LINGI2251 Software engineering

Facts and Fallacies: Management

Florian Thuin
Nicolas Kabasele
Nicolas Houtain
Nathan Magrofuoco

Ecole Polytechnique de Louvain

May 2, 2016

Plan

1. Fallacies 3: People
   - Fact 1
   - Fact 2
   - Fact 3
   - Fact 4
2. Fallacies 6: Estimation
3. Reuse

Fact 1

Fact

The most important factor in software work is *not* the tools and techniques used by the programmers, but rather the quality of the programmers themselves.

- **People matter**.
- We keep forgetting it for tools, techniques & processes.
- Exe : CMM, a good process is the way to good software
- Fixing people issue is way harder...
- ... even inventing new technologies seem easier!

Fact 1

Fact

The most important factor in software work is *not* the tools and techniques used by the programmers, but rather the quality of the programmers themselves.

- **People matter.**
- We keep forgetting it for tools, techniques & processes.
- Exe : CMM, a good process is the way to good software
- Fixing people issue is way harder...
- ... even inventing new technologies seem easier!

Fact 1

Fact

The most important factor in software work is *not* the tools and techniques used by the programmers, but rather the quality of the programmers themselves.

- **People matter**.
- We keep forgetting it for tools, techniques & processes.
- Exe : CMM, a good process is the way to good software
- Fixing people issue is way harder...
- ... even inventing new technologies seem easier!

Fact 1

Fact

The most important factor in software work is *not* the tools and techniques used by the programmers, but rather the quality of the programmers themselves.

- **People matter**.
- We keep forgetting it for tools, techniques & processes.
- Exe : CMM, a good process is the way to good software
- Fixing people issue is way harder...
- ... even inventing new technologies seem easier!

Fact 1

Fact

The most important factor in software work is *not* the tools and techniques used by the programmers, but rather the quality of the programmers themselves.

- **People matter**.
- We keep forgetting it for tools, techniques & processes.
- Exe : CMM, a good process is the way to good software
- Fixing people issue is way harder...
- ... even inventing new technologies seem easier!

Fact 1

Fact

The most important factor in software work is *not* the tools and techniques used by the programmers, but rather the quality of the programmers themselves.

- **People matter**.
- We keep forgetting it for tools, techniques & processes.
- Exe : CMM, a good process is the way to good software
- Fixing people issue is way harder...
- ... even inventing new technologies seem easier!

Fact 2

Fact

The best programmers are up to 28 times better than the worst programmers, according to “individual differences” research. Given that their pay is never commensurate, they are the biggest bargains in the software field.

- **People matter... a lot!**
- Some programmers are 5 to 28 times better than others.
- They are the best bargain in software devlopment.
- But we can’t even identify them.

Fact 2

Fact

The best programmers are up to 28 times better than the worst programmers, according to “individual differences” research. Given that their pay is never commensurate, they are the biggest bargains in the software field.

- **People matter... a lot!**
- Some programmers are 5 to 28 times better than others.
- They are the best bargain in software devlopment.
- But we can’t even identify them.

Fact 2

Fact

The best programmers are up to 28 times better than the worst programmers, according to “individual differences” research. Given that their pay is never commensurate, they are the biggest bargains in the software field.

- **People matter... a lot!**
- Some programmers are 5 to 28 times better than others.
- They are the best bargain in software devlopment.
- But we can’t even identify them.

Fact 2

Fact

The best programmers are up to 28 times better than the worst programmers, according to “individual differences” research. Given that their pay is never commensurate, they are the biggest bargains in the software field.

- **People matter... a lot!**
- Some programmers are 5 to 28 times better than others.
- They are the best bargain in software devlopment.
- But we can’t even identify them.

Fact 2

Fact

The best programmers are up to 28 times better than the worst programmers, according to “individual differences” research. Given that their pay is never commensurate, they are the biggest bargains in the software field.

- **People matter... a lot!**
- Some programmers are 5 to 28 times better than others.
- They are the best bargain in software devlopment.
- But we can’t even identify them.

Fact 3

Fact

Adding people to a late project makes it later.

- More than a fact : **Brook's law** (1995).
- People added later on a project delay it even more :
  - They need time to be productive.
  - They need time to learn team communication.
  - **We** need time to teach them all of these.
- But not always!

