What exactly defines production?

Asked 15 years, 10 months ago Modified 7 years, 2 months ago

Viewed 15k times

17

Like almost anyone who's been programming for a while, I'm familiar with the term "production code" and have a vague sense of what it means. However, can someone offer a semi-rigorous definition, since it seems Wikipedia and Google can't? It seems like there are a lot of gray areas in what counts as production, such as internal tools that are used by a small group of people and therefore not "formalized" in terms of UI, documentation, etc. and open source apps that are feature complete, reasonably bug free and working, but lack polish, UI and extensive testing.

definition **semantics** **production**

Share

Improve this question

Follow

edited May 18, 2010 at 1:48

Jon Seigel

12.4k ● 8 ● 60 ● 93

asked Jan 29, 2009 at 2:23

dsimcha

68.6k ● 55 ● 219 ● 340

9 Answers

Sorted by: Highest score (default)

27

When your code runs on a production system, that means it is being used by the intended audience in a real-world situation.

Production code, however, does not necessarily mean robust, reliable, or stable code. The Daily WTF provides plenty of evidence in this regard.

Share Improve this answer

Follow

answered Jan 29, 2009 at 2:33

Jeff

21.9k ● 6 ● 52 ● 55

This doesn't cover software like internal tools, where the intended audience is a group of people inside the company. I'm not sure if I would consider that "production code."
– Tim Frey Jan 29, 2009 at 6:29

1 I would count internal tools as production code, perhaps unless the target audience is the same people who maintain the tools. For example, if the business depends on internal tools working, then it is production code. – Eddie Jan 29, 2009 at 7:03

3 I agree with Eddie: "internal tools" is meaningless, what matters is intent. If things break and money is lost and the ultimate goal is thwarted, then the relevant processes and actors are part of "production" by definition. – Rob Williams Jan 29, 2009 at 19:34

Production means anything that you need to work reliably, and consistently.

22

Whether is a build script, or a public facing web server.

When others rely on your code, particularly folks who may not understand it (i.e. even "smart" developers but perhaps not in your group, but using a library you wrote), that code is production code.

It's production because "work stops" and "money is lost" when the production code fails.

Share Improve this answer

Follow

answered Jan 29, 2009 at 2:26

Will Hartung

118k ● 20 ● 133 ● 207

Can you explain the work stops portion? Can't a developer just work to maintain code or fix bugs when they happen? I think thats part of the software development cycle – committedandroider Dec 29, 2014 at 2:20

@committedandroider I believe he meant that those who rely on the software product to do work can no longer do it. Their work stops. – Usagi Jul 12, 2018 at 18:21

8

The definition as I understand it is that production code is any code that is installed or in use on a live, non-test-bed system. A server used internally to a company is a production system if it is the live system used by the employees of the company. The point here is that code running on a server internal to the company writing the code can be production code.

Usually, a good distinction when looking at internal code is whether or not the group maintaining the code is separate from the group using the code. If the groups are separate, odds are that the code is production code. If running the business depends on the code, then it is certainly production code, even if it is developed and maintained in-house.

Share Improve this answer Follow

edited Jan 29, 2009 at 20:57

answered Jan 29, 2009 at 2:25

Eddie

54.4k ● 22 ● 128 ● 146

How is this circular, exactly? The question is fairly basic, so I gave a basic answer. Is something deep needed here? – Eddie Jan 29, 2009 at 6:30

@Rob Williams: Yes, I agree that the 2nd sentence of my answer is totally redundant. I will delete the 2nd sentence. This is a fair criticism. – Eddie Jan 29, 2009 at 20:53

I deleted the 2nd sentence and added a clarification to make more obvious the point I was aiming for. – Eddie Jan 29, 2009 at 21:03

5

EDIT: The short answer: If you are "betting the farm on it", it is ***"production"***.

This is a great question--an absolutely critical distinction that routinely gets everyone in trouble due to

misunderstandings. The question of what is "*production*" is a subset of the related question of what is an "*environment*".

> **So part of the answer is that "*production*" is THE "*environment*" that is most important and is most trusted as THE "*real*" thing.**

So now we must define "*environment*" (and then revisit "*production*"). **We are still far from a satisfactory answer.**

We programmers use the term "*environment*" constantly to refer to computer systems consisting of hardware that is executing software. That software is the code that we wrote plus software that it depends upon, which was written by others. We write our code and integrate it with the other software, then we typically run the integrated software through an escalating series of tests (unit tests, integration tests, functional tests, acceptance tests, regression tests, etc.), until we finally run the integrated software in the full manner in which it was intended.

Of course, not everything is fully automated. There are usually numerous people involved, and they have manual processes to perform. We programmers look for ways to automate as many of these processes as possible, but there is always a "man/machine boundary" in the systems we work on. Often, there are many such boundaries in any particular case.

