</tr>
<tr>
<td>100Mb</td>
<td>Typical software product today</td>
</tr>
</tbody>
</table>
IBM method based on fundamental misconception

Complete design up front ...

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 – 1982

200kb Our first software product – 1986

18 Mb  Human Genome – active proteins

100Mb Typical software product today

750Mb Human genome (3 × 10^9 base-pairs)
IBM method based on fundamental misconception

Complete design up front ...

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 – 1982

200kb  Our first software product – 1986

18 Mb  Human Genome – active proteins

100Mb  Typical software product today

750Mb  Human genome (3 × 10^9 base-pairs)

4Gb  Windows Vista
IBM method based on fundamental misconception

Complete design up front ...

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 – 1982

200kb Our first software product – 1986

18 Mb Human Genome – active proteins

100Mb Typical software product today

750Mb Human genome \(3 \times 10^9\) base-pairs

4Gb Windows Vista

200Gb Storage on my laptop
Robert Brady

**IBM method based on fundamental misconception**

Complete design up front...

<table>
<thead>
<tr>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1-1kb</td>
<td>Typical punch-card program</td>
</tr>
<tr>
<td>2kb-10kb</td>
<td>Typical software module/class</td>
</tr>
<tr>
<td>16kb</td>
<td>Operating system of Sinclair Spectrum – 1982</td>
</tr>
<tr>
<td>200kb</td>
<td>Our first software product – 1986</td>
</tr>
<tr>
<td>18 Mb</td>
<td>Human Genome – active proteins</td>
</tr>
<tr>
<td>100Mb</td>
<td>Typical software product today</td>
</tr>
<tr>
<td>750Mb</td>
<td>Human genome (3 × 10^9 base-pairs)</td>
</tr>
<tr>
<td>4Gb</td>
<td>Windows Vista</td>
</tr>
<tr>
<td>200Gb</td>
<td>Storage on my laptop</td>
</tr>
<tr>
<td>≥Tb</td>
<td>Google code</td>
</tr>
</tbody>
</table>
1. Why bother managing the development process?

2. Software development before 2001

3. Agile software development (from 2001)

4. Building the team

5. Thank you
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
- Miscommunication
Prototyping

“Playcode” it

Review it

Amend or reject it

Test and deploy it

Good where there are significant project risks or unknowns

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

“Playcode” it

Review it

Amend or reject it

Test and deploy it

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
Evolution

- Waterfall model changes
- Prototyping model changes
- Small Bug-fixes

Version control system

- Review for release
- Integration; manual and automated tests
- Deploy
How well did it do? (1)

Success rate of projects (Johnson 1998)

Break up large projects up into shorter ones (weeks not months)
How well did it do? (2)

Actual use of requested features (Johnson 2002)

- Always, 7%
- Often, 13%
- Sometimes, 16%
- Rarely, 19%
- Never, 45%
Most bugs are very hard to find

Number of Bugs (log scale)

Mean time to failure (log scale)

10^5 years  10 days

Most bugs are very hard to find

Number of Bugs (log scale)

Mean time to failure (log scale)

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

Most bugs are very hard to find


<table>
<thead>
<tr>
<th>Period</th>
<th>average</th>
<th>bugs</th>
<th>mttf</th>
</tr>
</thead>
<tbody>
<tr>
<td>10–20d</td>
<td>15d</td>
<td>1</td>
<td>15d</td>
</tr>
<tr>
<td>20–40d</td>
<td>30d</td>
<td>2</td>
<td>15d</td>
</tr>
</tbody>
</table>

Number of Bugs (log scale)

Mean time to failure (log scale)

Most bugs are very hard to find

Number of Bugs (log scale)

Mean time to failure (log scale)

10 days
10
5 years


<table>
<thead>
<tr>
<th>Period</th>
<th>average</th>
<th>bugs</th>
<th>mttf</th>
</tr>
</thead>
<tbody>
<tr>
<td>10–20d</td>
<td>15d</td>
<td>1</td>
<td>15d</td>
</tr>
<tr>
<td>20–40d</td>
<td>30d</td>
<td>2</td>
<td>15d</td>
</tr>
<tr>
<td>40–80d</td>
<td>60d</td>
<td>4</td>
<td>15d</td>
</tr>
</tbody>
</table>
Most bugs are very hard to find