Fact 3

Fact

Adding people to a late project makes it later.

- More than a fact : **Brook's law** (1995).
- People added later on a project delay it even more :
  - They need time to be productive.
  - They need time to learn team communication.
  - **We** need time to teach them all of these.
- But not always!

Fact 3

Fact

Adding people to a late project makes it later.

- More than a fact : **Brook's law** (1995).
- People added later on a project delay it even more :
  - They need time to be productive.
  - They need time to learn team communication.
  - **We** need time to teach them all of these.
- But not always!

Fact 3

Fact

Adding people to a late project makes it later.

- More than a fact : **Brook's law** (1995).
- People added later on a project delay it even more :
  - They need time to be productive.
  - They need time to learn team communication.
  - **We** need time to teach them all of these.
- But not always!

Fact 3

Fact

Adding people to a late project makes it later.

- More than a fact : **Brook's law** (1995).
- People added later on a project delay it even more :
  - They need time to be productive.
  - They need time to learn team communication.
  - **We** need time to teach them all of these.
- But not always!

Fact 3

Fact

Adding people to a late project makes it later.

- More than a fact : **Brook's law** (1995).
- People added later on a project delay it even more :
  - They need time to be productive.
  - They need time to learn team communication.
  - **We** need time to teach them all of these.
- But not always!

Fact 3

Fact

Adding people to a late project makes it later.

- More than a fact : **Brook's law** (1995).
- People added later on a project delay it even more :
  - They need time to be productive.
  - They need time to learn team communication.
  - **We** need time to teach them all of these.
- But not always!

Fact 4

Fact

The working environment has a profound impact on productivity and product quality.

- **Environment matter a lot !**
- Should facilitate thinking.
- Distraction and lack of privacy are key enemies to productivity !
- Counter-example : pair programming.

Fact 4

Fact

The working environment has a profound impact on productivity and product quality.

- **Environment matter a lot !**
- Should facilitate thinking.
- Distraction and lack of privacy are key enemies to productivity !
- Counter-example : pair programming.

Fact 4

Fact

The working environment has a profound impact on productivity and product quality.

- **Environment matter a lot !**
- Should facilitate thinking.
- Distraction and lack of privacy are key enemies to productivity !
- Counter-example : pair programming.

Fact 4

Fact

The working environment has a profound impact on productivity and product quality.

- **Environment matter a lot !**
- Should facilitate thinking.
- Distraction and lack of privacy are key enemies to productivity !
- Counter-example : pair programming.

Fact 4

Fact

The working environment has a profound impact on productivity and product quality.

- **Environment matter a lot !**
- Should facilitate thinking.
- Distraction and lack of privacy are key enemies to productivity !
- Counter-example : pair programming.

Plan

Fact 8

Fact

One of the two most common causes of runaway projects is poor estimation.

What are runaway projects?

- Projects that are out of control;
- Projects that are behind schedules;
- Projects that are over budget.

Fact 8

Fact

One of the two most common causes of runaway projects is poor estimation.

What are runaway projects?

- Projects that are out of control;
- Projects that are behind schedules;
- Projects that are over budget.

Fact 8

Fact

One of the two most common causes of runaway projects is poor estimation.

What are runaway projects?

- Projects that are out of control;
- Projects that are behind schedules;
- Projects that are over budget.

Fact 8

Fact

One of the two most common causes of runaway projects is poor estimation.

What are runaway projects?

- Projects that are out of control;
- Projects that are behind schedules;
- Projects that are over budget.

Fact 8

Fact

One of the two most common causes of runaway projects is poor estimation.

How *developers* explain the runaway projects?

- We use bad tools!
- We use bad methodologies!
- We lack of discipline and rigor!

What are the *real reasons*?

- We did poor estimations (i.e. we were too optimistic);
- We had unstable requirements

Fact 8

Fact

One of the two most common causes of runaway projects is poor estimation.

How *developers* explain the runaway projects?

- We use bad tools!
- We use bad methodologies!
- We lack of discipline and rigor!

What are the *real reasons*?

- We did poor estimations (i.e. we were too optimistic);
- We had unstable requirements

Fact 8

Fact

One of the two most common causes of runaway projects is poor estimation.

How *developers* explain the runaway projects?

