What makes a good spec? [closed]

Asked 16 years ago Modified 7 years, 10 months ago Viewed 12k times

38

Closed. This question needs to be more focused. It is not currently accepting answers.

Want to improve this question? Update the question so it focuses on one problem only by editing this post.

Closed 7 years ago.

Improve this question

One of the items in the Joel Test is that a project/company should have a specification.

I'm wondering what makes a spec good. Some companies will write volumes of useless specification that no one ever reads, others will not write anything down because "no one will read any of it anyway". So, what do you put into your spec? What is the good balance between the two extremes? Is there something particularly important that really, really (!) should always be recorded in a specification?

specifications

Share

Improve this question

Follow

edited May 11, 2012 at 11:39

Taryn
247k ● 57 ● 370 ● 408

asked Dec 18, 2008 at 21:34

Candidasa
8,700 ● 10 ● 32 ● 31

I know this is an old post but I will have my 2c. Joel wrote an article called "Painless Functional Specifications". He describes the fact that whether or not anyone reads it is irrelevant, whats most important is that in order to write it, you most likely had to get everyone together and actually think about each component. The specification, in this regard, can merely be thought of as the "Meeting Minutes". – Talon Jan 29, 2016 at 6:15

12 Answers

Sorted by: Highest score (default)

54

The best spec is one that:

1. Exists
2. Describes WHAT, not HOW (no solutions)
3. Can be interpreted in as few ways as possible
4. Is widely-distributed
5. Is agreed-upon as being THE spec by all parties involved
6. Is concise
7. Is consistent

8. Is updated regularly as requirements change
9. Describes as much of the problem as is possible and practical
10. Is testable

Share Improve this answer Follow

edited Dec 18, 2008 at 22:52

answered Dec 18, 2008 at 22:47

Robert C. Barth
23.3k ● 6 ● 48 ● 53

2 Alas, finally someone that understands the purpose of specification. – Hugo Sereno Ferreira Dec 19, 2008 at 0:52

I expanded a little bit on this list in a blog post if anyone's interested. norimek.com/blog – Robert C. Barth Dec 20, 2008 at 0:45

@RobertC.Barth I can't look at your blog, it says: 404 Not Found nginx/1.22.0 (Ubuntu) – MJ DLS Sep 17, 2023 at 10:32

14

What to put in a spec

You need to look at the audience of the spec and work out what they need to know. Is it just a document between you and a business sponsor? In this case it can probably be fairly lightweight. If it's a functional spec for a 100+ man-year J2EE project it will probably need a bit more detail.

The audience

The key question is: who is going to read the spec - A spec will have several potential sets of stakeholders:

- The business owner who is signing off the system.
- The developer who is building the system (which may or may not be you)
- QA people who have to write test plans for it.
- Maintenance staff wanting to understand the system
- Developers or analysts on other projects who may want to integrate other systems into it.

Requirements of typical key stakeholders:

- The **business owner** needs to have a clear idea of what the system workflows and business rules are so they can have a fighting chance of understanding what they have agreed to. If they are the only major audience of the spec, concentrate on the user interface, screen-screen workflow and business and data validation rules.

- **Developers** need a data model, data validation rules, some or all of the user interface design and enough description of the expected system behaviour so they know what to build. If you are writing for developers concentrate on the user interface, mapping to data model and rules in the user interface. This should be more detailed than if you are doing the development yourself because you are acting as an intermediary in a communication between two third parties.

  If you are specifying an interface between two systems, this has to be very precise.
- **QA staff** need enough information to work out how to test and validate the logic, validation and expected user interface behaviour of the application. A spec intended for developers and QA staff needs to be fairly unambiguous.
- **Maintenance staff** need much the same information as developers plus a system roadmap document describing the architecture.
- **Integrators** need a data model and clear definitions of any interfaces.

