مروری بر روش توسعه مبتنی بر تست در مهندسی نرم افزار

سیدسلمان معراجی، دانشجوی دکتری و مدرس دانشگاه، دانشگاه آزاد واحد تهران جنوب
s.meraji1986@gmail.com
دکترمیرعلی سیدی، استادیار، دانشگاه آزاد واحد تهران جنوب
ma_seyyedi@azad.ac.ir

چکیده

توسعه مبتنی برتست[1]، یک روش تست پیش از توسعه و حتی برخی آن را یک روش توسعه نرم افزار می دانند. بطوریکه قسمت های کوچکی که به آن کد تست گفته می شود، برای توسعه واحد[2] های برنامه استفاده می شوند. نوشتن کدهای تست نرم افزار قبل از کدهای محصول نرم افزاری باعث اثرات مثبتی در فرایند توسعه و همچنین سایر فرایند های مرتبط با آن می شود. این روش بیشتر در متدولوژی های چابک[3] توسعه نرم افزار کاربرد دارد و در چرخه های تولید نرم افزار روش نسبتا جدیدی محسوب می شود. از این رو، روش توسعه مبتنی بر تست با روش های سنتی تست و توسعه نرم افزار بطور کامل متفاوت است. در این مقاله مروری در ابتدا به معرفی فازهای مختلف و نحوه اجرای این روش پرداخته شده است. سپس از آنجایی که می توان این روش را یک فرایند توسعه در نظر گرفت، مقایسه ای بین روش توسعه مبتنی بر تست و روش های سنتی چرخه حیات توسعه نرم افزار[4] پرداخته می شود. در ادامه همچنین به بررسی 2 مورد توسعه از این روش پرداخته شده است. همین طور به بررسی مشکلات و برخی نارسائی های استفاده از روش توسعه مبتنی برتست و راه حل هایی برای این نارسایی ها پرداخته شده است. در نهایت تاثیرات استفاده از روش توسعه مبتنی برتست بر روی صفات کیفی درقالب یک جدول ارزیابی می شود که نشان دهنده توانایی قابل ملاحظه این روش در بهبود برخی از صفات کیفی نرم افزار است.

کلیدواژه‌ها: آزمون نرم افزار، آزمایه، توسعه مبتنی بر تست، کد منبع، صفات کیفی، فرایند توسعه

[1] Test Driven Development
[2] Unit
[3] Agile
[4] Software Development Life Cycle (SDLC)

1- مقدمه و بیان مسئله

توسعه مبتنی بر تست که به آن کدنویسی تست اولیه یا طراحی مبتنی برتست نیز گفته می شود، روشی است که در آن برنامه نویس آموخته است که ابتدا قبل از نوشتن کد عملیاتی یک قسمت از نرم افزار، آزمایه[1] مربوط به خرابی[2] آن قسمت از کد را بنویسد[Beck2000] این روش توسعه بطورکلی از روش های سنتی تست در مرحله نهایی که معمولا در توسعه نرم افزار استفاده می شود، متفاوت است. در روش های سنتی معمولا کد عملیاتی مطابق با ویژگی و مشخصات طراحی نوشته می شود و تنها پس از آنکه بیشتر کد عملیاتی نوشته شد، یک تکه کد تست نوشته می شود تا کدعملیاتی را تست کند.

کنت بک[3] که خیلی ها او را خالق این روش می شناسند، از نام توسعه مبتنی برتست درکتابش، درسال 1999نام برد[Beck2000]. بک، از این روش در اوایل دهه 90 زمانی که Smalltalk را توسعه می داد، استفاده نمود. اما همانطور که بک نیز اشاره کرده بود، ایده نوشتن آزمایه قبل از کد اجرایی از او شروع نشده بود. این در حالی است که در حال حاضر روش تست پیش از کد نهایی به راحتی توسط اکثر محیط های توسعه یکپارچه نرم افزار[4] پشتیبانی می شود بطوریکه برنامه نویس می تواند آزمایه و کد اجرایی را با هم بنویسد و باز خورد مناسب را بسرعت دریافت کند[Beck2001]. همچنین در[Larman2003] بیان شده است که اولین مرجع بکارگیری روش توسعه مبتنی بر تست در اوایل دهه 60 میلادی در پروژه مرکوری ناسا بود، که از این روش برای توسعه نرم افزار یک سفینه فضایی استفاده شد.

توسعه مبتنی بر تست براساس اصول و قواعد مشخصی توصیف شده است[Beck2003] [Astels2003] . اما برای اجرا، راهنمای دقیقی ندارد والبته برخی از دیدگاه های آن بیشتر به آرمان و آرزو شبیه بود نظیر این که "هیچ وقت یک خط از کد اجرایی برنامه را ننویسید مگر آنکه آزمایه آن شکست خورده باشد[Beck2001].چراکه همانطور که در ادامه هم بیان خواهد شد عملا امکان اینکه پوشش کد به 100% برسد غیر ممکن است.

ویلیام ویک[5] در [Wake2001] از مفهوم رنگ های چراغ راهنمایی و رانندگی در کنترل ترافیک برای فهم بهتر مفهوم توسعه مبتنی بر تست استفاده کرده است. اگر همان سه رنگ سبز و زرد و قرمز را در نظر بگیریم. شروع به نوشتن کد یک آزمایه همان چراغ سبز است. زمانی که آزمایه در حین کامپایل شدن است چراغ زرد را می گذراند و درکامپایل شکست می خورد زیرا که هیچ کد عملیاتی دردسترس نیست و مرحله ای است که برنامه نویس در بدنه اصلی برنامه هیچ کدی برای گذراندن آزمایه ننوشته است. این وضعیت حالت چراغ قرمز را تداعی می کند. زمانی که کد اجرایی نوشته شد و آزمایه با موفقیت گذرانده شد، توسعه دهنده دو مرتبه به چراغ سبز بازگردانده می شود. وضعیتی را که ویک معرفی کرد را می توان بصورت خلاصه با نام های قرمز[6]،سبز[7]و بازسازی کد[8] نام گذاری کرد.[Beck2003] . نور قرمز، نشان دهنده آزمایه شکست خورده

[1] Test Case
[2] Failure
[3] Kent Beck
[4] Integrated Development Environment (IDE)
[5] William Wake
[6] Red
[7] Green
[8] Refactoring

است، نور سبز، نشان دهنده نتیجه نهایی نوشتن کمترین میزان کد عملیاتی برای با موفقیت گذراندن آزمایه است و بازسازی کد، برای حذف کردن هر کدی که دوبار نوشته شده است و سرو سامان دادن به کد اجرایی است.