On the other hand, there may not be any significant automation at all. For example, we spoke of "*production*" way back when we had a room full of people performing manual labor which *produced* a *product*. So, there doesn't have to be any automation present in our "*production*" "*environment*". There is also a middle ground, where the automation involved does not include software, such as in the case of a person running a loom to weave cloth.

Also, there may not be a *product*, since we have adapted our language of "*production*" "*environment*" to include product-less service providers.

Likewise, the testing may not involve software, since we may be testing a non-software-driven machine (e.g., the loom) or even the people (training and evaluation).

Now we have touched on all the crucial elements of an "*environment*":

- there is a purpose, an `intent`, being pursued
- an `intent` requires an intender, so there must be a `sponsor` (a person or group, but not a machine) that specifies the `intent`
- that `intent` is pursued through various `processes` that are performed by various `actors`
- those `actors` may be people, they may be software executing on hardware, or they may be non-software-driven machines, so there may or may not be automation present

Now we can properly and fully define our original terms.

> An `environment` consists of all the `processes` and their `actors` that collaborate to pursue a particular `intent` on behalf of its `sponsor`. That means software executing on hardware, that means non-software-driven machines, and that means people performing their various duties. It is the `intent` that primarily defines an `environment`, not its `processes` or its `actors`.

Furthermore...

> If the `intent` being pursued in a particular `environment` is the `sponsor's` ultimate goal, which usually involves producing a `product` or providing a `service` in exchange for money, then we refer to that `environment` as `production`.

Now we can go a bit further.

> If the `intent` being pursued in an `environment` is the verification of `processes` and their `actors` in preparation for `production`, we call that a `test` `environment`.
>
> We further call it an `integration` `environment` if that testing involves the initial joining together of

> significant individuals or groups of **`processes`** and their **`actors`**.
>
> If that preparation involves the "programming" of human **`actors`** to perform new **`processes`**, or the subsequent verification (evaluation), then we call that a **`training`** **`environment`**.

Armed with these distinctions and definitions, we can now understand several common scenarios.

An **`environment`** can be mislabeled with a name that does not match its **`intent`**, such as when a **`training`** environment is used as **`test`**.

An **`environment`** can be grossly misused, such as when **`integration`** or **`training`** is done in **`production`**.

An **`environment`** can be misrepresented, such as when key **`processes`** or **`actors`** are left unidentified (e.g., manual reconciliations, or even by ignoring the people altogether).

An **`environment`** can be retasked, by repurposing its **`processes`** and **`actors`** to a new **`intent`**. A very successful technique for some organizations is to routinely "flip" several sets of **`actors`** (servers hosting software) between **`production`**, **`test`**, **`training`**, and **`integration`** upon each release.

In most cases, a single **`actor`** (person or hardware) can execute multiple **`processes`** which can participate in

multiple **`environments`**. For example, a single computer server can host software that performs **`production`** transactions while also hosting other software that performs **`test`** or **`training`** functions.

Normally, a single instance of an **`actor`** should participate in only one **`environment`** at a time. On very rare occasion, a single **`actor`** can be shared across **`environments`** if the **`intents`** are mutually compatible. Most of the time, it is very unwise to attempt such sharing because the **`intents`** are not really compatible. A perfect example is running a **`test`** **`process`** on a server that also supports **`production`** **`processes`**, resulting in downtime because the **`test`** caused the entire server to fail.

Therefore, the **`intent`** of an **`environment`** must be construed with very wide latitude, to include concepts such as *availability*, *reliability*, *performance*, *disaster recovery*, *accuracy*, *precision*, *repeatability*, *longevity*, etc. This means that the **`actors`** and **`processes`** must often be construed to include things like *providing power*, *cooling*, *backups*, and *redundancy*.

Finally, note that the situation can get quite complex. For example, a desktop computer (**`actor`**) may be tasked by the development team (**`sponsor`**) to host their source control (**`process`**), which the team relies upon for their primary jobs (**`production`**). Nevertheless, the IT staff sees that same desktop computer as simply a developer workstation (**`development`**, not **`production`**) and treats it with contempt and nonchalance when it develops a

hardware problem. But the developers are producing `production` code, so aren't they also part of `production`? Perspective matters.

EDIT: Production quality

A solid verification (`testing`) methodology should take packaged code from `development` and run it through a series of `tests` (integration, TQA, functional, regression, acceptance, etc.) until it comes out the other side "stamped" for `production` use. However, that makes the package `production` *quality*, but not actually `production`. The package only becomes `production` when a `sponsor` actually deploys it into an `environment` with that ultimate level of `intent`.

However, if your organization merely produces that package (its `product`) for the consumption of others, then such a release comes as close to `production` as that organization will experience with respect to that `product`, so it is common to stretch the term `production` to apply rather than clarify that it is `production` *quality*. In reality, that organization's `production` environment consists of the `actors` and `processes` involved in its development/release efforts that result in that `product`.

