"به نام خداوند بخشنده و مهربان"

آزمون پایان ترم تست نرم افزار - تیر ماه 1400 - آریا وارسته نژاد.

1)

الف) مدل RIPR شامل اصول زیر می‌باشد که هر کدام را به صورت مختصر توضیح میدهیم.

Reachability : ⟵⟶ Prefix values ، محل هایی که برنامه ما دچار احتمال خطا دارد بایستی به صورت دستی قرار گیرند.

Infection : ⟵⟶ test case ، بایستی حالت برنامه به وضعیت ارور منجر بشود.

Propagation : ⟵⟶ Post fix ، ارور تولید شده در قسمت قبل بایستی تا یک جای منتشر شود که بتوان از خروجی متوجه شد.

Revealability : ⟵⟶ Expected Results ، باید بخش هایی را چک کرد که پس از ارور دستخوش خطا شده اند.

ب) Testability : درجه‌ای است که یک مصنوع نرم افزاری خواه یک سند نیازمندی بوده یا سیستم نرم افزاری یا مدل و سند طراحی و ... از تست کردن پشتیبانی می‌کند. هر میزان قابلیت تست یک مصنوع نرم افزاری بالاتر باشد، کشف خطا ها در آن ساده‌تر است.

ج) TestCase : مجموعه مشخص سازی و تعریف ورودی های تست، شرایط اجرای تست، روند اجرای یک تست و نتایج مورد انتظار که جهت برآورده کردن یک هدف خاص می‌باشد، تست کیس گفته می‌شود.

2) ~~Metamorphic Testing~~ : در یک سری موارد تست کردن حالت های دیگری از مصنوع نرم‌افزاری ای که مایل به تست آن هستیم راحت تر و بهینه تر است. مثالی که در کلاس مطرح کردید این بود که فرض کنید یک تابع داریم که مقدار sin را محاسبه میکند. برای تست کردن درستی عملکرد آن بهتر است که آن را در این قالب تست کنیم: $\sin^2+\cos^2=1$. به این مدل از تست کردن Metamorphic Testing می‌گوییم.

اصلاح می‌کنم که این مورد ... است: ← Mutation Testing

3) Class Integration Test Order : بعد از این که برای هر کامپوننتی تست یونیت انجام شد، می‌خواهیم آنها را در کنار یک دیگر قرار داده و [illegible] یکپارچه تست کنیم. دو چالش اساسی داریم: 1) ناقص بودن

2) ترتیب تست:

هدف پیدا کردن ترتیبی برای تست کردن است که کمترین هزینه را داشته باشد. ← این عمل را با رسم گراف وابستگی برای کامپوننت ها انجام میدهیم ← چالش مهم در این امر وجود داشتن حلقه در وابستگی های بین کلاس ها است.

4) ~~Mutation Testing~~ : یک روش دیگری مورد برای تست نرم افزار است که راه حل های موثری برای test oracle Problem و test case generation Problem ارائه دهد.

اصلاح می‌کنم این مورد درست ← Metamorphic Testing

2)

الف) متد equals در زیر کلاس ColorPoint با Superclass خود ناسازگاری دارد.
من نتوانستم پیشنهادی برای تغییر ارائه کنم.

ب) همه‌ی آزمون‌ها عیب را اجرا می‌کنند.

ج) دوشنی از جنس ColorPoint بسازیم و آن‌ها را با متدهای equals شان evaluate کنیم. در این حالت عیب مورد اجرا قرار می‌گیرد اما خروجی کلاس‌ها یکسان هستند و هیچ ارور نداریم.

د) امکان پذیر نیست

ه)

3) مزایای Mdtd : میتوان به صورت مستقیم طبق معیار پوششی که داریم ازتزاع را انجام بدهیم.
ساخت خودکار تست‌ها با استفاده از Mdtd راحت‌تر می‌شود.
تعداد تست کیس های لازم خیلی کم می‌شود ← زمان لازم برای اجرای تست کاهش قابل توجهی پیدا می‌کند.
← امکان اجرای همزمانی و در نتیجه افزایش سرعت را میدهد
تعلیک وظایف را تسهیل می‌کند.
برای کد تست نمی نویسیم بلکه برای عملکرد تست می نویسیم

مقایسه:

آزمون مدل را نیز تست را به مجموعه‌ای از وظایف کوچک تقسیم کرده که موجب ساده سازی تولید تست می شوند و همین امر می تواند موجب ایجاد یک سطح انتزاع بالاتر برای طراحی تست گردد و سایر مزایایی که در صفحه قبل برای ها اشاره کردیم و آزمون مبتنی بر مدل یک تکنیک آزمون جعبه سیاه است که از اطلاعات موجود در مدل به عنوان مبنایی برای تولید تست استفاده می‌کند.