توسعه مبتنی برتست یک روش پایه در متدولوژی XP برای توسعه نرم افزار است.[Beck2000] اما می توان آن را در جاهای دیگر نیز استفاده نمود. بعنوان نمونه فعالان حوزه متدولوژی های چابک در سال 2010 با ارائه گزارشی در مورد اعتبارسنجی نرم افزار، در حدود 53% از 293 نفر روش توسعه مبتنی بر تست را توصیه کرده بودند [Ambler2010] [Beck2010] همچنین در [West2010] گزارش شده است که در جریان یک نظرسنجی عمومی تر 3.4% از 1298 نفر فعالان حوزه نرم افزار به استفاده از روش توسعه مبتنی بر تست بعنوان یک روش توسعه نرم افزار رای داده اند. علت درصد پایین روش توسعه مبتنی برتست این بود که در نظرسنجی مذکور سوال این بود که "لطفا نام متدولوژی ای که بیشتر برای فرایند توسعه از آن استفاده می کنید را نام ببرید" و جالب اینجاست که توسعه مبتنی بر تست به عنوان یک متدولوژی درکنارمتدولوژی هایی نظیر اسکرام[1]،XP ، روش آبشاری و... ذکر شده بود. این در حالی است که توسعه مبتنی برتست یک متدولوژی توسعه نرم افزار نیست بلکه روشی است که می تواند به همراه سایر متدولوژی ها مطرح شود.

در بخش دوم، مروری بر کارهای گذشته انجام شده در حوزه توسعه مبتنی بر تست انجام می شود سپس بخش سوم به توصیف چرخه روش توسعه مبتنی برتست پرداخته خواهد شد. در بخش چهارم به بررسی تفاوت های این روش با روش هایی سنتی توسعه نرم افزار از قبیل روش آبشاری[2] و مدل V[3] و روش تکرار[4] وهمچنین روش های چابک پرداخته می شود. در بخش پنجم به بیان دو توسعه که به اجرایی کردن هرچه بهتر این روش کمک می کند، پرداخته می شود. در بخش ششم به برخی مشکلات استفاده از توسعه مبتنی برتست در پروژه های واقعی و در صنعت و بیان برخی راه حل های پیشنهادی در این حوزه پرداخته می شود و در نهایت به بررسی اثرات مختلف روش توسعه مبتنی برتست در حوزه های مختلف نرم افزار و صفات کیفی بیان نتایج حاصل از مقالات در این حوزه پرداخته خواهد شد.

2- پیشینه تحقیق

روش توسعه مبتنی برتست، در تعدادی از مطالعات پیشین مورد بررسی قرار گرفته است. در برخی از مقالات بررسی شده ازمطالعه نتایج می توان دریافت که مزایا و بهبود هایی با استفاده از این روش بدست آمده است در حالیکه در برخی از مطالعات دیگر، بهبود ها مبهم و یا ناقص بیان شده اند. در ادامه به بررسی این موراد می پردازیم.

کوماردر [Kumar2013] یک مطالعه مقایسه ای از روش توسعه مبتنی بر تست با روش های سنتی انجام داده است و به بیان مزایا و معایب هریک از این روش ها در مقاله خود پرداخته است. همچنین در [Pancur2011] نیز به بررسی روش توسعه مبتنی بر تست و مقایسه آن با روش تست تکراری درآخرین

[1] Scrum
[2] Waterfall Model
[3] V Model
[4] Iterative Model

مرحله[1] با تاکید بر سه پارمتر میزان بهره وری[2]، صفات کیفی داخلی[3] وخارجی[4] و کیفیت کد[5] مقایسه کرده است. جنزن نیز در مقاله خود [Janzen2006] به بررسی میزان تاثیر این روش در فرایند توسعه نرم افزار پرداخته است و نتایج را به این شکل بیان کرده است که این روش بر میزان بهره وری برنامه نویسان تاثیر ثبت دارد. با این حال برای برخی صفات کیفی داخلی مشکلاتی را فراهم می کند.

مقالاتی به چاپ رسیده اند که درباره مزایای این روش مواردی را بیان کرده اند از جمله [Cauevic2012] و[Consulting] که نویسندگاه آن به وضوح به افزایش میزان بهره وری اشاره کرده اند. ویا در مقالات [Edwards2004] و[Erdogmus2005] به بهبود کیفیت اشاره شده است. از دیگر مزایای روش توسعه مبتنی برتست افزایش میزان پوشش تست[6] است که در [Janzen2008] و[Kaufmann2003] به آن پرداخته شده است و در نهایت افزایش اعتماد به نفس در توسعه دهندگان که در [Edwards2004] و[Janzen2008] بیان شده است.

همین طور در [Bhat2006] اقدام به ارزیابی کیفیت در آزمایه ها درهردو روش تست در ابتدا یعنی توسعه مبتنی برتست و تست در انتها بصورت جداگانه انجام داده اند و نتایج مطالعه آنها به این ترتیب است که کیفیت آزمایه ها در هر دو روش تقریبا یکسان است. هر چند که با اطمینان بیان شده است که کیفیت کد اصلی برنامه در روش توسعه مبتنی بر تست بسیار بهتر است.

در [Amlani2013] یک مقایسه جامع میان نیز میان مدل های مختلف چرخه حیات توسعه نرم افزار از قبیل مدل های آبشاری، مدل تکراری، مدل V و مدل حلقوی[7] با توجه به معیارهای مختلف با روش توسعه مبتنی بر تست صورت گرفته که نتایج آن نیز قابل توجه است.

همچنین در برخی از مقالات مروری نیز پس از مقایسه روش با سایر روش ها، مزایای روش مبتنی برتست مبهم عنوان شده است بعنوان مثال کولانوس در [Kollanus2010] حدود 40 مورد کار انجام شده در حوزه توسعه مبتنی برتست را مقایسه کرده و در نهایت هم نتایج زیر را در مورد 3 پارمتر کیفیت خارجی، کیفیت کد داخلی و میزان بهره وری منتشر کرده است که عبارتست از:

1.صفات کیفی خارجی به میزان اندکی بهبود یافته اند.

