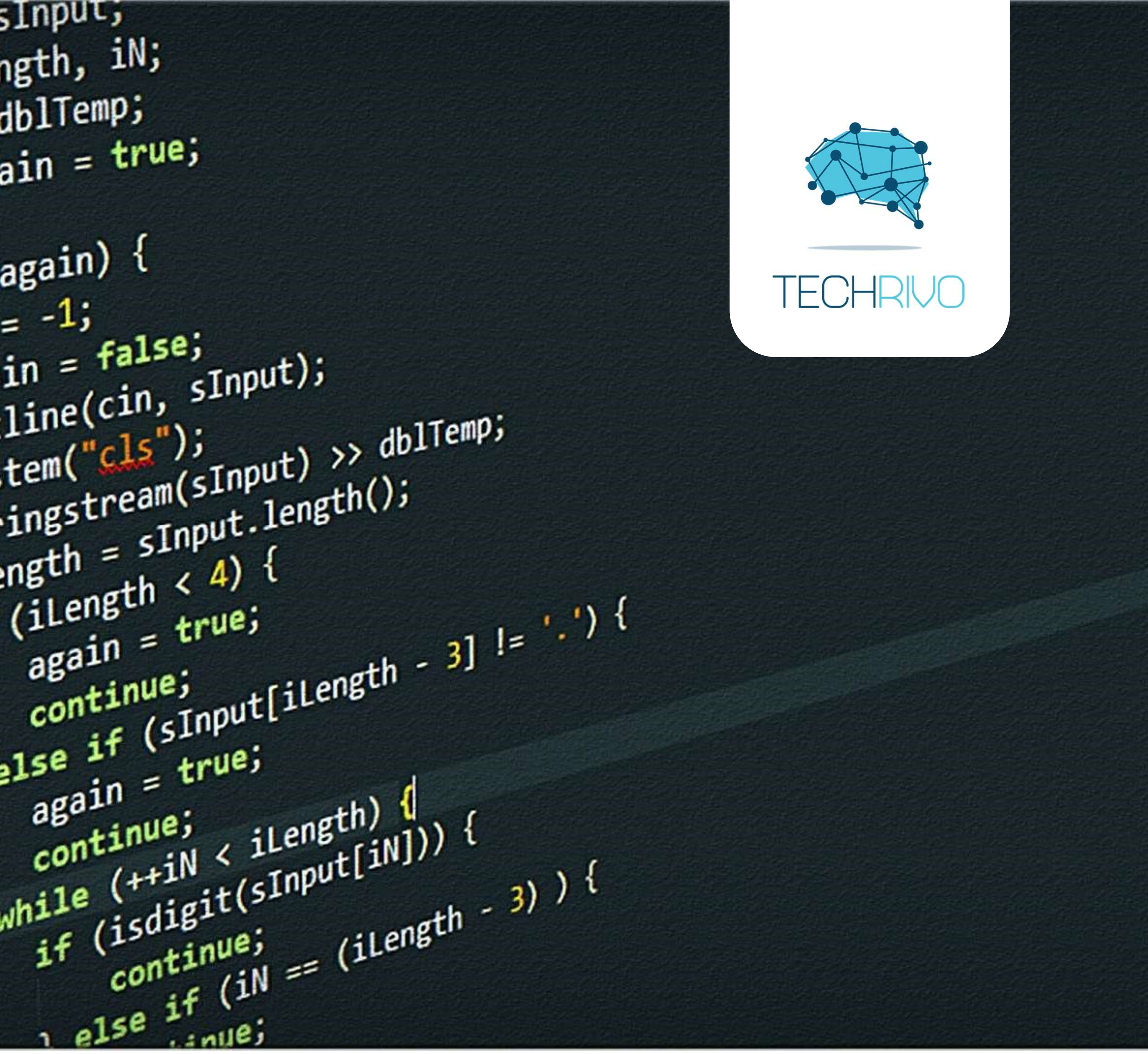


```
sInput;
ngth, iN;
dblTemp;
ain = true;

again) {
= -1;
in = false;
line(cin, sInput);
tem("cls");
ingstream(sInput) >> dblTemp;
ength = sInput.length();
(iLength < 4) {
 again = true;
 continue;
else if (sInput[iLength - 3] != '.') {
 again = true;
 continue;
while (++iN < iLength) {
 if (isdigit(sInput[iN])) {
   continue;
 } else if (iN == (iLength - 3) ) {
   inue;
```

TECHRIVO

# White Paper: Myths and Misunderstandings in building software products

Antonio Gualdino

10th May 2021

As in any other area, software development is flooded with misconceptions and myths. Some derive from the dot com bubble almost two decades ago. Others have been brought to light only recently.

In any case, understanding and knowing how to deal with them is crucial to all stakeholders involved in building quality software.

