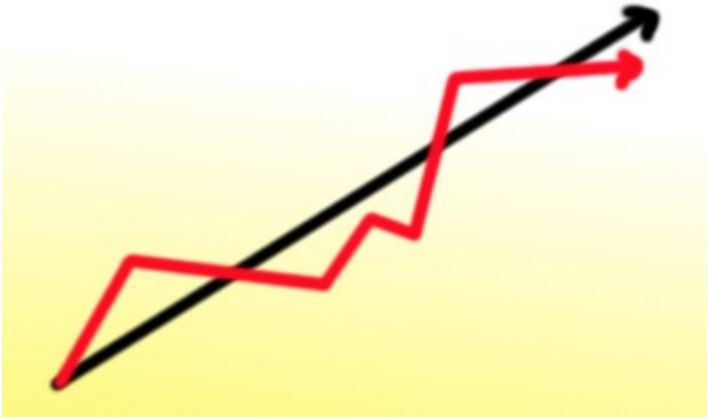
ZIG-ZAG DEVELOPMENT

The more you plan and the more you do analysis, the more your development will be linear. If you don't plan or you don't do any kind of analysis upfront, you have to constantly correct your direction. That can look like a zig-zag approach and cost more time and effort.



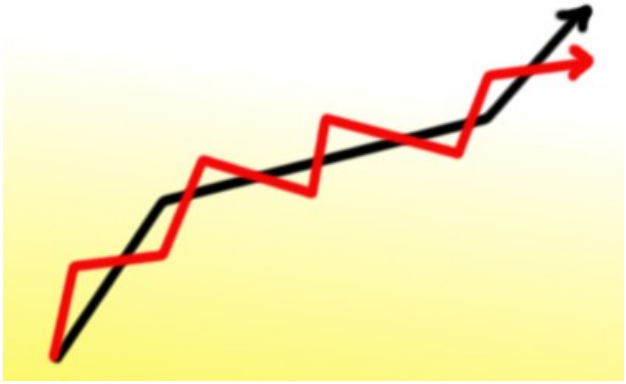
The black line is your shortest path, and the red line your actual path. Obviously you cannot plan everything and there is always a certain amount of uncertainty or unknown, but you should plan far enough to avoid rework.

At the beginning of the project you need to set a frame, then you need further analysis as you get more information, before the development of each part. It's like a writer that frames his novel upfront, then focuses on each chapter before writing it. He does that far enough to avoid rework.

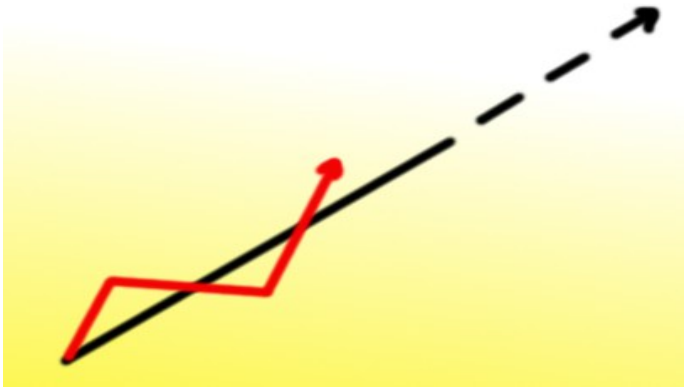
Requirement changes are a completely different story: they don't alter the red line, they alter the black line. So, they require replanning and further analysis.

Every requirement change has a cost and extend

development time by bending the black line.



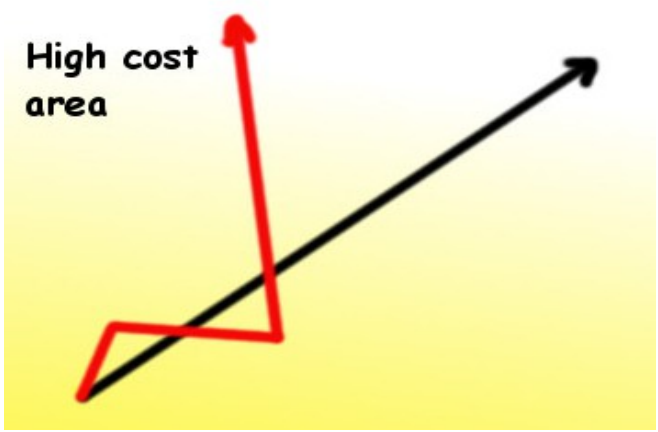
The introduction of a new requirement, not necessarily behave like a requirement change, by bending the black line. It often extends it:



Every requirement change (black line) or correction of direction (red line) will be less expensive, if we follow good practices, as the code is easier to change.
The lack of analysis is not the only reason the red line can drift away from the black line. There are three major causes:

- Lack of requirements
- Lack of analysis
- Wrong implementation or bugs

The more the two lines divert, the higher the costs.



This is also related to time. A wrong implementation, discovered after a long period of time, can be the foundation of other implementations. An imperfect analysis discovered after a long time, and translated into code, can have huge costs.

You can avoid problems from going too far, by getting feedbacks and testing frequently. You can avoid zig-zag development, by gathering all the requirements, asking questions, and compiling a proper design. One of the reason "mental design" doesn't work is that you need to simulate how the software works on paper, to really understand the dynamics and find out the obstacles and the problems ahead. Only after you visualize your schema on paper, you can realize it doesn't make sense at all. Only after you visualize your user interface on paper, you realize it doesn't make sense at all. Realizing this at that stage, will save you a lot of work. Never forget that a good analysis is your best ally to prevent zig-zag development.