2.صفات کیفی داخلی نیز به میزان بسیار اندکی بهبود یافته اند.

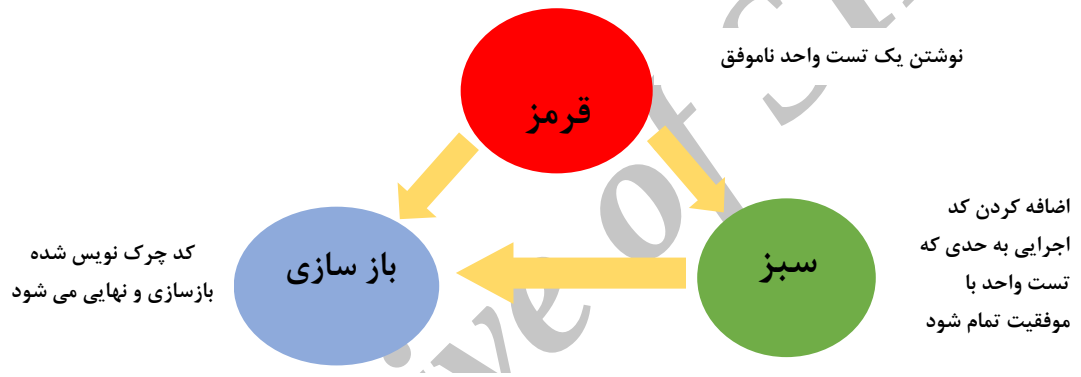
3. میزان بهره وری کاهش قابل توجه داشته است.

ویا بعنوان مثال در[Aniche2010] بیان شده بطور متوسط 33 درصد از برنامه نویسان قابلیت تطبیق با این روش را پیدا نخواهند کرد و همچنین بسیاری از برنامه نویسان ترجیح می دهند کد های ساده و آشنا را ابتدا بنویسند که این خود تخطی از قانون مهم این روش یعنی نوشتن آزمایه در ابتدای کار است.

3- چرخه حیات سبک توسعه مبتنی بر تست

[1] Iterative Test Last
[2] Productibility
[3] Internal Quality
[4] External Quality
[5] Code Quality
[6] Test Coverage
[7] Spiral Model

روش توسعه مبتنی برتست بطور کامل فرایند سنتی توسعه نرم افزار را برعکس کرده است. بجای نوشتن کد اجرایی قبل از کد تست، در این روش کد تست قبل از کد اجرایی برنامه نوشته می شود. توسعه مبتنی بر تست از 2 گام اساسی تشکیل شده است. ابتدا نوشتن کد تست و آزمایه ها و سپس بازسازی کردن کد اجرایی. پس اگر سعی دارید یک ویژگی[1] جدید به محصول اضافه کنید، اولین سوالی که بوجود می آید این است که طراحی حال حاضر، آیا اجازه پیاده سازی این ویژگی را می دهد یا خیر؟ اگر بله، پس کار خود را با نوشتن کد تست آزمایه شروع می کنیم. اگر پاسخ خیر است شروع به بازسازی کد به اندازه کافی در ابتدا می کنیم. بعنوان یک نتیجه می توان گفت کیفیت طراحی افزایش می یابد و طراحی کد قابل توسعه[2] و قابلیت نگهداری[3] بهتری پیدا می کند. توسعه مبتنی بر تست تکرار مکرر چرخه اضافه کردن آزمایه ای است که قرار است شکست بخورد و سپس گذراندن آن آزمایه با موفقیت و باز سازی مجدد کد است. یکی از نکات مهم زمان پیاده سازی این روش آن است که واحد های کد[4] را کوچک نگه داریم. یک واحد کد در برنامه نویسی شی گرا شامل کلاس ها و در برنامه نویسی پیمانه ای شامل رویه ها و توابع است.کوچک کردن واحدهای کد نه تنها تلاش[5] برای رفع ایرادات[6] را کاهش می دهد بلکه باعث افزایش میزان قابلیت فهم[7] کد می شود.



شکل 1: چرخه حیات روش توسعه مبتنی بر تست

درشکل 1 چرخه حیات روش توسعه مبتنی بر تست را ملاحظه می کنید.چرخه حیات این روش به ترتیب شامل مراحل زیر است:

1.اضافه کردن تست: هر ویژگی جدید نرم افزار با نوشتن آزمایه یا کد تست در ابتدا آغاز می شود. این تست درواقع برای ویژگی ای که هنوز پیاده سازی نشده است، نوشته می شود. پس بنابراین باید با نتیجه خطا همراه باشد. اگر تست نتیجه خطا به همراه نداشته باشد، یا در طراحی تست ایراد وجود دارد و یا از قبل آن ویژگی مد نظر، در کد اجرایی برنامه نوشته شده است.

2.اجرای تمامی تست ها و مشاهده اینکه آزمایه جدید همراه با خطا باشد: این قسمت باعث می شود اطمینان حاصل کنیم که بازدارندگی تست کار می کند و یک قطعه کد جدید بدون آنکه تست شود به کد اصلی اضافه نمی گردد.

---

[1] Feature
[2] Scalable
[3] Maintainability
[4] Code Units
[5] Effort
[6] Debugging
[7] Understandability

3.نوشتن مقداری کد: مقداری کد می نویسیم تا آزمایه با موفقیت گذرانده شود. این کد در این مرحله نیاز نیست که خیلی بهینه باشد.

4.اجرای تست: تمام تست ها را اجرا می کنیم. هم آزمایه های قبلی که با موفقیت تمام شده اند و هم آزمایه آخرکه قبل از مرحله 3 با خط روبرو شده بود. حال اگر بعد از مرحله 3 با موفقیت از این مرحله عبور کند، می توان به مرحله بعدی رفت.

5.بازسازی کد: پاک سازی کد در این مرحله صورت می گیرد. بصورت خیلی ساده این مرحله عبارتست از قراردادن کد ها در جایگاه منطقی شان یا از بین بردن کدهایی که چند بار تکرار شده اند. بطور کلی هر فرایندی اینجا انجام شود، با هدف بهبود در کد نوشته شده در مرحله 3 است.