Most of these misconceptions arose from a specific industry or specific circumstances. Some might even have been true at some point in time, but technological advancement leads them to become simply myths.

These misconceptions revolve around four main themes.

- The Best way to Develop a Product or a Project
- The Software Development Process
- The Developers
- The Product

## The best way to Develop a Product or a Project

This straightforward concept gets many small online business owners up during long evenings, figuring out the best way to build their applications.

The pre-conceived notions associated with online misinformation led entrepreneurs and leaders to discard solutions, partnerships, systems, teams, or projects that otherwise would have been very profitable or productive.

### #1 There's no need for specifications

On the top of the misconceptions list, there is the notion that there is no need to spend money on specifications because nowadays, everything is agile.

Specifications provide a clear blueprint of what needs to be done. They are the base for any software project. The detail of specifications must be, however, balanced. Too detailed might become a burden to the project. Too broad will not serve its purpose. Inexistent is sending the development team into guessing, inventing, and reinventing the wheel.

### #2 Users don't know what they want

The notion that users will be dazzled by a product shown to them just because it's innovative and new comes from several misinterpretations from well-known software and business founders. The misinterpretation of this leads to severe issues in building products and software. It's one of the most profound rooted myths in the industry.

It's not that users don't know. It's more that they don't know how to express it. This happens to business owners and entrepreneurs as well. So instead of figuring out what users want, determine their needs before they realize them. This needs to be done by observing users and customers and doing proper research.

When developing custom software tailored to a particular need, how could you not consider the users?

## #3 Ideas are rare. Executing them is cheap

What is an idea worth if it doesn’t see the day of light? This misconception is precisely the other way around. The world is flooded with great ideas that don’t go anywhere. Countless sums of money are spent on protecting Ideas, NDA’s, patents, and copyrights that don’t get executed or are poorly executed. These ideas simply don’t generate any stream of income. Quite often, great Ideas are poorly executed and don’t even reach the production state.

On the other hand, mediocre ideas get a flawless execution and thrive. Execution is, therefore, everything to get to the market. And it’s not cheap to get it done right.

## #4 Selling a software product isn’t that hard

Yes, it is. Software in the traditional sense is not a commodity that one feels and touches. This simple fact puts it at a disadvantage in the start line. On top of it, there are many items to consider: The demand for the product, how easy it is to use it, are customers willing to change from competitors, the sales cycle, the perception of value the customer has about the software.

## #5 If there’s a plan and we follow it, then everything will be all right

Some development approaches rely on loose planning for software development. Some projects don’t even rely on a plan at all. There’s some truth in saying that having a bad plan is way better than no plan at all. In the end. If there’s no plan, how to know where we’re heading and when to get there?

However, plans need to be flexible because all stakeholders must understand that conditions, requirements, and specifications might change along the way.

This flexibility is crucial to a successful product. Rigidly sticking to a plan, deadlines, budget, or specifications is a halfway trip to a crippled product.

A software project must always consider a contingency plan. This contingency plan must be part of the overall plan and account for delays in deliverables and raises in development costs.

## #6 An MVP is how to start and all that is needed

Much literature exists about the Minimum Valuable Products and their variations. The term is overused, so that pretty much serves for everything in software development.

An MVP is just one step in the creation of remarkable software. Before an MVP gets built, the concept or idea should be proved, and a prototype should get made.

Several iterations or refinements of the MVP could be necessary until a Great Product is achieved.

## The Software Development Process

### #7 Software development is linear and predictable

At least for medium to large projects, it is not.

The only way to avoid this trap is to make sure that product requirements are well known in advance. This aligns with a waterfall approach where everything is planned before a single line of code is created.

However, in today's fast-paced world, this approach only works for specific projects. The need to deliver on an ever-changing world of new requirements makes the process of creating software, mainly if supported by an agile approach, totally unpredictable and non-linear. This clashes with the need to control budget, and delivery dates, so how to resolve this paradigm?

The solution lies on a middle ground between early planning, specification detail, and agile approach. This means the process will still be non-linear and unpredictable but will have more risks mitigated.

The higher the planning and granularity of details, the more predictable and linear the development process will be.

### #8 One development technology or programming language is better than another

In software development, there are also advocates, evangelists, and dogmas about which development language is better. Tiobe indicator defines the popularity of the most common languages among software engineers; however, it doesn't tell us one important aspect, which is its fit for purpose.

To make matters more complex, libraries associated with each language might make a specific language more suitable to a given task than others.

There is also the combination of technologies or [Stacks] that compete for a prime position.

Therefore the technology or language to use must be tightly associated with its fit for purpose. More often than not, the bias towards a development language comes from one's specific experience and not from an independent view.

## #9 Software development is costly

It is, if not done right. The notion that software development is a costly process comes from poor planning and poor execution.