Key components of a spec:

I'm assuming that one is writing specs for business apps, so the content below is geared to this. Specs for other types of systems will have different emphasis. In my experience the key elements of a functional spec are:

- **User Interface:** screen mockups and a description of the interaction behaviour of the system and workflow between screens.
- **Data Model:** Definition of the data items and mapping to the user interface. User interface mappings are normally done in the bits of the spec describing the user interface.
- **Data Validation and Business Rules:** What checks for correctness need to be be made on the data and what computations are being made, along with definitions. Examples can be quite useful here.
- **Definitions of interfaces:** If you have interfaces exposed that other systems can use, you need to specify those pretty tightly. The simpler internet RFC's give quite good examples of protocol designs and are quire a good start for examples of interface documents. Clearly defining interfaces isn't easy but almost certainly save you grief down the track.
- **Glue:** this is where use cases, workflow diagrams and other requirements related artifacts help. Generally an exhaustive listing of these is pointless, but there will be key areas within the system where this type of documentation helps to put items in context. My experience is that selective inclusion of use cases and other requirements level descriptions does a lot to add clarity and meaning to a spec but

> writing up a user story for every single interaction with the system is a waste of time.

Joel (of 'on software' fame) wrote a good series of articles on this called *Painless Functional Specification* which I've referred people to on quite a few occasions. It's quite a good set of articles and well worth a read. In my opinion, your objective is to clearly explain what the system is supposed to do in a way that minimises ambiguity. It's quite useful to think of the spec as a reference document - what might the various stakeholders want to be able to easily look up.

Having written a glib set of bullet points about specs, the clear communication part is harder than it looks. Specs are actually non-trivial technical documents and are quite a test of one's technical writing and editorial skills. You are actually in the business of writing document that describes what someone is supposed to build. Doing good specs is a bit of an art.

The pay-off for doing specs is that no-one else wants to do them. As you've written what is probably the only document of any importance for the system, you get to call the shots. Anyone else with an agenda has to either lobby you to change the spec or somehow impose a competing spec on the project. This is a good example of the pen being mightier than the sword.

EDIT: It has been my experience that debate about the distinction between 'how' and 'what' tends to be pretty self-serving. On any non-trivial project the data model and user interface will have multiple stakeholders, not all of whom are the system's developers. Working in data warehousing will give one a taste for the chaos that ensues when an application data model is allowed to become a free-for all, and PFS should give one a feel for the potential set of stakeholders a spec has to cater to.

The fact that someone owns a data model or user interface design doesn't mean that these are just decided by fiat - there can be a discourse and negotiation process. However, as a project gets larger the value of ownership and consistency in these gets greater. It's been my observation in the past that the best way to appreciate the value of a good analyst is to see the damage done by a bad one.

Share Improve this answer Follow

edited May 23, 2017 at 11:46
Community Bot
1 ● 1

answered Dec 18, 2008 at 22:19

ConcernedOfTunbridgeWells
66.5k ● 15 ● 148 ● 198

I disagree with data model and UI. Those describe "how." A spec only describes "what." Capturing data elements is different from creating a data model. – Robert C. Barth Dec 18, 2008 at 22:51

Someone has to design the data model and UI and much hilarity will ensue if either of these are allowed to become a free for all. Reporting and MIS are often the poor cousins in this regard. I know of one European telco that has 800 people working in just one of its data warehousing units (the mobile division has its own data warehouse department) - largely because of poor data architecture. I'm also aware of a company selling financial software that is essentially bankrupt because of the TCO of its system driven by constant data fixes. – ConcernedOfTunbridgeWells Jul 22, 2009 at 8:52

11

In my experience a spec will have more chance of being read if it has the following:

- Use diagrams where possible - pictures are worth 1000 words
- Have a title page that clearly indicates what the spec is describing
- Have a style that is used throughout the document. Make all headers the same font, size and style. Make the font the same all the way through, use the same bullet styles etc
- DONT WAFFLE - Be clear concise and to the point, and don't add extra cruft to pad out your document. If a point can't be explained in a few lines of text, then maybe you need to break it down further

I have seen in companies where the person writing the spec doesn't understand the system. It's almost a way of learning the system by writing the spec. This usually ends in tears...

