Requirements Determination is Unstoppable: An Experience Report

Daniel M. Berry, CS;
Krzysztof Czarnecki, Michał Antkiewicz,
Mohamed AbdElRazik, ECE;
University of Waterloo

Some Terminology

Requirements analysis (RA) is known also as "requirements engineering (RE)".

Requirements specification (RS) is what RE yields.

Requirements determination (RD) may or may not be done as part of RE.

Requirements Determination

RD is the making of some requirements relevant decision …

perhaps as small as deciding one word in a RS.

RD may or may not be part of any conscious RE process.

Consulting at Company X

X has a well-managed IT department.

X's IT department has produced award-winning applications.

Nevertheless an X VP asked us to look at their RE process for problems.

How We Did the Consulting

Semi-structured interviews with 18 people:

- **5 focus groups**
- **23 hours of recordings capturing about 40 people's remarks**
- **logged hundreds of quotations**

Clustering the Quotations

While clustering the quotations, we noticed that they told a story, …

a story that some of us had seen before.

A Model of the Lifecycle

From the story, we came up with a model of X's software lifecycle using long-understood ideas from the RE field.

The model explains about 95% of what we heard.

Outline of the Rest of Talk

- **Paraphrases of some quotations**
- **The model, in three parts, Model I, Model II, and Model III**
- **Conclusions and implications of the model**
- **What happened at presentation of the model at X**

Paraphrases of some quotations

Not enough time for RE.

RE is timeboxed.

Coding starts too early.

Coding is done to early requirements.

Results in many project change notices (PCNs).

Stealth changes with
no PCN to avoid
reproach a PCN earns.

Testing effort estimated based on very early requirements.

It seems that ...

RE is being stopped before it has run its course.

Reality

Michael Jackson [1995] once said:

“Requirements engineering is where the informal meets the formal.”

- **Raw ideas: informal**
- **Code: formal**

→ Model I

Informal Meets Formal (Model I)

Two Extremes:

- ***Upfront RE*, in which as much time as necessary is spent to determine requirements before proceeding with design and implementation.**

- ***RD during coding*, in which the programmers and testers determine all requirements as they write the code and test cases.**

Informal Meets Formal (Model I), Cont'd

In Most Projects…

the meeting point is somewhere in the middle.

Meeting Point is Unavoidable

There is no way to go from ideas to code without determining requirements for the code from the ideas.

That is, no programmer can write code without knowing what the code is to do, even if he or she has to decide what the code is to do on the spot.

When upfront RE cut short, ...

the RS is incomplete.

When programmers receive an incomplete RS, ...

they cannot continue until they decide what the missing requirements are.

How programmers should decide missing requirements?

They should ask the client.

When programmers ask the client, …

delay

Sadly, …

Often, the programmer does not ask the client:

- **cannot find client, or**
- **has no access to client**

So, …

the programmer invents requirements on the spot.

(It's called "creativity" or "initiative"! ☺)

When programmer invents, …

It's not good, because:

- **Programmers are not trained in RE.**
- **Programmers have interests that are different from the client's, to simply their own coding.**
- **Each programmer needing a missing requirement is working independently.**

Throw in Testers

Add to all this that the testers are trying to write test cases for incomplete requirements.

Ergo, even more independent invention of requirements.