I said it could get quite complex...

edited Jan 29, 2009 at 20:09

answered Jan 29, 2009 at 6:22

Rob Williams

7,921 ● 1 ● 37 ● 42

This is a great collection of information! Structure, clear definitions, etc. Thanks! – florien Apr 4, 2017 at 22:13

1

Any code that will be used by it's intended userbase would fit into my definition of 'production code'.

Of course, the grey area in that definition would be clearly defining who your userbase is.

G-Man

Share Improve this answer

Follow

answered Jan 29, 2009 at 3:26

GeoffreyF67

11.1k ● 11 ● 50 ● 57

1

- The production software can perform at the necessary workload without disruption or degradation of the service
- Software has been successfully tested in different production scenarios
- Transforming working prototype into production software which runs on fail-safe redundant architecture that can work in real business, i.e. production environment, needs time, code refactoring, and attention to details

- The production code has acceptable level of maintainability and is reasonably well commented
- The documentation manual explains functionality, all features and facilitates maintenance
- If the production software is an international service or application, it must be localized
- Production code is used by end-users, often customers under conditions described in Terms-of-Service Agreement
- Production software does not necessarily mean reliable mission critical software
- The software does well, what it was intended to do
- Log files provide an accurate description of run-time performance and software reliability metrics and reporting which do facilitate debugging and software maintainability

Share Improve this answer Follow

edited Apr 5, 2017 at 15:55

answered Dec 30, 2014 at 22:12

jjpcondor

1,416 ● 1 ● 20 ● 30

1

I think the best way to describe it, is as any code that "leads-to" deployment and "follows-up" deployment. Deployment itself is defined as all of the activities that make a software system available for use. If your code is

ready to be used by people, in-house or otherwise, then it is production code.

Share Improve this answer

Follow

answered Sep 26, 2017 at 22:14

Sammy CV

11 ● 3

0

In simple words "Production code which is live and in use by its intended audience"

Share Improve this answer

Follow

answered Jan 29, 2009 at 7:20

Bhushan Bhangale

11k ● 5 ● 45 ● 72

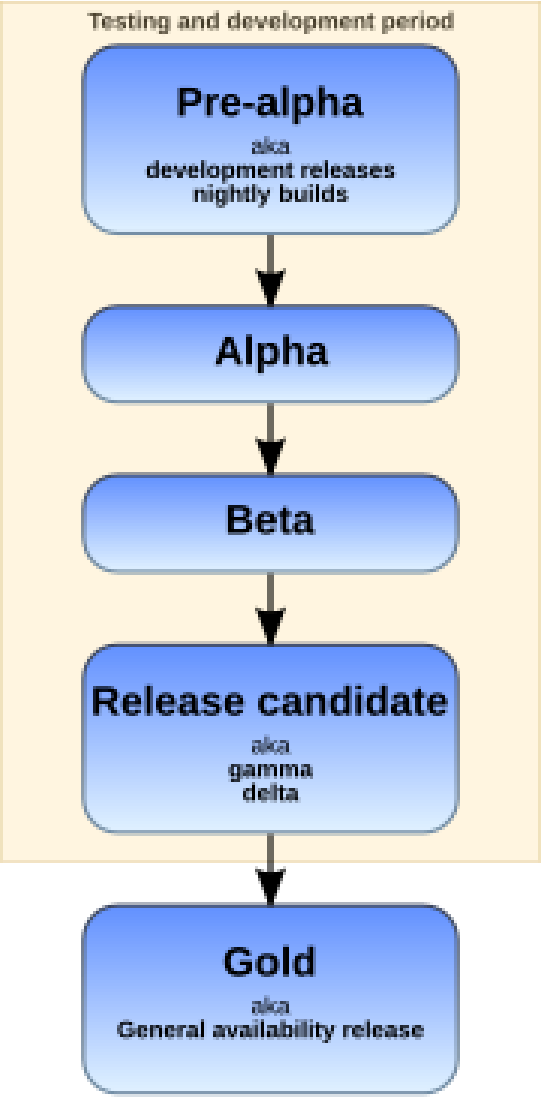
-1

The term "production code" mixes two different concepts. One is deployment management and the other is release life cycle.

In the strict sense of the word, a system is in production when it is being used as part of business or service operation. What's not in production are development, testing, QA, demo, and staging system. Production system does not immediately imply quality.

From release life cycle's point of view, a "production" build is the build that is released to general public or clients. It is the stage after pre-alpha, alpha, beta, (feature complete, code complete, etc.) and release

candidate. For shrink-wrap products that cannot easily deploy updates, reaching the production stage likely implies series of testing and bug fixes.



Share Improve this answer

Follow

edited Feb 8, 2017 at 14:09

Community Bot

1 ● 1

answered Jan 29, 2009 at 7:17

Eugene Yokota

95.5k ● 45 ● 217 ● 320

This really makes it fuzzy. The "production" release package SHOULD be the exact same package that went from development through all the testing stages. But that

verification process merely establishes that package as "production quality" (suitable for, but not in use).
– Rob Williams Jan 29, 2009 at 19:08