6.تکرار: تکرار کل چرخه برای اینکه عملکرد [1]بعدی را بگنجانیم.

4- مقایسه بین توسعه مبتنی بر تست و سایر روشهای چرخه حیات توسعه نرم افزار

درروش های سنتی فرایند با نوشتن کد اجرایی برنامه آغاز می شود و سپس کد تست یا آزمایه نوشته شده و در نهایت آزمایه ها اجرا می شوند. بنابراین این روش ها با عنوان تست آخر[2]شناخته می شوند. دقیقا برعکس این موضوع در توسعه مبتنی برتست اتفاق می افتد، جایی که تست ها در ابتدا نوشته می شوند. برخی از مدل های چرخه حیات نرم افزار از قبیل مدل آبشاری، مدل V ، مدل تکرار و مدل حلقوی و یا متدولوژی های چابک را می توان به دودسته مدل های پیشگویانه[3] و مدل های تطابق پذیر[4] تقسیم بندی کرد. همه این تکنیک ها و مدل ها هر یک خصوصیات خاص خود را دارا می باشند اما همه آنها یک ویژگی مشترک دارند که همانا تست در مرحله آخر است.

مطابق با [Kumar2013] برخی از مشکلات این مدل ها به قرار زیر است:

1.تمام نیازمندی ها باید از قبل شناخته شده باشند.

2.یک سری کد های عملیاتی وجود دارند که هیچ گاه اجرا نمی شوند به این معنی که کیفیت کد پایین است.

3.هیچ تضمینی برای اینکه آزمایه نوشته شده عملکرد جدیدی از سیستم را تست کند، وجود ندارد.

توسعه مبتنی بر تست بعبارت دیگر مبتنی بریک متدولوژی ساده است و آن همانا تست واحد[5] و بازسازی. همانطور که قبلا بیان شد. البته لازم به ذکر است که روش توسعه مبتنی برتست برمبنای پیش طراحی برای فهم ساختار نرم افزار نیز نیست. روشی که توسعه مبتنی بر تست عمل می کند این است که نیازمندی ها به شکل موارد کاربری[6] به مجموعه ای ازسناریو ها تجزیه می شوند. برای هر سناریو اولین کاری که باید انجام شود این است که، تست واحدی نوشته شود که این سناریو را تست کند. مزایای نوشتن آزمایه در ابتدا این است که کمک می کند رفتار سیستم بهتر درک شود و همچنین به برخی از سوالات مرحله طراحی[7] نیز ممکن است پاسخ دهد.

[1] Functionality
[2] Last Test
[3] Predictive
[4] Adaptive
[5] Unit Test
[6] Use Case
[7] Design

در ادامه در مقایسه ای از روش توسعه مبتنی بر تست با متدولوژی های پیشگویانه و تطابق پذیر از لحاظ کیفی وکمی انجام داده شده است.در متدولوژی های پیشگویانه، فرایند توسعه نرم افزار به صورت گام های پیاپی و پشت سر هم انجام می شود ازقبیل مدل V و مدل آبشاری و در متدولوژی های تطابق پذیر همانند روش های چابک بیشتر برروی کارتیمی تمرکز وجود دارد تا برروی فرایند.

در[Bajaj2015] ، 24 پارامتر کیفی مختلف اقدام به مقایسه روش توسعه مبتنی بر تست با این دو نوع مدل شده است. معیارهای کیفی مقایسه عبارتند از:1-میزان اهمیت برنامه ریزی پیش از توسعه 2-فهم نیاز های کسب و کار 3-دخالت مشتری 4-نیاز به آماده سازی تمام نیازمندی ها قبل از شروع به توسعه 5- پیچیدگی 6-محصول قابل تغییر 7-سبک توسعه 8- آموزش و مستند سازی 9-میزان کار آمدی 10-انعطاف پذیری 11-قابلیت نگهداری 12-سایز پروژه 13-چرخه انتشار 14-منابع مورد نیاز 15-قابلیت استفاده مجدد 16-هزینه کار مجدد 17-میزان درگیر بودن ریسک با پروژه 18-نرخ موفقیت 19-سایز تیم 20-تست کردن محصول 21-تنوع در نیازمندی ها 22- در دسترس پذیری محصول آماده به کار 23-نکات مثبت 24-نکات منفی. سپس به مقایسه این روش با مدل های پیشگویانه پرداخته شده است که در بسیاری از پارامتر ها این روش وضعیت بهتری داردهمچنین در [Bajaj2015] مقایسه ای نیز بین روش توسعه مبتنی بر تست با روش های تطابق پذیر توسعه نرم افزار انجام شده است و نتایج در بسیاری از پارامتر ها همانند قبل حالت بود. از نقطه نظر مقایسه کیفی روش توسعه مبتنی بر تست با سایر روش ها می توان به این نتیجه رسید که این روش پیچیدگی پایین، هزینه دوباره کاری پایین و قابلیت پشتیبانی از کارها و تغییرات آینده را به خوبی دارا می باشد.

علاوه بر مقایسه کیفی در[Bajaj2015] یک مقایسه کمی هم در 16پارامتر با سایر متدولوژی ها صورت گرفته است.

5- توسعه های روش توسعه مبتنی برتست

برای روش توسعه مبتنی بر تست مدل های تکاملی جهت بهبود هرچه بیشتر و انطباق بیشتر آن با چرخه حیات نرم افزار ارائه شده است که در ادامه 2 مورد از این موارد توسعه را بررسی می کنیم.

5-1-توسعه مبتنی بر رفتار[1]

توسعه مبتنی بر رفتار برای اولین بار در [North2006] توسط نورث معرفی شد. اولین دلیل برای تغییر نام از توسعه مبتنی بر تست به توسعه مبتنی بر رفتار، بحث آن است که اینکه توسعه مبتنی برتست یک روش تست است یا یا یک روش طراحی را حل کنیم. ازتلز در[Astels2006] می گوید " توسعه مبتنی بر رفتار همان همان کاری است که شما انجام داده اید اگر روش توسعه مبتنی بر تست را بخوبی پیاده سازی کرده باشید."