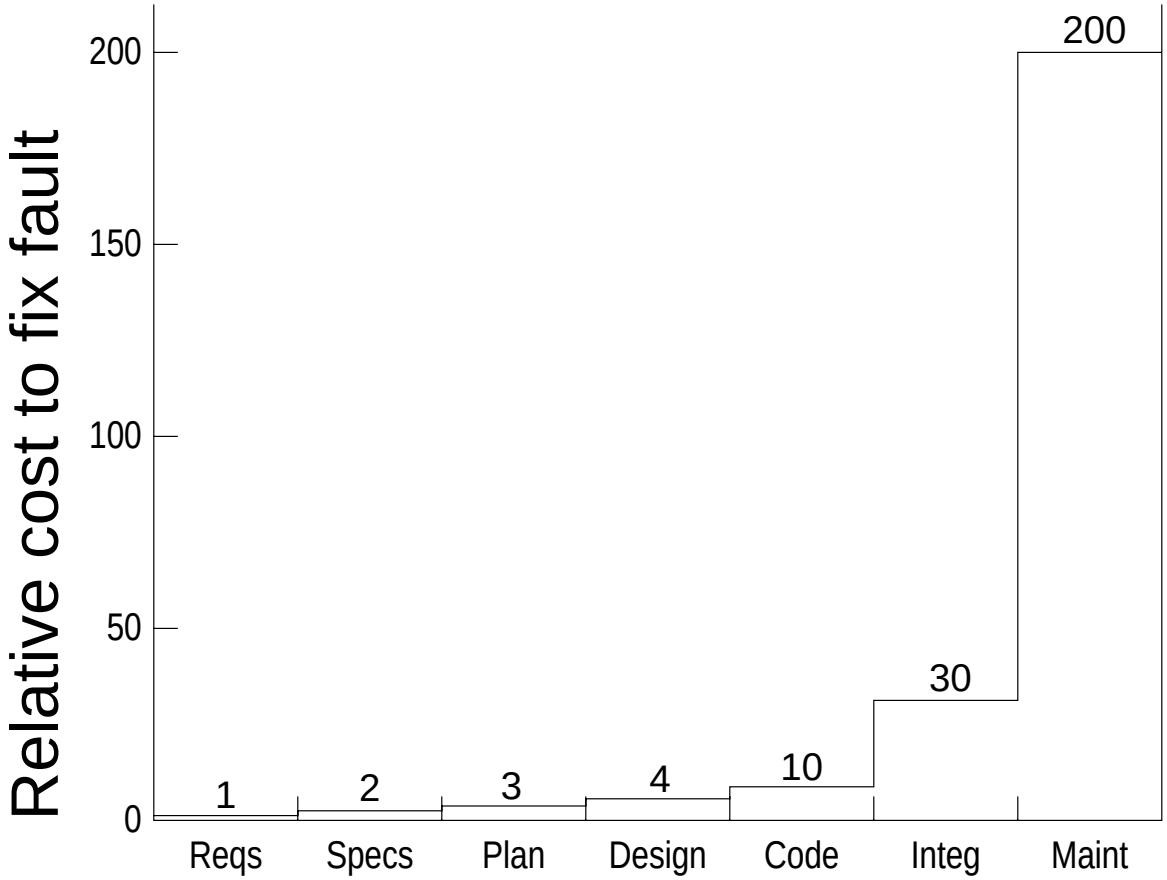
Programmer- and Tester-Determined Requirements

They are bad…. and

They are expensive.

→ Model II

Why Expensive? (Model II)



Perceptions

How do people perceive any new requirements determined after delivery of the RS?

Creep!

even though the new requirements may be what was missing because of terminated RE.

Logical Conclusion for Models I & II

RD always continues until it answers all questions any programmer has about writing the code and any tester has about writing test cases.

- **There is no escaping this reality.**
- **It's like death and taxes!**

Logical Conclusion for Models I & II, Cont'd

The client must be accessible for the entire duration of RE.

We are talking about the *actual* duration of RE, not the official duration, …

especially if the actual duration of RE is the full lifecycle.

Logical Conclusion for Models I & II, Cont'd

The client must be accessible for the entire duration of …

***RD*.**

But, But, But ...

If we don't stop RE,
it will go on forever!

Like a mother's work, RE is never done!

Yes and No

Yes:

There are *always* new requirements for software that is being used [Lehman], …

and iterative methods are for dealing with those kinds of new requirements.

Yes and No, Cont'd

No:

Once a scope is picked — and you cannot complete the code without pinning down *some* scope — there are no *new* requirements, only *as yet undiscovered* requirements.

How to Know if RE is Done-1:

RE *for a scope* is done when the RS is complete enough that every …

programmer can program the required code …

without having to ask anyone to clarify a requirement and without having to invent any requirements on the spot.

Every good programmer knows such an RS instinctively.

How to Know if RE is Done-2:

RE *for a scope* is done when the RS is complete enough that every …

tester can write the required test cases …

without having to ask anyone to clarify a requirement and without having to invent any requirements on the spot.

Every good tester
knows such an RS instinctively.

Another Conclusion for Models I & II

At least one programmer and one tester should be part of the RS writing team …

in order to help the team determine when to continue and when to stop.

BEGIN SKIP FOR CONFERENCE:

Model I Applies to Iterative and Agile Lifecycles

The full line is repeated for each iteration.

Each iteration serves as part of the RE for the next iteration.

Model I Applies to Iterative and Agile Lifecycles, Cont'd

In an agile lifecycle,

- **the scope is smaller and**
- **the client is available all the time.**

So it's OK that the programmer is doing RD as he or she writes code.

END SKIP FOR CONFERENCE:

Not In Paper

There was no room for the following in the paper.

However, it is the third model.

If time permits, I will cover it!

If not, I will only mention it and move to the conclusions.

Problem of the Lack of Benefit of a Document to its Producer (PotLoBoaDtiP)

This problem plagues all the documents that people just hate to write or to keep up to date.

PotLoBoaDtiP was first identified by Paul Arkley & Steve Riddle as the *traceability benefit problem*.

But, the problem exists for more than creating and maintaining traces.