Share Improve this answer Follow

answered Dec 18, 2008 at 21:42

JamesSugrue
15k ● 10 ● 62 ● 93

1 Diagrams aren´t searchable. If you have a hundred diagrams, how do you search for information contained in them? You can search through text... – Alvaro Rodriguez Dec 18, 2008 at 22:44

Give the diagram a useful title in text. If you have a lot of words instead it is going to make search no different as you will be searching on the keywords anyway – JamesSugrue Dec 18, 2008 at 23:23

I prefer textual descriptions. But that is my preference, all I'm saying is don't skip the description because you have a diagram. Not everyone sees things the same way. – Brian B. Dec 19, 2008 at 1:15

ASCII diagrams are searchable! – Connor Doyle Feb 12, 2015 at 16:02

4

As someone who develops bespoke software for clients, **the best spec is the one which the customer has signed**.

It doesn't matter how refined your spec is - if the customer hasn't explicitly agreed to it in writing, they'll change it and expect you to roll with their changes seamlessly, wrecking your beautiful architecture...

Share Improve this answer Follow

answered Dec 18, 2008 at 22:13

Dan Vinton
26.8k ● 9 ● 41 ● 82

3 Yeah... And after they sign it, they say: well, we didn't understand it the way you did, so... A beautiful architecture is the one that will adapt to their changes. Embrace them, don't deny ;-) – Hugo Sereno Ferreira Dec 19, 2008 at 0:54

2

Good specs should contain requirements that are measurable and verifiable. When looking at each requirement, you should be able to easily answer the question, "How can I prove I have fulfilled this requirement?".

Share Improve this answer Follow

answered Dec 18, 2008 at 23:34

dalesmithtx
247 ● 1 ● 11

1

Read Joel's series of "[Painless Functional Specifications]" followups to the Joel Test article. They also appear in the "Joel on Software" book.

Share Improve this answer Follow

answered Dec 18, 2008 at 21:44

Dave Ray
40k ● 7 ● 85 ● 82

1

Depends on how big the project is and (like all architecture decisions) what the constraints are. A good start is

- a short description, a "one pager"
- a context diagram -- where are the boundaries, what interacts with the system?
- use cases/user stories
- a GUI prototype or paper prototype, if applicable
- a description of the needed nonfunctional requirements (performance etc.)

Best of all is to have an acceptance test, ie, a testable statement of things that can be checked, along with an agreement that when those things are done, the project is complete.

Share Improve this answer Follow

answered Dec 18, 2008 at 22:06

Charlie Martin
112k ● 26 ● 196 ● 264

1

It also helps if you start by stating the goal the user has or what the global idea of a certain function is; rather than filling in the exact implementation. This always feels to me like narrowing down the open mindedness or using less creative (more usable) solutions. So you should keep "all options open".

Example Your writing a software to measure "X".

Instead of stating: There has to be a start button and a save button.

Use: The user has to be able to start a measurement and save it.

Why? Because in the first situation you already determined what the solution has to be, while the second situation gives you flexibility on **how** to implement something. Now this may seem trivial, but I have the feeling "programmers" tend to think more in solutions rather than in problems (or situations). When you add more functionality this becomes more obvious, because then it might have been better to use a wizard or automate the process, but you already narrowed the idea's down to using buttons.

Share Improve this answer Follow

answered Mar 13, 2009 at 13:33

Ivo Flipse
10.3k ● 18 ● 51 ● 63

1

For functional requirements—or, more specifically, behavioral requirements—I like to use Cucumber and Gherkin.

