۱.

الف) در Unit testing، برای تست کردن هر واحد، به جای استفاده از اشیأ اصلی از اشیأ ساختگی استفاده می‌شود که به آن‌ها Test Double گفته می‌شود.(مانند mock، stub و ...)

ب) به مجموعه‌ای از فعالیت‌ها که طولانی‌ترین مسیر را در طی یک پروژه تشکیل می‌دهند، Critical Path گفته می‌شود. اگر از این مجموعه برای برنامه‌ریزی پروژه (Project Scheduling) استفاده شود، به این روش Critical Path Method می‌گویند.

ج) Risk: ریسک به هر حالتی گفته می‌شود که عدم قطعیت و امکان ضرر وجود داشته‌باشد. می‌تواند از جهات مختلفی بررسی شود؛ مانند ریسک ابزار، ریسک نیازمندی‌ها، شناخته شده، قابل پیش‌بینی بودن یا نبودن، چه چیزی را تحت تأثیر قرار می‌دهد(پروژه، محصول یا Business) و ...

Risk Severity (Impact): به معنای میزان شدت ریسک که می‌توانیم آن را سطح‌بندی کنیم؛ مانند: فاجعه‌بار (در شرایطی که جان انسان‌ها در خطر باشد)، جدی، قابل تحمل، ناچیز

Risk Likelihood: احتمال وقوع اتفاق مورد نظر که باز هم می‌توانیم آن را سطح‌بندی کنیم؛ مانند: ناچیز، کم، متوسط، زیاد و خیلی زیاد

د) به مجموعه‌ای از اطلاعات برای یک عنصر از سیستم (مانند Firmware یا Database) که برای مدیریت پیکربندی استفاده می‌شود، Software Configuration Item می‌گویند.

ه) Cycle Time یکی از معیارهای اندازه‌گیری در Agile است. این معیار نشان‌دهنده مدت‌زمان لازم برای اعمال تغییر است.

و) یکی از دسته‌های نگه‌داری نرم‌افزار (Software Maintenance) که به معنای بازنویسی یا Refactor کردن کد است.

۲.

الف) پارتیشن‌ها:

P1) No white-space (input = output)

Test case e.g.:

input = “HelloWorld!” output = “HelloWorld!”

P2) Single white-space (input = output)

Test case e.g.:

input = “Hello World!” output = “HelloWorld!”

P3) Multiple white-spaces (input ≠ output; need to remove redundant white-space(s))

Test case e.g.:

input = “Hell o W or ld!” output = “HelloWorld!”

P4) Input isn’t a string (input ≠ output, Error: wrong input)

Test case e.g.:

input = “109” output = “Error! Input must be a string”

ب)

جامع (Exhaustive Testing): تست جامع برای قسمت‌های مهم و سیستم‌های حیاتی به کار می‌رود. با توجه توضیحات متد، بعید است که این نوع تست مناسب این متد باشد.

جعبه سفید(White box): این نوع از تست برای قسمت‌هایی مناسب‌تر است که از کارکرد و جزئیات آن اطلاعات کاملی در اختیار باشد. در این تست با توجه به نحوه‌ی کارکرد سیستم، سناریوهایی طرح می‌شود که

پیش‌بینی می‌شود سیستم در آن‌ها دچار اشکال شود. اگر بدنه‌ی متد مذکور را در اختیار داشته‌باشیم، می‌توانیم از تست جعبه سفید استفاده کنیم.

جعبه سیاه(Black box): این نوع تست بدون توجه به ساز و کار داخلی سیستم طراحی می‌شود. رفتار سیستم تنها داده‌ای است که می‌توان براساس آن از درستی کار سیستم اطمینان حاصل کرد. اگر از ساز و کار متد هیچ اطلاعاتی در دست نباشد، می‌توان از این نوع تست بهره برد.