<table>
<thead>
<tr>
<th>Period</th>
<th>average</th>
<th>bugs</th>
<th>mt tf</th>
</tr>
</thead>
<tbody>
<tr>
<td>10–20d</td>
<td>15d</td>
<td>1</td>
<td>15d</td>
</tr>
<tr>
<td>20–40d</td>
<td>30d</td>
<td>2</td>
<td>15d</td>
</tr>
<tr>
<td>40–80d</td>
<td>60d</td>
<td>4</td>
<td>15d</td>
</tr>
<tr>
<td>80–160d</td>
<td>120d</td>
<td>8</td>
<td>15d</td>
</tr>
</tbody>
</table>


10^5 years  10 days

Mean time to failure (log scale)
Most bugs are very hard to find

Number of Bugs (log scale)

Mean time to failure (log scale)

<table>
<thead>
<tr>
<th>Period</th>
<th>average</th>
<th>bugs</th>
<th>mttf</th>
</tr>
</thead>
<tbody>
<tr>
<td>10–20d</td>
<td>15d</td>
<td>1</td>
<td>15d</td>
</tr>
<tr>
<td>20–40d</td>
<td>30d</td>
<td>2</td>
<td>15d</td>
</tr>
<tr>
<td>40–80d</td>
<td>60d</td>
<td>4</td>
<td>15d</td>
</tr>
<tr>
<td>80–160d</td>
<td>120d</td>
<td>8</td>
<td>15d</td>
</tr>
</tbody>
</table>

No hope without automated tests

1. Why bother managing the development process?

2. Software development before 2001

3. Agile software development (from 2001)

4. Building the team

5. Thank you
Short development cycles (2-4 weeks)

- Daily Scrum Meeting
- 24 Hours
- 2-4 Weeks
- Product Backlog
- Sprint Backlog
- Potentially Shippable Product Increment
Short development cycles (2-4 weeks)

- Manage using open source systems (or market leader, Atlassian Jira)
  - Integrated with customer bug reporting, feature requests etc.
Daily scrum meeting

- Daily
- 15-minutes
- Stand-up

Co-ordination, not problem solving

- Whole world is invited
- Only ‘pigs’ may talk
  - team members
  - Scrum master
  - product owner
- Helps avoid other unnecessary meetings
Write tests before the code if possible

Report a bug

Write a test script

Run it with each daily build (it fails!)

Make it work and you are done

Run the tests automatically with each daily build

Find out when someone else breaks your feature

May find regression problems unrelated to your feature

Refactor safely

New features

User stories from the customer

Write the tests based on the user stories (check with user!)

Make it work and you are done

Run the tests automatically with each daily build
Write tests before the code if possible

Report a bug

- Write a test script
- Run it with each daily build (it fails!)
- Make it work and you are done
- Run the tests automatically with each daily build
  - Find out when someone else breaks your feature
  - May find regression problems unrelated to your feature
  - Refactor safely
Write tests before the code if possible

Report a bug

- Write a test script
- Run it with each daily build (it fails!)
- Make it work and you are done
- Run the tests automatically with each daily build
  - Find out when someone else breaks your feature
  - May find regression problems unrelated to your feature
  - Refactor safely

New features

- User stories from the customer
- Write the tests based on the user stories (check with user!)
- Make it work and you are done
- Run the tests automatically with each daily build
1. Why bother managing the development process?

2. Software development before 2001

3. Agile software development (from 2001)

4. Building the team

5. Thank you
Focus on the team, not the individual

Scrum process builds team commitment
- Team members commit at the beginning of a sprint
- They help each other to achieve a shippable product
- Autonomy – management not allowed to interfere during sprint

Developers work with customers
- Key to success and motivation
- Help the team understand what the customer really needs
Getting teams to work together

Try to combine the best features of two products

- Bring two teams together
- Each sees the other as competition
- Personal experience – Two key managers left, with half their teams

What seems to be working

- Start slowly (common log-on screen)
- Cross-platform technology (restful services)
- Promote re-usable code into a common repository with more rigorous controls
- Introduce features from product A into B
- Lubricate with plenty of alcohol
Getting teams to work together

Try to combine the best features of two products

- Bring two teams together
- Each sees the other as competition
- Personal experience – Two key managers left, with half their teams

What seems to be working

- Start slowly (common log-on screen)
- Cross-platform technology (restful services)
- Promote re-usable code into a common repository with more rigorous controls
- Introduce features from product A into B
- Lubricate with plenty of alcohol
1. Why bother managing the development process?

2. Software development before 2001

3. Agile software development (from 2001)

4. Building the team

5. Thank you