- We use bad tools!
- We use bad methodologies!
- We lack of discipline and rigor!

What are the *real reasons*?

- We did poor estimations (i.e. we were too optimistic);
- We had unstable requirements

Fact 8

Fact

One of the two most common causes of runaway projects is poor estimation.

How *developers* explain the runaway projects?

- We use bad tools!
- We use bad methodologies!
- We lack of discipline and rigor!

What are the *real reasons*?

- We did poor estimations (i.e. we were too optimistic);
- We had unstable requirements

Fact 8

Fact

One of the two most common causes of runaway projects is poor estimation.

How *developers* explain the runaway projects?

- We use bad tools!
- We use bad methodologies!
- We lack of discipline and rigor!

What are the *real reasons*?

- We did poor estimations (i.e. we were too optimistic);
- We had unstable requirements

Fact 8

Fact

One of the two most common causes of runaway projects is poor estimation.

How *developers* explain the runaway projects?

- We use bad tools!
- We use bad methodologies!
- We lack of discipline and rigor!

What are the *real reasons*?

- We did poor estimations (i.e. we were too optimistic);
- We had unstable requirements

Fact 8

Fact

One of the two most common causes of runaway projects is poor estimation.

How *developers* explain the runaway projects?

- We use bad tools!
- We use bad methodologies!
- We lack of discipline and rigor!

What are the *real reasons*?

- We did poor estimations (i.e. we were too optimistic);
- We had unstable requirements

Fact 8

Fact

One of the two most common causes of runaway projects is poor estimation.

Why are we so bad at estimates?

- We believe in “expert” people (“I’ve been there and done that!”);
- We believe in algorithmic approach;
- We believe in the lines of code approach;

Are there better approaches?

- The function point (FP) approach: estimates based on the number of input and output;
- The features point approach: estimates based on the number of features to be developed;
- The human-mediated estimation process: mix of the opinion of an expert and an algorithm that produces reasonably good answers for this family of softwares.

Fact 8

Fact

One of the two most common causes of runaway projects is poor estimation.

Why are we so bad at estimates?

- We believe in “expert” people (“I’ve been there and done that!”);
- We believe in algorithmic approach;
- We believe in the lines of code approach;

Are there better approaches?

- The function point (FP) approach: estimates based on the number of input and output;
- The features point approach: estimates based on the number of features to be developed;
- The human-mediated estimation process: mix of the opinion of an expert and an algorithm that produces reasonably good answers for this family of softwares.

Fact 8

Fact

One of the two most common causes of runaway projects is poor estimation.

Why are we so bad at estimates?

- We believe in "expert" people ("I've been there and done that!");
- We believe in algorithmic approach;
- We believe in the lines of code approach;

Are there better approaches?

- The function point (FP) approach: estimates based on the number of input and output;
- The features point approach: estimates based on the number of features to be developed;
- The human-mediated estimation process: mix of the opinion of an expert and an algorithm that produces reasonably good answers for this family of softwares.

Fact 8

Fact

One of the two most common causes of runaway projects is poor estimation.

Why are we so bad at estimates?

- We believe in “expert” people (“I’ve been there and done that!”);
- We believe in algorithmic approach;
- We believe in the lines of code approach;

Are there better approaches?

- The function point (FP) approach: estimates based on the number of input and output;
- The features point approach: estimates based on the number of features to be developed;
- The human-mediated estimation process: mix of the opinion of an expert and an algorithm that produces reasonably good answers for this family of softwares.

Fact 8

Fact

One of the two most common causes of runaway projects is poor estimation.

Why are we so bad at estimates?

- We believe in “expert” people (“I’ve been there and done that!”);
- We believe in algorithmic approach;
- We believe in the lines of code approach;

Are there better approaches?

- The function point (FP) approach: estimates based on the number of input and output;
- The features point approach: estimates based on the number of features to be developed;
- The human-mediated estimation process: mix of the opinion of an expert and an algorithm that produces reasonably good answers for this family of softwares.

Fact 8

Fact

One of the two most common causes of runaway projects is poor estimation.

Why are we so bad at estimates?

- We believe in “expert” people (“I’ve been there and done that!”);
- We believe in algorithmic approach;
- We believe in the lines of code approach;

Are there better approaches?