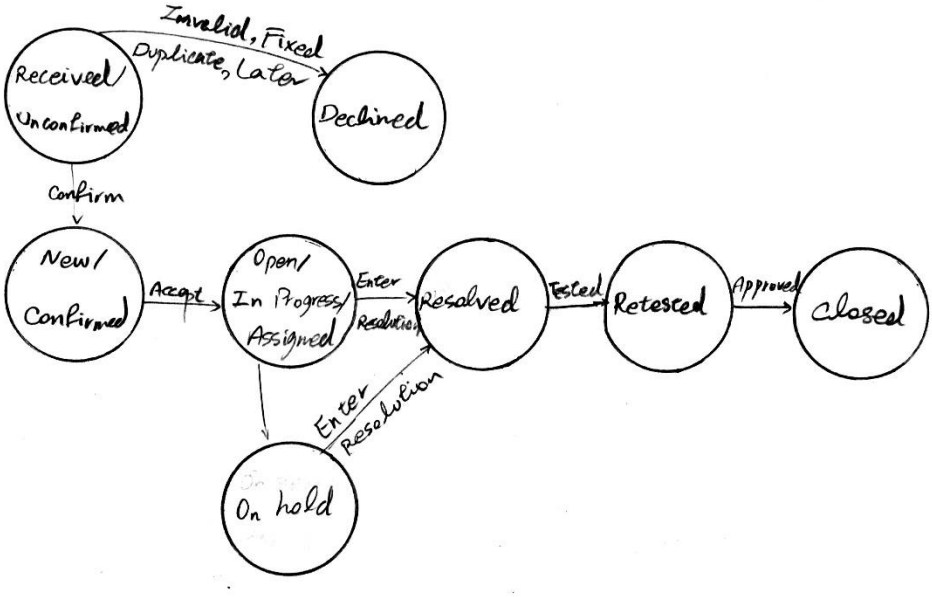
جعبه خاکستری(Grey box): اگر اطلاعاتی در مورد ساز و کار سیستم وجود داشته‌باشد اما از جزئیات کامل آن بی‌خبر باشیم، این نوع از تست مناسب است.

۳. هدف از تست پیدا کردن اشکالات یک سیستم است و نه درستی کامل آن. برای همین، هر چند تست حالات متفاوتی را در نظر بگیرد و جامع باشد، حتی در صورت عدم وجود اشکال، نمی‌توان از درستی صددرصد برنامه مطمئن بود.

۴.

ابتدا issue وارد سیستم می‌شود. در صورتی که ایرادی وارد نباشد، قبلاً به آن رسیدگی شده باشد یا تأیید شده‌باشد(تکراری باشد) و در یک release آینده به آن رسیدگی شده باشد رد می‌شود.

در صورت جدید بودن و تأیید، به لیست issueها اضافه می‌شود. با شروع رسیدگی به آن وارد حالت Open می‌شود. در صورت رفع اشکال، وارد حالت Resolved می‌شود. در صورت تست شدن به حالت Retested و با تأیید وارد حالت Closed می‌شود و چرخه حیات آن خاتمه می‌یابد.



۵.

مهندسی مبتنی بر مولفه(Component based engineering) به معنای تولید یک سیستم با کنار هم قراردادن مولفه‌های ازپیش‌آماده است. این مولفه‌ها می‌توانند توسط خود تیم توسعه یا از بیرون تهیه شوند.
فعالیت‌های اصلی:

Domain Engineering: ابتدا باید حوزه‌ی فعالیت (حساب‌داری، رابط کاربری، و ...) شناسایی شود. سپس مولفه‌های مخصوص هر حوزه که می‌توانند مورد استفاده قرار بگیرند را بررسی می‌کنیم. سپس یک مسنتد (Documentation) تهیه می‌کنیم که در آن هر مولفه و مشخصات آن آمده‌اند.

Component Qualification: در این مرحله مولفه‌های مورد نیاز غربال می‌شوند تا آن‌هایی که مناسب‌تر خواسته‌ی ما هستند مشخص شوند.

Component Adaptation: در این قسمت باید مطابقت‌پذیری مولفه‌ها بررسی شوند. در صورت نیاز می‌توان از Wrapperها، Adapterها و یا APIها به منظور سازگاری استفاده کرد.

Component Composition: این قسمت آخر است که اجزا با هم ترکیب شده تا سیستم اصلی را شکل دهند.