در بسیاری از موارد آزمون مبتنی بر مدل از دیاگرام های حالت UML استفاده می کند.

روش MBT به ۵ مرحله نیاز دارد

1) تحلیل مدل رفتاری موجود برای نرم افزار یا ایجاد آن مدل

2) پیمایش مدل رفتاری و مشخص سازی ورودی هایی که موجب تغییر حالت نرم افزار می شوند

3) بازبینی مدل رفتاری و یادداشت خروجی های مورد انتظار

4) اجرای موارد آزمون

5) مقایسه نتایج واقعی و مورد انتظار و اتخاذ تصمیم اصلاحی در صورت نیاز

⇐ روش مبتنی بر مدل به کشف خطاهای موجود در رفتار برنامه کمک کرده و هنگام اجرای برنامه های مبتنی بر رویداد می تواند بسیار کارآمد باشد.

4) هرچه معیار پوشش دقیق‌تر باشد آن را به صورت پررنگ‌تر داشته باشد در مقایسه در جایگاه بهتری قرار می‌گیرد:

1) متادیرتست به صورت بهینه تولید شوند.

2) نتایج تست بتواند بیشترین تعداد خطای ممکن را آشکار کند

3) بتوان به راحتی نیازمندی‌های تست را به صورت خودکار بدست آورد

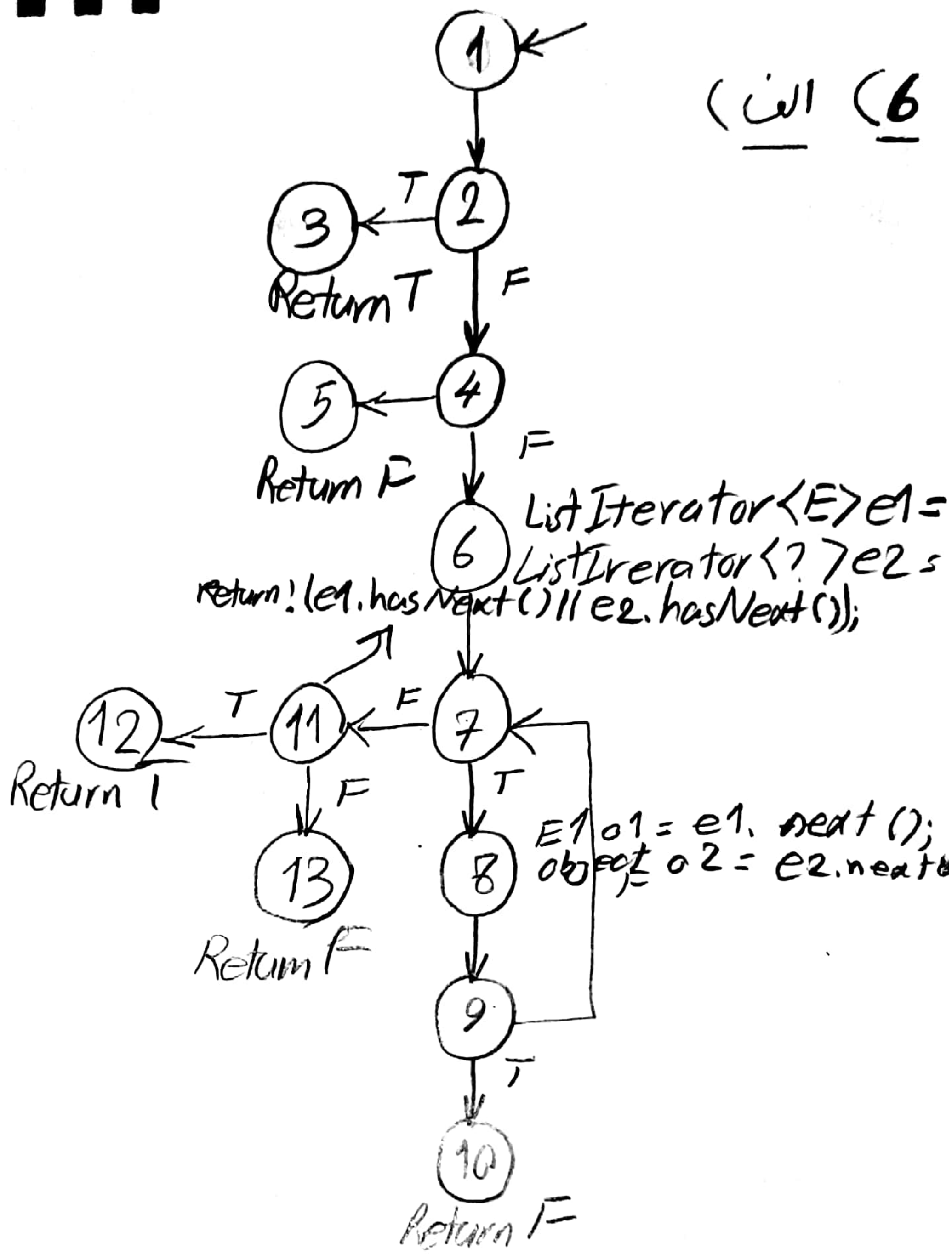
- مقایسه معیارها با یک مفهوم در برداشتن (Subsumption)