Here’s an example of a simple and short specification for a new feature in a simple mapping application. The feature allows small businesses to sign up to the mapping platform and add their places of business on a Google Maps-like service.

```
Feature: Allow new businesses to appear on the
map

  Scenario Outline: Businesses should provide
```

```
required data

    Given a <business> at <location>
     When <business> signs up to the map
platform
     Then it <should?> be added to the platform
      And its name <should?> appear on the map
at <location>

    Examples: Business name and location should
be required
      | business         | location | should?
|
      | UNNAMED BUSINESS | NOWHERE  | shouldn't
|

    Examples: Allow only businesses with
correct names
      | business         | location
| should?   |
      | Back to Black    | 8114 2nd Street,
Stockton | should    |
      | UNNAMED BUSINESS | 8114 2nd Street,
Stockton | shouldn't |

    Examples: Allow businesses with two or more
establishments
      | business      | location
| should? |
      | Deep Lemon    | 6750 Street South, Reno
| should  |
      | Deep Lemon    | 289 Laurel Drive, Reno
| should  |
```

This specification looks deceivably simple, but is in fact quite powerful.

- Good specifications are clear, unambiguous and concrete. They don't need to be deciphered in order to write working code. That's exactly what Gherkin specs are. They're best served short and simple.

Instead of writing a long ass specification document, you let the specification suite evolve along with your product by writing new specs in every iteration.

- Gherkin is a business-readable language for writing specification documents based on the Given-When-Then template. The template can be automated into acceptance tests. Automating the specification ensures it stays up to date because the captured conversation is directly tied to testing code. This way, tests can be used as documentation, because Gherkin features have to change every time the code changes.
- When each business rule is given an automated test, Gherkin specifications become so-called executable specifications—specifications that can be run as computer programs. The program tests whether the acceptance criteria were implemented correctly. So at the end of the day, we get a yes-or-no answer to the question of whether our product is actually doing what we expect it to do—which in itself is very valuable, as it contributes to making software of better quality.
- The direct connection between Gherkin specifications and testing code often reduces the damage of waste by creating and cultivating a system of living documentation. Thanks to frequent validation of tests, as in continuous integration systems, you can know that Given-When-Thens are still up to date—and when you trust your tests, you

can use the corresponding Gherkin specifications as documentation for the entire system.

- In fact, there’s an entire methodology called Specification by Example that uses tools like Gherkin. Specification by Example's practices reduce possibility for misunderstandings and rework by giving you a framework for talking with business stakeholders by forcing you to use concrete, discrete, unambiguous examples in your specification documents.

If you want to read more about Cucumber, Gherkin, BDD, and Specification by Example, I wrote a book on the subject. “Writing Great Specifications” explores the art of writing great scenarios and will help you make executable specifications a core part of your development process.

If you are interested in buying “Writing Great Specifications,” you can **save 39% with the promo code 39nicieja2** :)

Share Improve this answer Follow

answered Feb 1, 2017 at 21:13

thion

101 ● 1 ● 1

0

I think writing "Use cases" should save you bunch of pages

Share Improve this answer Follow

answered Dec 18, 2008 at 21:36

Chanakya
875 ● 2 ● 9 ● 22

0

+1 @KiwiBastard and I would add write bullet-like and make each bullet testable.

Share Improve this answer Follow

edited May 23, 2017 at 12:25

Community Bot
1 ● 1

answered Dec 18, 2008 at 21:49

kenny
22.3k ● 8 ● 51 ● 87

1 Im Pretty sure this should be a comment, not an answer – JsonStatham Aug 12, 2013 at 14:04

0

A blueprint that describes all of the critical information necessary for the implementation, but doesn't waste any effort on describing all of the trivial or obvious information that is also necessary.

It should just be enough information to insure that the implementation is "as expected", without providing too much additional noise that isn't necessary.

In practice, most people get this wrong, as they focus on the easy stuff (which is the least necessary) and shy away from the hard stuff (which is what you really really want to lock down). I've seen way too many 2 inch documents that completely and utterly miss the point, and very few 3 page ones that hit it dead on.

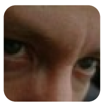
Specs don't have to be long, they just have to contain the right stuff!

(hint: if the programmer didn't look at that page while coding, it probably wasn't required)

Paul.

Share Improve this answer Follow

answered Dec 18, 2008 at 23:26



Paul W Homer

2,740 ● 1 ● 19 ● 25