با تغییر نگرش و تمرکز توسعه دهنده به سمت رفتار کد، اثبات شده است که توسعه دهنده مجموعه فکری خود را از تست به سمت طراحی سوق دهد. یعنی بر روی بحث طراحی و معماری در حین توسعه، در این روش تمرکز بیشتری داشته باشد. برای توضیح بهتر توسعه مبتنی بر رفتار یک چرخه حیات مبتنی بر روش

[1] Behavier Driven Developement

توسعه مبتنی بر تست در [Rimmer2010] بیان شده که نحوه انطباق این را نشان می دهد. مراحل چرخه حیات بطور خلاصه عبارتند از:

1.توسعه دهنده اقدام به نوشتن تست واحد در کنار کد اجرایی در محیط هایی نظیر NUnit وJUnit می کند.

2.همین طور که بدنه کد تست افزایش می یابد توسعه دهندگان از حس اعتماد بنفس بالاتر خود در این کار لذت بیشتری می برند.

3.در برخی از اوقات توسعه دهنده به این بینش می رسد که نوشتن مکرر تست پیش از کد باعث می شود تا توسعه دهندگاه به طور اتوماتیک یادبگیرند بصورت بهینه کد اجرایی لازم و صحیح بنویسند.

4.توسعه دهندگان همچنین توجه می کنند که وقتی پس از گذشت زمانی به قطعه کدی که نوشته اند باز می گردندکه مدتی آن را ندیده اند، تست همانند یک مستند به فهم کد برای توسعه دهنده کمک می کند.

5.یک نقطه الهام بخش اتفاق می افتد زمانی که توسعه دهندگان احساس می کنند، نوشتن تست به این روش به آنها کمک می کند. API مربوط به کد خود را کشف کنند. در اینجا می توان گفت توسعه مبتنی بر تست به یک فرایند توسعه تبدیل شده است.

6. خبرگان روش توسعه مبتنی برتست به این نتیجه می رسند که این روش درصدد توصیف رفتار است به جای آنکه یک روش تست باشد.

7.رفتار، درباره تعاملات بین مولفه[1] ها در یک سیستم است. پس گاهی تقلید کردن شروعی است که باعث پیشرفت در توسعه مبتنی بر تست می شود.

همچنین در پایان در [Rimmer2010] بیان شده است که اکثر توسعه دهندگان با کمی کمک به مرحله 4 می رسند و اما تعداد کمی هستند که می توانند از مرحله 4 فراتر روند. در اینجا به هیچ نتایج آزمایشگاهی و دقیقی اشاره نشده است چرا که معمولا براساس تجربیات و مشاهدات می توان این نتایج را گرفت.

نورث در [North2006] یک چهارچوب نرم افزاری جاوا[2] به نام JBehave را توسعه داده است که هر مرجعی که برای نوشتن تست وجود دارد را حذف کرده کرده و کلا ساختار زبان برنامه نویسی را تغییر داده که بر روی رفتار تمرکز بیشتری انجام گیرد تا بر روی زبان. بعنوان مثال بجای فراخوانی تابع assertion در زبان برنامه نویسی جاوا برای جایگذاری تست در این نرم افزار از تابعی بنام ensureThat استفاده می کند.و به این ترتیب فرمت یک فراخوانی عبارتند از:

ensureThat (actual,equalTo(expected));

برخی از ویژگی های توسعه مبتنی بر رفتار عبارتند از

1.استفاده از کلمه تست نباید داخل نام تستی که نوشته می شود، قرار گیرد.

2.نام های متدهای تست باید جمله باشند و این جملات با کلمه should به معنی باید شروع شوند بعنوان مثال:

ShouldTransferBalanceFromAccountWithSufficientFunds

[1] Component
[2] Java

3.تست ها نباید رابطه شان با کلاس ها یک به یک باشد و باید بر روی رفتار کلاس ها تمرکز کنند. این زمانی موثر است که کلاس ها شروع به رشد می کنند و تغییر ساختار آنها بسیار ساده تر می شود.

2-5-توسعه مبتنی بر تست پذیرش شده[1]

آزمایه هایی که برای توسعه استفاده می شوند آزمایه هایی هستند که نیازمندی های سطح پایینی را برآورده می کنند و هیچ راهی برای آنکه توسعه دهنده نیازمندی ها را بگنجاند، وجود ندارد. نتیجه این عمل طبق آن چیزی که در [Madeyski2010] بیان شده است. آن است که میزان وابستگی مولفه ها[2] بسیار کمتر از روش هایی تست در مرحله آخر است. ولی از طرفی تعداد تست های پذیرش[3]کاربران که با موفیت تمام شده نیز قابل توجه نیست. می توان گفت کیفیت کد بالاتر رفته ولی از طرفی نیاز مندی های سطح بالا کمتر برآورده شده اند.

برای اطمینان از اینکه کد همان عملکردی که کاربر می خواهد را پیاده کند برخی از مهندسین نرم افزار بر روی در جه بالاتری از تست تمرکز کرده اند و آن این است که به کاربران اجازه می دهند تست پذیرش را بنویسندو توسعه دهندگان از تست پذیرش کاربران استفاده کنند، تا ببینند آیا عملکرد لازم را نرم افزار پیاده می کند یا خیر؟ می توان چرخه توسعه مبتنی بر تست پذیرش شده را به 4 مرحله تقسیم کرد.

1.فازبحث[4]: در مرحله اول ذینفعان از جمله مشتری وکاربران نهایی معیارهای پذیرش یک ویژگی یا عملکرد[5] را مورد بحث و گفت و گو قرار می دهند. وبه یک نتیجه واحد در مورد پیاده سازی این سیاست می رسند.

2.فاز ترکیب کردن[6]: این فاز که مخلوط کردن نیز نامیده می شود. نتایج حاصل از فاز اول را به فرمت یکی از ابزارهای[7] این حوزه بشکل جدول[8] یا صفحات گستره[9] ایجاد می کنند. ابزار تست اتوماتیک می تواند به شکل استاندارد نیازمندی ها[10]را از این ابزارها دریافت کند. نمونه های این ابزار ها از قبیل FIT ،Fitnesse ، ConCordian وRobot می باشد.