مقایسه مهندسی مبتنی بر مولفه و رویکرد خط تولید نرم‌افزار: نقطه اشتراک این دو روش در Domain Engineering است؛ به این معنا که هر دو به دنبال راه‌حل‌های ممکن در یک حوزه خاص هستند. هم‌چنین هر دو رویکرد منجر به محصولی با قابلیت استفاده‌ی مجدد خواهدشد.

در رویکرد خط تولید، هدف تولید یک هسته مشترک به منظور استفاده در برنامه‌های مختلف است اما در مهندسی مبتنی بر مولفه، هدف تولید نرم‌افزار با استفاده از قسمت‌های ازپیش‌آماده است. در خط تولید رابطه بین اجزا معمولاً لایه‌ای است (لایه زیرین مشترک است) اما در مهندسی مبتنی بر مولفه رابطه بین اجزا بیشتر به صورت موازی خواهد بود.

۶. در صورت استفاده از نرم‌افزار به عنوان سرویس، نیازی به راه‌اندازی بر روی سیستم هدف نیست؛ چرا که نرم‌افزار روی یک سیستم cloud اجرا می‌شود. به همین دلیل مدیریت releaseها آسان‌تر خواهندبود؛ چرا که نسخه‌ی نهایی همیشه در دسترس است و احتیاجی به به‌روزرسانی توسط کاربر نیست.

[پاسخ این قسمت از سوال با در نظر گرفتن متدولوژی12factor App (12factor.net) انجام شده‌است.]

می‌توان برای هر محیطی که نرم‌افزار قرار است روی آن اجرا شود، یک اطلاعات پیکربندی سیستم در نظر گرفت و همراه کد بر روی سیستم اجرا کرد؛ اما با افزایش محیط‌ها و تعداد سرویس‌ها، این روش مشکل‌ساز خواهدبود.

اصل سوم متدولوژی 12factor app بیان می‌کند که اطلاعات پیکربندی سیستم باید جدا از کد باشد و باید آن‌ها را در متغیرهای محیطی (Environment Variables) نگه‌داری کرد. با این روش، در صورت افزایش تعداد سرویس‌ها هم احتیاجی به‌روز کردن فایل پیکربندی برای هر محیط نیست.

چالش‌ها:

- باید برای هر محیط اطلاعات پیکربندی جداگانه تعریف شود.
- کمبود ابزار و متخصصان
- پیچیدگی کار با هر نوع اطلاعات پیکربندی

مزایا:

- باعث افزایش مقیاس‌پذیری (Scalability) سیستم می‌شود.
- افزایش سرعت توسعه
- عدم نیاز به به‌روزرسانی توسط کاربر
- افزایش امنیت
- در صورت پیکربندی با در نظر گرفتن محیطی که نرم‌افزار روی آن اجرا می‌شود، سازگاری نرم‌افزار افزایش پیدا می‌کند.
- نگه‌داری از سیستم راحت‌تر خواهدبود؛ چرا که می‌توان پیکربندی را برای یک محیط خاص تغییر داد.

۷. با پیشرفت پروژه، تخمین‌ها با قاطعیت بیشتری انجام شده و به واقعیت نزدیک‌تر خواهد بود. در ابتدا، همه‌چیز در نظر گرفته نمی‌شود و تخمین بسیار نادقیق است. با جلو رفتن علاوه بر کسب اطلاعات بیشتر در مورد نیازمندی‌ها و افزایش داده‌های موجود، مهارت و تجربه ذی‌النفعان نیز بیشتر شده که منجر به تخمین‌های بهتر می‌شود.

۸.

چون معماری لایه‌ای است پس می‌توان بخش‌های logic برنامه را جدا از بخش‌های دیگر (مانند database) توسعه‌داد؛ چرا که هر لایه تنها از لایه‌ی زیرین خود سرویس می گیرد و این سرویس‌گرفتن با یک interface ساده انجام می‌شود و جزئیات هر لایه به لایه‌ی دیگر مربوط نیست.

با جدا کردن لایه‌ها قابلیت تغییر در هر لایه بدون لازمه تغییر در لایه‌ی دیگر به‌وجود می‌آید. پس قابلیت نگه‌داری از سیستم (Maintainability) افزایش می‌یابد؛