- The function point (FP) approach: estimates based on the number of input and output;
- The features point approach: estimates based on the number of features to be developed;
- The human-mediated estimation process: mix of the opinion of an expert and an algorithm that produces reasonably good answers for this family of softwares.

Fact 8

Fact

One of the two most common causes of runaway projects is poor estimation.

Why are we so bad at estimates?

- We believe in “expert” people (“I’ve been there and done that!”);
- We believe in algorithmic approach;
- We believe in the lines of code approach;

Are there better approaches?

- The function point (FP) approach: estimates based on the number of input and output;
- The features point approach: estimates based on the number of features to be developed;
- The human-mediated estimation process: mix of the opinion of an expert and an algorithm that produces reasonably good answers for this family of softwares.

Fact 9

Fact

Most software estimates are performed at the beginning of the life cycle. This makes sense until we realize that estimates are obtained before the requirements are defined and thus before the problem is understood. Estimation, therefore, usually occurs at the wrong time.

When an software company meets a client, the client wants to know when he will get the product. That forces software company to give an estimation before starting the work (and so, before starting the requirements definition and analysis).

Fact 9

Fact

Most software estimates are performed at the beginning of the life cycle. This makes sense until we realize that estimates are obtained before the requirements are defined and thus before the problem is understood. Estimation, therefore, usually occurs at the wrong time.

When an software company meets a client, the client wants to know when he will get the product. That forces software company to give an estimation before starting the work (and so, before starting the requirements definition and analysis).

Fact 10

Fact

Most software estimates are made either by upper management or by marketing, not by the people who will build the software or their managers. Estimation is, therefore, done by the wrong people.

When estimates are made by the upper management or marketing, they are more related to *wishes* than to reality.

One big problem is that there is two points of view:

The political point-of-view : The contract specifies that the release must happen on a certain date.

The rational point-of-view : It is not possible to develop those features with quality requirements at such cost with this schedule.

Fact 10

Fact

Most software estimates are made either by upper management or by marketing, not by the people who will build the software or their managers. Estimation is, therefore, done by the wrong people.

When estimates are made by the upper management or marketing, they are more related to *wishes* than to reality.

One big problem is that there is two points of view:

The political point-of-view
: The contract specifies that the release must happen on a certain date.

The rational point-of-view
: It is not possible to develop those features with quality requirements at such cost with this schedule.

Fact 10

Fact

Most software estimates are made either by upper management or by marketing, not by the people who will build the software or their managers. Estimation is, therefore, done by the wrong people.

When estimates are made by the upper management or marketing, they are more related to *wishes* than to reality.

One big problem is that there is two points of view:

The political point-of-view
: The contract specifies that the release must happen on a certain date.

The rational point-of-view
: It is not possible to develop those features with quality requirements at such cost with this schedule.

Fact 10

Fact

Most software estimates are made either by upper management or by marketing, not by the people who will build the software or their managers. Estimation is, therefore, done by the wrong people.

When estimates are made by the upper management or marketing, they are more related to *wishes* than to reality.

One big problem is that there is two points of view:

The political point-of-view
: The contract specifies that the release must happen on a certain date.

The rational point-of-view
: It is not possible to develop those features with quality requirements at such cost with this schedule.

Fact 11

Fact

Software estimates are rarely adjusted as the project proceeds. Thus those estimates done at the wrong time by the wrong people are usually not corrected.

Even if everyone in the developer team knows the schedule will not be respected, nobody will try to create a realistic schedule.

Fact 11

Fact

Software estimates are rarely adjusted as the project proceeds. Thus those estimates done at the wrong time by the wrong people are usually not corrected.

Even if everyone in the developer team knows the schedule will not be respected, nobody will try to create a realistic schedule.

Fact 12

Fact

Since estimates are so faulty, there is little reason to be concerned when software projects do not meet estimated targets. But everyone is concerned anyway.

Everyone knows the estimations were bad. But people still give credits to them because it's not cool to be late.

Maybe it's our vision of the quality of a project that has to be reviewed. Is a good project only a project following its schedule?

Fact 12

Fact

Since estimates are so faulty, there is little reason to be concerned when software projects do not meet estimated targets. But everyone is concerned anyway.

Everyone knows the estimations were bad. But people still give credits to them because it's not cool to be late.