3.فاز توسعه[11]: زمان پیاده سازی کد، توسعه دهندگان از روش توسعه مبتنی بر تست استفاده می کنند. آنها تست پذیرش را که از مرحله اول و دوم بدست آمده است را اجرا می کنند و منتظر شکست آن می شوند. حال ما یک آزمایه داریم که شکست خورده است زیرا کد مربوط به آن ویژگی در نرم افزار وجود ندارد. البته باید مطمئن شد که بدلیل درستی آزمایه شکست خورده است زیرا ممکن است اورهایی که در ابزارهای ذکر شده وجود دارد باعث شکست آزمایه شده باشند. توسعه دهندگان کد اجرایی را باید بگونه ای بنویسند که تست پذیرش با موفقیت گذرانده شود.

---

[1] Acceptance Test Driven Development (ATDD)
[2] Coupling
[3] Acceptance Test
[4] Discuss
[5] Functionality
[6] Distill
[7] Tool
[8] Table
[9] Spreadsheet
[10] Requirments
[11] Develop

4.فاز نسخه آزمایشی[1]: حال که تست با موفقیت گذرانده شده توسعه دهندگان به همراه ذینفعان اقدام به استفاده از نمونه آزمایشی می کنند تا اطمینان از این قضیه حاصل شود که عملکرد پیاده سازی شده به خوبی کار می کند و این چرخه به همین منوال ادامه می یابد

استفاده از توسعه مبتنی بر تست پذیرش شده توسعه دهندگان را مجبور به پیاده سازی نیازمندی های عملکردی نرم افزار با بررسی تست پذیرش می کنند.

6- مشکلات روش توسعه مبتنی بر تست و بیان راه حل ها

ابتدا در بخش 6-1 به توضیح انواع مشکلات در این روش براساس [Nanthaamornphong2015]به صورت کلی پرداخته می شود و سپس در 6-2 به توضیح و تبیین مشکلات حوزه تست و سپس در بخش 6-3 به توضیح مشکلات حوزه بازسازی کد پرداخته می شود.

6-1- تقسیم بندی مشکلات روش توسعه مبتنی برتست

بطور کلی روش توسعه مبتنی بر تست همانطور که بیان شد از 3 بخش اصلی تشکیل شده است نوشتن آزمایه، پیاده سازی کد و بازسازی کد. نتایج نشان دهنده آن است که با توجه به [Nanthaamornphong2015] سخت ترین قسمت کار نوشتن آزمایه است و بعد از آن پیاده سازی کد از مراحل مشکل روش توسعه مبتنی بر تست است. در ادامه به بررسی هریک از موارد مربوط به هر یک از این مراحل، اشاره می شود و انواع هریک رابررسی می کنیم.

6-2-مشکلات مربوط به تست کردن و راه حل های آن

بطور کلی می توان مشکلات بخش تست روش توسعه مبتنی برتست را به 6 بخش و راه حل های آن را به 5 بخش تقسیم کرد[Nanthaamornphong2015] که در ادامه هر یک را مورد بررسی قرار می دهیم.

1.سختی در نوشتن یک تست یا آزمایه: مشکل اصلی زمان نوشتن یک آزمایه، نوشتن یک آزمایه خوب است.اکثر توسعه دهندگان اعتقاد دارند نوشتن یک آزمایه خوب بسیار سخت تر ازپیاده سازی کد واقعی نرم افزار است.مخصوصابرای کد های تستی که با تغییر کد اصلی تغییر نمی کنند. علاوه براین در حین بازسازی بعضا توسعه دهندگان نیازبه تغییر کد را احساس می کنند و نیازبه تغییر در خود آزمایی ها نیز هست و این قضیه برای توسعه دهندگان مبتدی سخت تر نیز می شود.

2.برنامه های با پیچیدگی بالا: می توان این مشکل را خود به 3 بخش تقسیم بندی کرد: الگوریتم های پیچیده، محاسبات عددی و پردازش موازی. در نرم افزارهای علمی در اکثر اوقات کد شامل الگوریتم های پیچیده است که این پیچیدگی نیازمند تست های پیچیده است. بیشتر مشکلات با کد های پیچیده زمان اجراست ازقبیل بن بست[2]و اتصالات برنامه در حال اجرا است. بعلاوه برای محاسبات پیچیده نیز تست بسیار مساله مشکلی است زیرا توسعه دهندگان خود پاسخ درست را با تمام اعتماد بنفس نمی دانند. و در نهایت پردازش موازی، نوشتن آزمایه برای تست وضعیت اجرای همزمان در پردازش موازی کار بسیار دشواری است.

3.پوشش کد[3]: بصورت تئوری تست ها باید اکثر کد را پوشش دهند. اما عملا پوشش کد به 100% نمی رسد و همچنین این مشکل باز در نرم افزار های علمی با چندین بخش پویا، برآورده کردن پوشش حداکثری

[1] Demo
[2] Deadlock
[3] Code Coverage

کاربسیار دشواری است. همچنین می توان مشکل را به این ترتیب بیان کرد که آیا ازمایه نوشته شده همان چیزی را تست می کند که برای آن نوشته شده است ؟

4. کمبود روش ها، ابزارها و استانداردهای مهندسی نرم افزار: کسانی که اقدام به نوشتن آزمایه ها می کنند، نیازمند آن است که اصول مهندسی نرم افزار را بدانند. در حالی که بسیاری مثلا در مورد تست واحد اطلاعی ندارند. ازطرفی بعضا محدودیت در بکاربردن ابزارهای اتوماتیک سازی تست وجود دارد. که برای نوع خاصی ازنرم افزارها یا زبان های برنامه نویسی ساخته شده اند و همه نوع نرم افزاری را پشتیبانی نمی کنند. در مورد آخر اینکه توسعه دهنده قصد دارد آزمایه ای را برای مقدار خروجی بنویسد که نتیجه داشته باشد یا حداقل یک بازه جواب داشته باشد ولی فرض کنید زمانی را که هیچ استانداردی در آن حوزه وجود نداردکه بازه ای را برای جواب مشخص کند.