Suppose there's no budget and no control over the software development expenses. There's also no way to know how much will the software development process cost.

The notion that software development is risky and that to account for the unknown, there shouldn't be much planning. Results in projects being much more costly than initially anticipated.

On the other hand, unanticipated issues and subpar execution will lead to bugs, overwork, and technical debt, all of this making the bill significantly higher.

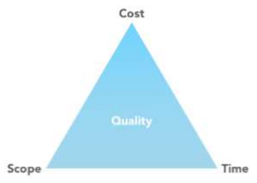
These two factors associated with Bill & Materials contracts for Software Development lead to the rise of this Myth.

## #10 We can't control cost and timeframe when building software

There are many unknown variables in building software, ranging from expectations to project complexity and budget. Tracking and controlling them is not an easy task for a Project Manager.

Cost and time are part of the Project management Triangle model (or triple constrain). The other triangle apex is Scope. By properly managing these constraints, an efficient manager is able to deliver a project in time and within budget at the expense of the other third triangle apex.

The caveat is that the quality of the work is constrained in the triangle. Meaning if one or more constraints are significantly and unfavorably changed from the initial planning, then most likely quality will be at stake.



## #11 Outsourcing will resolve software development issues

Outsourcing is a two-edged sword. If, on the one hand, it allows for offloading work from the internal development team. On the other hand, it brings more challenges in proper knowledge transfer, code quality, and integration.

So, handing over the execution to an external 3rd party might resolve some development bottlenecks. Still, it doesn't come without a significant amount of risks. To mitigate these risks, proper and consistent attention to the relationship with the outsourcing provider is needed.

## #12 Outsourcing means quality will go down

This misunderstanding came from the result of less qualified offshore development firms—and from the bloat of unproven skills and gratuitous use of technology, simply for the sake of it.

This led to the notion that qualified or overqualified outsourcing companies presenting themselves as a cheap alternative could not deliver quality software.

To understand this notion, it’s also crucial to understand the triple constraints described above and the culture and demographics of offshore and outsourcing companies.

Outsourcing can provide excellent quality software, but it all relates to the budget, the available time, and the Scope of the work to be carried out. These will determine the quality of the end product.

There is, however, one more variable that impacts the quality of the produced work in outsourcing. And that is the Engagement towards the chosen 3rd party—the greater the Engagement, the more chances in increasing the quality of the deliverables.

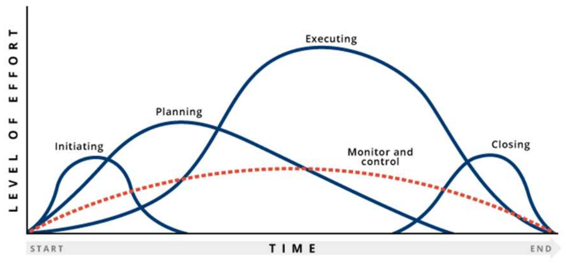
Hand in hand, it all boils down to picking the right provider.

## #13 The work should be done as fast as possible

Quite often, development gets done just on the basis of code only, which means that after a couple of meetings, development gets done without much planning. The rush to time to market usually comes from the Fear of Missing Out on a great idea. Sometimes there is also the notion that if development gets done fast, it will be cheaper.

When a development project is in a tight schedule constrain or has a strict deadline, several other aspects get put in the backburner.

Planning, testing, and quality get put consciously or unconsciously aside. The development team is pressured to deliver, and mistakes start ramping up. This is more critical in the later stages of project execution. The result is that productivity dramatically goes down, and as a consequence, projects get delayed.



Avoiding all project development phases, rushing, and cutting corners is a beginner’s costly mistake.

## The Developers

### #14 More developers mean more work and getting things done faster

This one is counterintuitive because adding more resources to a project when it’s in the execution phase means that the new resources must be onboarded and trained on the work to be carried out. The existing resources that were working in cruise speed will need to slow down to support the newcomers. Depending on the complexity of the project, this onboarding process could take several days or weeks.

On top of this, there are other important factors to consider related to how development teams operate. The increase in code leads to an increase in code review processes, integration needs, in management needs.

The developers must be determined and added prior to or in the initial phases of the coding process. Suppose developers need to be added along the way. In that case, they must be onboarded, ensuring that the proper structure is in place to support them. This prevents disrupting the existing Team.

### #15 Remote developers are worst than in-house developers

Managers need to control productivity. This comes from the Industrial age era, where performance was tightly related to the number of factory worker pieces.

This still holds today in the sense that if one sees a developer sitting at his desk profusely typing, it must mean he’s working hard and producing good software.

The quality of development work is not tied to the number of lines of code, the countless hours of work, arriving early and leaving late, or being at the office.