Maybe it's our vision of the quality of a project that has to be reviewed. Is a good project only a project following its schedule?

Fact 12

Fact

Since estimates are so faulty, there is little reason to be concerned when software projects do not meet estimated targets. But everyone is concerned anyway.

Everyone knows the estimations were bad. But people still give credits to them because it's not cool to be late.

Maybe it's our vision of the quality of a project that has to be reviewed. Is a good project only a project following its schedule?

Fact 12

Fact

Since estimates are so faulty, there is little reason to be concerned when software projects do not meet estimated targets. But everyone is concerned anyway.

What if we managed

- by product (is the product available and working?);
- by issue (are the issue well and quickly resolved?);
- by risk (are the identified risks overcome?);
- by business objectives (did the business performance improve?);
- by quality (number of quality attributes reached?).

Fact 12

Fact

Since estimates are so faulty, there is little reason to be concerned when software projects do not meet estimated targets. But everyone is concerned anyway.

What if we managed

- by product (is the product available and working?);
- by issue (are the issue well and quickly resolved?);
- by risk (are the identified risks overcome?);
- by business objectives (did the business performance improve?);
- by quality (number of quality attributes reached?).

Fact 12

Fact

Since estimates are so faulty, there is little reason to be concerned when software projects do not meet estimated targets. But everyone is concerned anyway.

What if we managed

- by product (is the product available and working?);
- by issue (are the issue well and quickly resolved?);
- by risk (are the identified risks overcome?);
- by business objectives (did the business performance improve?);
- by quality (number of quality attributes reached?).

Fact 12

Fact

Since estimates are so faulty, there is little reason to be concerned when software projects do not meet estimated targets. But everyone is concerned anyway.

What if we managed

- by product (is the product available and working?);
- by issue (are the issue well and quickly resolved?);
- by risk (are the identified risks overcome?);
- by business objectives (did the business performance improve?);
- by quality (number of quality attributes reached?).

Fact 12

Fact

Since estimates are so faulty, there is little reason to be concerned when software projects do not meet estimated targets. But everyone is concerned anyway.

What if we managed

- by product (is the product available and working?);
- by issue (are the issue well and quickly resolved?);
- by risk (are the identified risks overcome?);
- by business objectives (did the business performance improve?);
- by quality (number of quality attributes reached?).

Fact 12

Fact

Since estimates are so faulty, there is little reason to be concerned when software projects do not meet estimated targets. But everyone is concerned anyway.

What if we managed

- by product (is the product available and working?);
- by issue (are the issue well and quickly resolved?);
- by risk (are the identified risks overcome?);
- by business objectives (did the business performance improve?);
- by quality (number of quality attributes reached?).

Fact 12

Fact

Since estimates are so faulty, there is little reason to be concerned when software projects do not meet estimated targets. But everyone is concerned anyway.

eXtreme Programming is another approach, the customer can choose 3 out of the 4 factors:

- cost;
- schedule;
- features;
- quality.

Fact 12

Fact

Since estimates are so faulty, there is little reason to be concerned when software projects do not meet estimated targets. But everyone is concerned anyway.

eXtreme Programming is another approach, the customer can choose 3 out of the 4 factors:

- cost;
- schedule;
- features;
- quality.

Fact 12

Fact

Since estimates are so faulty, there is little reason to be concerned when software projects do not meet estimated targets. But everyone is concerned anyway.

eXtreme Programming is another approach, the customer can choose 3 out of the 4 factors:

- cost;
- schedule;
- features;
- quality.

Fact 12

Fact

Since estimates are so faulty, there is little reason to be concerned when software projects do not meet estimated targets. But everyone is concerned anyway.

eXtreme Programming is another approach, the customer can choose 3 out of the 4 factors:

- cost;
- schedule;
- features;
- quality.

Fact 13

Fact

There is a disconnect between management and their programmers. In one research study of a project that failed to meet its estimates and was seen by its management as a failure, the technical participants saw it as the most successful project they had ever worked on.

Why management consider it as a failure?

- 27 months instead of 14.
- 130% for the sotfware.
- 800% for the firmware.

Fact 13

Why programmer consider it a success?

- Final product was complete.
- Developing was a technical challenge.
- Freedom given by the management.

The main problem is to define what's constitute a succesful project.

Fact 14

Fact

The answer to a feasibility study is almost always “yes.”