5. زمان بر بودن نوشتن آزمایه ها: زمان زیادی صرف نوشتن تست ها می شود، که بر روی هزینه و برنامه ریزی پروژه تاثیر گذاراست. دربرخی اوقات گزارش شده که کد تست از کد عملیاتی بیشتر شده است و کد تست از طرفی خود یک مشکل برای بهبود کد اصلی برنامه است زیرا مثلا یک تغییر در برنامه درحدود500 خط ازکد اصلی برنامه را در بر می گیرد، باعث تغییر 100 خط از کد تست می شود. واین باعث می شود توسعه دهندگان کمتر به سراغ تغییر در کد اصلی بروند. درواقع نوشتن آزمایه یک کار تمام وقت است خصوصا زمانی که مشتری از کاربر نهایی درخواست یک تغییر در نرم افزار آن هم در زمان کوتاه را می کند.

6.کد یا نیازمندی ها عوض شده باشد: این موضوع واضح است که اگرنیازمندی ها یا API ها تغییر کنند، تست نیزباید عوض شود.بعنوان مثال اگر الگوریتم عوض شود ممکن است تست دیگر معتبر نباشد، چرا که مثلا تابع تولید اعداد تصادفی عوض شده است.

حال به بررسی برخی راه حل ها در این حوزه می پردازیم.

1.استفاده ازابزار های تست مناسب: محیط های تست خوب و استفاده ازابزارهای تست اتوماتیک، تا حد زیادی نوشتن آزمایه ها را راحت می کنند.و همچنین استفاده ازاین ابزار ها بسیارباعث صرفه جویی در زمان می شوند و زمان بسیار کمتری نسبت به حالتی که آزمایه ها بصورت دستی نوشته شوند تلف می شود.

2.تجربه: نوشتن یک آزمایه و تست خوب نیازمند تجربه است بخصوص زمانی که اقدام به کد نویسی الگوریتم ها و کد های پیچیده می کنید. بعلاوه توسعه دهندگان حرفه ای می توانند با اعتماد بنفس بالا به تیم در حل مشکلات کمک کنند. دراین حوزه مجموعه ای ازمثال ها به همراه استفاده از بهترین روش ها[1]می تواند تاثیر گذار باشد.

3. طراحی مجدد تست : طراحی مجدد زمانی که پوشش ازحد مطلوب فاصله دارد و پایین تر است و توسعه دهندگان نیازاست که تست واحد ها را طراحی مجدد کنند. دربرخی اوقات استفاده از تست واحد و تست سیستم بصورت ترکیبی می تواند به مشکل، پوشش تست کمک کند همچنین با در نظر گرفتن تعداد خطا ها[2]، توسعه دهندگان می توانند تست های واحد را طراحی مجدد کنند تا پوشش تست بالاتر رود.

4.فهم نیازمندی ها: فهم درستی ازنیازمندی ها یکی از شرایط مهم قبل ازشروع به نوشتن آزمایه است، مخصوصا زمانی که بر روی مشکلات بزرگ و پیچیده محاسباتی کار می شود. استفاده ازروش توسعه مبتنی

[1] Best Practice
[2] Bug

برتست پذیرش شده که در بخش5-2 بیان شد، یکی از روش هایی است که در این زمینه بسیار کمک کننده است و زمان زیادی راصرفه جویی می کند.

5.کسب اعتماد بنفس : زمانی که توسعه دهندگان بر روی نرم افزار کار می کنند، نیازاست که خروجی های ساده تر را ابتدا چک کنند تا اعتمادبنفس بالاتری درنوشتن تست ها پیدا کنند.

جدول1 نگاشت مشکلات شش گانه تست و راه حل های پیشنهاد شده پنجگانه را نشان می دهد و بیان می کند هر یک ازاین راه حل ها کدام مشکل یا مشکلات را حل می کنند که با کلمه بله و کدام مشکلات با راه حل های پیشنهادی قابل حل نیست با کلمه خیر مشخص می شود.

جدول 1 : مشکلات و راه حل های تست در روش توسعه مبتنی بر تست

نام مشکل	استفاده از ابزار تست مناسب	تجربه	طراحی مجدد تست	فهم نیازمندی ها	کسب اعتماد بنفس
نوشتن آزمایه	بله	بله	خیر	بله	بله
پیچیدگی بالا	خیر	بله	خیر	بله	بله
پوشش کد	بله	بله	بله	بله	خیر
کمبود روش، ابزار،استاندارد	بله	خیر	خیر	خیر	خیر
زمان بر بودن نوشتن آزمایه	بله	بله	خیر	بله	خیر
تغییر کد یا نیازمندی ها	خیر	خیر	خیر	خیر	خیر

6-3-مشکلات بازسازی کد و راه حل های آن

در مجموع می توان به پنج مشکل رایج و چهار راه حل پیشنهاد شده در این حوزه اشاره کرد[Nanthaamornphong2015]. در ابتدا به بیان مشکلات می پردازیم:

1.وابستگی بین تست واحدها: ازآنجایی که پیاده سازی کد اغلب براساس تست است تا سند نیازمندی ها[1]بازسازی کد بسیار مشکل است اگر تست واحد ها خوب طراحی نشده باشد. ضمن اینکه اگر تست واحد ها در دسترس نباشد عملا بازسازی کد غیر ممکن است.

2.وابستگی در طراحی معماری: گرچه تست کردن لازمه عملیات بازسازی کد است، اما طراحی ضعیف یابد معماری ممکن است بازسازی کد را سخت یا غیر ممکن کند.

3.وابستگی در محیط توسعه: محیط توسعه شامل پلتفرم،زبان برنامه نویسی، ابزار و اجزای تعاملی است. بعنوان مثال بازسازی کدی که با زبان C نوشته شده است، به مراتب مشکل تر از بازسازی کد نوشته شده به زبان Python است. زیرا پایتون شی گرا است و C نیست. همچنین بازسازی کردن بدون ابزار نیز کار دشواری است. بعنوان مثال منطبق کردن API ها با ورژن های قبلی یکی از این نیاز هاست.

[1] Requirement Specification

4. کمبود دانش در ارتباط با اینکه چه زمانی و چگونه بازسازی کد انجام شود: یک مشکل مرسوم این است که اغلب توسعه دهندگان قادر به فهم بازسازی کد و مزایای آن نیستند. کمبود دانش در زمینه نحوه انجام بازسازی و یا علت باز سازی کد وجود دارد. همچنین بازسازی کد نیازمند دانش تکنیک های پیشرفته برنامه نویسی است مثل استفاده از قالب کد[1]. همچنین زمان استفاده از بازسازی کد نیز بسیار حائز اهمیت است زیرا تاخیردر این کار ممکن است باعث مشکلات فراوانی شود.