It’s strictly tied to the developer’s skills, attitude, and reliability, and this ultimately translates into his performance.

There are indeed challenges in handling remote developers, however that has nothing to do with the quality of the outcome produced by the developer, but rather how he’s managed or mismanaged.

## #16 Developers only write code

There is a myriad of developer types. The term is very loosely used to designate someone that produces software. Within the umbrella of developers, there are architects, software engineers, automation developers. The list goes on. In some way or form, they all contribute to building software, but almost all of them don’t code 100% of the time. They analyze, design, estimate, architect, evaluate, log, meet, and also code.

A great deal of the time is spent in understanding the requirements, getting to know what needs to be done. Another significant amount is testing what was done to make sure it was done right under all angles. Depending on some processes, some time is also spent on integrating with existing code and reviewing other’s code.

## #17 More hours of Coding means there is a commitment

Hours and number of lines of code do not relate to the quality or more software produced. However, knowledge and skill do.

A developer might be committed to working long hours, day and night and on weekends, and still, produce very little.  There are several reasons for this. It might be over-optimizing a feature. It might be gold plating. It might be struggling with a task without knowing what to do. It might be simply reinventing a new way of doing things rather than using commonly accepted patterns.

Instead of commitment, coding hours are better associated with productivity metrics like meeting timeframes, task estimations, actual time spend on each task, and generated bugs.

## #18 Developers are cheap and replaceable

The term "Code monkey" is a derogatory term associated with a developer whose skills are very basic or simplistic.

This notion also implies that developers conduct repetitive tasks and that they can be easily replaced in the same way as factory line workers.

This is one of the most dangerous misconceptions that fortunately has less and less adherence in people's minds.

As in any other professional activity, the value of a developer is directly linked to his skills and experience.

One might argue that Junior Developers are cheap and replaceable. Still, most of the time, their contribution to the software project is also marginal.

Good developers are not cheap and are not easily replaceable. Good developers move a project forward on their own. Often a good developer is worth two or more, less skilled developers, with fewer communication issues and less management overhead.

## #19 Developers don't need to know the business. There are Managers and Analysts for it

The more involved Developers are, the more productive they'll become. Instead of just coding what is being handed over by the analysts. By understanding the business, developers also act as a gatekeeper of the requirements. This is valuable not only to be efficient in implementing what is required but also to provide feedback on possible flaws or gaps in the initial scope definition.

As an added advantage, developer teams involved in every aspect of the business tend to grow a more profound sense of accountability for the work carried out.

## The Product

### #20 Adding new features is easy

Adding new features in the beginning of the development process is indeed far more manageable. These new features must come with proper planning and documentation in the form of specifications.

However, adding features when the project is in development or has been released is far more complex, mainly due to two well-known effects: Software entropy and Feature Creep.

Changing existing software code will result in a significant rise in issues and unmet objectives due to the increasing complexity. On the other hand, the more features get added, the more distant the software is from its initial goals. This is why it is far less costly to add features at the beginning of the software development cycle.

Adding new features might seem easy but will have devastating effects if not done right.

### #21 Testing is not needed

Including testing when developing software assures that software has a certain degree of quality. The effort that gets put into the several testing forms has a direct relationship with the quality (perceived or not) of the software.

Without testing, there is no way to ensure that the required functionalities work as initially specified. The sooner test gets put in place, the better. The cost of finding issues is multiplied several times if the issues are found in production rather than in the initial phases of development.

## #22 When the software is released, there's no more work to be done

This Myth is one that most of us want to believe in because it tries to set a finish line not only to get to market but also to stop the cash expense in the project.

In reality, this hardly ever happens. If it does, it's because the project didn't get traction in the market and got abandoned.

After release, there are new features to be added, bugs to be fixed, and a myriad of changes to implement, usually by customer demand.

## #23 Having a Feature-Rich Product means having a great User Experience

User Experience has nothing to do with a product being Feature-rich. It has to do with usability, how information is organized, and Its visual design.

It has to do with how easy it is for the user to accomplish his goals when using the software. A specific Software can have a significant amount of features and still be hard to use, thus having an inferior User Experience.

## #24 UX is about UI

It's also about User Interface, but User Experience is much more than that. It's the whole process of designing the product so that the user has a great experience using it.

User Interface plays a part in it, but it's very reductive to state it's the only factor to focus on. How the user interacts with the software, the design of that interaction, what value is perceived, and ultimately how easy, it is to use the product are some of the main focus to target to achieve a great UX.

## Final Thoughts

It’s important to demystify these myths. Most of the time, they are the root cause of miscommunication between Software Development and Project stakeholders. Some are so deeply rooted in the software industry that they become natural causes for project failures.

For the Entrepreneur aiming to build a digital product, understanding how and why these misconceptions exist is essential to mitigate project risks.