PotLoBoaDtiP (Model III)

PotLoBoaDtiP occurs whenever those who have the knowledge to produce a document are not the ones who benefit from the document, and…

those who benefit from the document, its consumers, do not have the knowledge to easily and quickly produce the document when they need it.

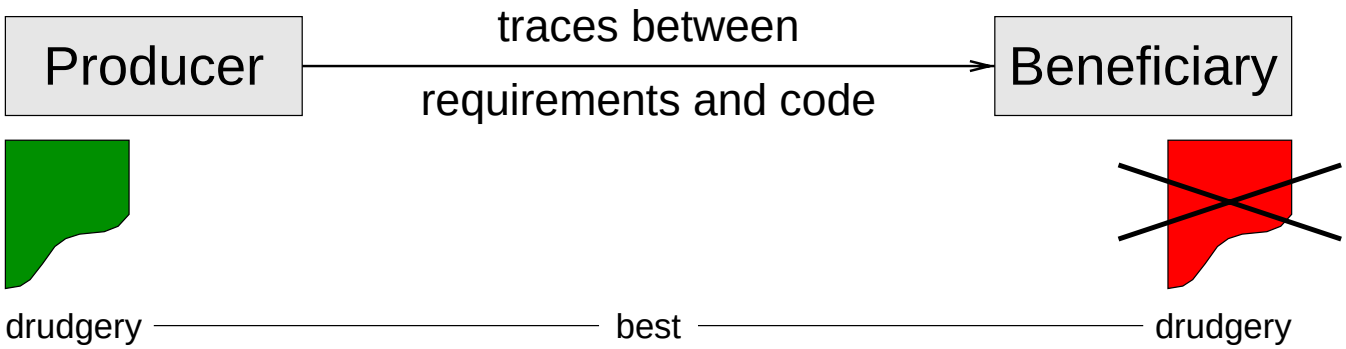
PotLoBoaDtiP

At best, for doing the document, the producer suffers drudgery; at worst, the producer suffers rebuke.

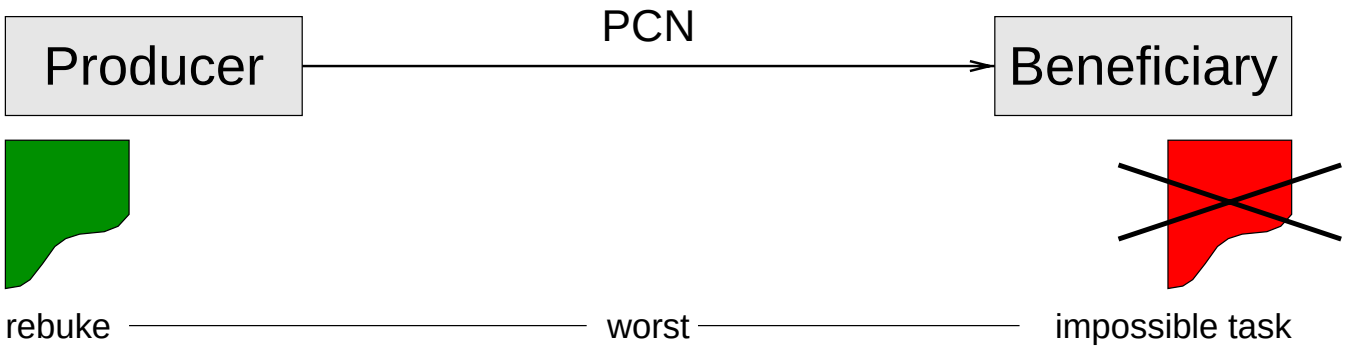
At best, for not having the document, the beneficiary suffers drudgery; at worst, the beneficiary suffers an impossible task.

Consequences of PotLoBoaDtiP

Example 1 of PotLoBoaDtiP



Example 2 of PotLoBoaDtiP



Technology Does Not Solve PotLoBoaDtiP

There are lots of tools out there for tracing.

But, just as there is no incentive to produce the trace, there is no incentive to use the tools.

PotLoBoaDtiP must be addressed as an incentive problem.

Conclusions

- **RD continues until all programmers' and testers' questions are answered.**
- **Client must be accessible throughout actual RE.**
- **Programmers and testers should help determine when RE is done.**
- **PotLoBoaDtiP should be addressed via incentives.**

Presentation to X's VPs

X's VPs expected boring presentation about quotations and their frequencies and limited our presentation to 15 minutes.

We surprised them by presenting only a summary of the quotations and then focused on the model and its conclusions.

We believe that …

focusing on the model was the right thing to do…

because of the lively ½-hour discussion that ensued.

One VP said that we hit the nail on the head!

Now Go Read Our Paper!

But please be polite to the other authors of this session, and stay until the end of this session.