- Developer tends to be too optimistic.
- Illusion that everything will go without problem.
- Time frame between the answer and discovery of its falseness too long.

Plan

Fact 15

Fact

Reuse-in-the-small (libraries of subroutines) began nearly 50 years ago and is a well-solved problem.

- First libraries of subroutines in the mid-1950s.
- Math functions, String handlers,sorts,. . .
- Succesful idea which try to be expanded.

Fact 16

Fact

Reuse-in-the-large (components) remains a mostly unsolved problem, even though everyone agrees it is important and desirable.

- More difficult to build a large reusable component than a small one.
- Two points of view:
  - Futur of the field, where program are an aggregate of components.
  - Nearly possible to generalize enough.

Fact 16

- It depend on sofware diversity, is there enough common problem across project?
- *Not-Invented-Here* phenomenon
- A lack of skill?

Fact 17

Fact

Reuse-in-the-large works best in families of related systems and thus is domain-dependent. This narrows the potential applicability of reuse-in-the-large.

- Reuse-in-the-large is **difficult**

because need component connection across application domain
but within a specific domain it's easy

- *One-size-fits-all* tools is always a believe of latter people
  $\Rightarrow$ They are wrong to thinks that construction of software are the same for all domain

Fact 18

Fact

There are two “rules of three” in reuse: (a) It is three times as difficult to build reusable components as single use components, and (b) a reusable component should be tried out in three different applications before it will be sufficiently general to accept into a reuse library.

$\Rightarrow$ Couple of rules of thumb

(a) Reusable component (RC) need:

1. Find the "general problem"
2. Build the RC such at the general and specific is solved by the same way
3. Find a "general testing approach"

$\rightarrow$ 3 come from knowledgeable reuse expert

(b) arbitrary (magical?) number which is more "at least" three times

$\Rightarrow$ implies minimal reusability

Fact 18

Fact

There are two “rules of three” in reuse: (a) It is three times as difficult to build reusable components as single use components, and (b) a reusable component should be tried out in three different applications before it will be sufficiently general to accept into a reuse library.

- Everyone acknowledge that RC take more time: more think, more verification.
- Number three is accepted:

because it’s a rules of thumb

anyone has the conviction that it’s the exact number and that it MUST NEVER CHANGE

Fact 19

Fact

Modification of reused code is particularly error-prone. If more than 20 to 25 percent of a component is to be revised, it is more efficient and effective to rewrite it from scratch.

Hard to have reuse-in-the-large: why not modify component?

1. Software design = framework + philosophy
   $\rightarrow$ modification = understand framework and accepts philosophy
2. Design envelop constraint problem that not fit in the envelope
3. Need to "comprehending the existing solution"
   $\rightarrow$ programmer who originally built the solution may find it difficult to modify some months later.

$\Rightarrow$ Documentation is the solution to these problem but:

- Documentation is for almost NIL
- If there is documentation he is not update on modification

$\rightarrow$ No maintenance documentation at the end

Fact 19

Fact

Modification of reused code is particularly error-prone. If more than 20 to 25 percent of a component is to be revised, it is more efficient and effective to rewrite it from scratch.

Easy to accept if accept *"software products are difficult to build and maintain"*

$\Rightarrow$ People who don't accept this have

- Never see complicated software solution
- As play with toy problems only
- Only exposed to software through literacy course with complicated software such as "Hello world"

For those people, let them do shit.

Fact 19

Fact

Modification of reused code is particularly error-prone. If more than 20 to 25 percent of a component is to be revised, it is more efficient and effective to rewrite it from scratch.

Corollary with *"It is almost always a mistake to modify packaged, vendor-produced software systems."*

- Many different release (added functionality, solve problem)

  IF Approach review THEN old modification remake
  SO Financial cost + moral cost

Fact 20

Fact

Design pattern reuse is one solution to the problems inherent in code reuse.

Reusing code Vs Reusing design = **design pattern**

- It's a description of a problem and a solution (When apply and the consequences)
- More conceptual and abstract than code itself

Note : emerge from practice not theory

Adopted by practitioners AND academics but difficult to measure the impact of this new work on practice.
$\Rightarrow$ Design patterns represent one of the most unequivocally satisfying, least forgotten, truths of the software field.