یک معیار آزمون C1 و C2 را در برداشته اگر و فقط اگر هر مجموعه‌ای از موارد آزمونی که معیار C1 را برآورده می‌کند، معیار C2 را هم برآورده کند.

توجه ← این مطلب باید برای همه مجموعه از موارد آزمونی برقرار باشد.

برای نمونه اگر یک مجموعه آزمونی همه انشعاب‌های برنامه‌ای را پوشش داده باشد (معیار انشعاب را برآورده کرده باشد) آنگاه این مجموعه آزمونی به طور تضمین شده همه عبارت‌های برنامه را نیز پوشش داده است.

6) الف)



ب) جهت تأمین پوشش گره (Node Coverage) باید مسیرهای داشته باشیم که روی هم دیگر شامل همه گره‌ها بشوند.

1-2-3 / 1-2-4-5 / 1-2-4-6-7-11-12 /
1-2-4-6-7-11-13 / 1-2-4-6-7-8-9-10

1 2 3 / 1 2 4 5 آزمون
1 2 4 6 7 11 13 /
1 2 4 6 7 8 9 7 8 9 10 /
1 2 4 6 7 8 9 7 11 12 /

ج) نیازمندی

123 / 124 / 1245 / 1246 / 467
6 7 11 / 7 8 9 / 6 7 8 / 9 7 11
9 7 8 / 8 9 7 / 7 11 12 / 7 11 13 / 7 9 10

الف) 7

RACC	g	f	C	P_C	P_f	P_g
1	F	F	F			
2	F	F	T	T	T	
3	F	T	F	T		
4	F	T	T		T	T
5	T	F	F		T	T
6	T	F	T	T		
7	T	T	F	T	T	
8	T	T	T			

RACC

P_C : (5,6) - (3,4) P_f : (7,5) - (4,2)

P_g : (7,3) - (2,6)

Min : (3,5,6,7)

GACC

P_g : (3,2) - (7,6)

P_f : (5,2) - (7,4)

P_C : (4,6) - (5,3)

ب) برآورده نمی شد

باید حالت های 2,4 حذف شود.

۸) BDD از TDD مشتق شده است.

TDD توسعه دهندگان نرم افزار را بر نوشتن تست پیش از نوشتن کد متمرکز کرده است. حال ممکن است که نرم افزاری داشته باشیم که خیلی خوب مورد تست قرار گرفته اما با درخواست های مشتری همخوان نداشته باشد. به همین علت BDD تعریف شد تا تعریف کردن User Story ها را بهبود ببخشد و تبدیل کردن User Story ها به نیازمندی ها را ساده تر کنیم. ← هدف BDD این است که مشتری را قادر سازیم که بتواند نیازمندی های خود را بدون گفتمان فنی برما بیان کند.

در TDD هدف اصلی تست واحد است و در BDD نیازمندی های مشتری است.