5. نرم افزار ها و کد های سنتی[2]: زمانی که توسعه دهندگان با کد های قدیمی و نرم افزار های سنتی کار می کنند بسیاردشوار است. چون توسعه دهنده ممکن است تست کافی برای اطمینان ازاینکه بازسازی کد مشکل نیست را ندارد.

حال به بررسی راه حل هایی در این زمینه می پردازیم:

1.راهنمایی توسعه دهندگان: فراگیری روش بازسازی کد امری لازم است. رهبر تیم باید اعضای تیم را متقاعد کند، که بازسازی کد بر روی کیفیت نرم افزار تاثیر بسیاری می گذارد. توسعه دهندگان با تجربه باید با راهنمایی و کمک به سایر توسعه دهندگان این امر را تسهیل کنند.

2.استفاده ازابزار های بازسازی کد: ابزارهای اتوماتیک سازی فرایند بازسازی کد می توانند بسیار مفید واقع شوند، مخصوصا زمانیکه بر روی برنامه های بزرگ و پیچیده کار می شود. مهم تر اینکه ابزارهای بازسازی باید قابلیت یکپارچه سازی با IDE ها را داشته باشند.

3.طراحی مجدد معماری نرم افزار:دربرخی موارد بسیار دشوار است که یک کد ضعیف را بازسازی کرد، لذا طراحی مجدد معماری نرم افزار ممکن است، زمان کمتری نسبت به بازسازی کد ضعیف ببرد.

4. طراحی مجدد تست واحد ها: طراحی مجدد تست واحدها به توسعه دهندگان کمک می کند زمانی را که صرف بازسازی کد می شود، کاهش دهند. از جمله ازابتدا طراحی کردن برخی ازبدترین آزمایه ها.

درجدول 2 نگاشت مشکلات و راه حل های پیشنهادی در حوزه بازسازی کد را ملاحظه می کنید.

جدول 2 : مشکلات و راه حل های بازسازی کد در روش توسعه مبتنی برتست

نام مشکل	راهنمایی توسعه دهندگان	استفاده از ابزار بازسازی کد	طراحی مجدد معماری نرم افزار	طراحی مجدد تست واحد ها
وابستگی بین تست واحد ها	خیر	خیر	خیر	بله
وابستگی در طراحی معماری	خیر	خیر	بله	خیر
وابستگی در محیط توسعه	خیر	بله	خیر	خیر
کمبود دانش در مورد بازسازی کد	بله	خیر	بله	خیر
نرم افزارهای سنتی	بله	بله	خیر	بله

[1] Code Template
[2] Legacy Softwares

7- اثرات روش توسعه مبتنی برتست بر روی صفات کیفی

این قسمت به بررسی نتایج مطالعات موردی را برروی صفات کیفی نرم افزار بررسی می کند و اطلاعات جمع آوری شده ازاین مقالات را دسته بندی می کند. صفات کیفی یک محصول، فرایندها ومنابع می توانند مرتبط با کیفیت داخلی اشیا یا کیفیت خارجی باشند که نیازمند نه تنها توجه به خود شی، بلکه به شرایط خارجی آن نیز به همان میزان است[Pezze2008]. این صفات می توانند بصورت آماده و مستقیم اندازه گیری شوند، یا بصورت غیر مستقیم از ترکیبی از چند صفت مختلف بوجود بیایند. در طول مطالعات، حدود 10 صفت کیفی داخلی وخارجی که می توانند تاثیرات روش را به خوبی بیان کنند، جمع آوری شد. مروری بر روی نتایج نشان دهنده آن است که تاثیرات مختلفی بر روی صفات کیفی در مطالعات ثبت شده است. در ادامه به بررسی این صفات پرداخته شده است.

1.چگالی خطا[1] و تعداد خطا: در توسعه مبتنی بر تست که کد تست پیش از کد اجرایی نوشته می شود، بطور اتوماتیک خود این موضوع باعث کاهش تعداد خطا ها می شود. زیرا قسمت کمی از کد تست نشده باقی می ماند. میزان خطا ها در تعدادی از پروژه ها با سایز های مختلف کاهش یافته بعنوان مثال کمپانی IBM یک تیم برای پیاده سازی توسعه مبتنی بر تست استخدام کردبرای اولین بار و با ایجاد یک نرم افزار اعلام کرد که تعداد خطا ها نسبت به حالت بدون استفاده از توسعه مبتنی برتست نصف شده است.وتعداد دقیق خطا در این پروژه 4 خطا در 1000 خط پروژه بوده است[George2003]. درحالی که تعداد خطا ها در این روش کاهش می یابد، ولی خطا های بزرگ درنرم افزار نسبت به حالتی که از این روش نیز استفاده نشود چندان متفاوت نیست[Willaim2003] .مایکروسافت دربسیاری از پروژه های خود با این روش کاهش 60 تا 90 درصدی در تعداد خرابی ها در مقایسه با پروژه های مشابه بدون استفاده از این روش را تجربه کرده است [Nagappan2008] .

2.پوشش کد: پوشش دادن خط خاصی از کد اجرایی نیازمند آن است که برای آن قسمت یک تست اجرایی به طوری که آن خط را بیازماید، نوشته شود. در کنار پوشش عبارات ساده در برنامه نویسی می توان، میزان پوشش کد در این روش را نیز مشخص نمود [Pezze2008] .بدون آزمایه در واقع پوششی برای کد وجود ندارد و باید کدها را با روش های دیگر در پایان کار تست نمود. اما در صنعت نتایج نشان دهنده پوشش مناسب کد توسط آزمایه هاست. در مایکروسافت و IBM میزان پوشش کد درپروژه های پیاده سازی شده با این روش در حدود 80 تا 90 درصد بوده است در حالیکه در یک پروژه بسیار بزرگ نیز این پوشش، به میزان 60 درصد بوده است [Nagappan2008]. در [William2003] همچنین به نتایج مشابهی به آنچه توسط IBM و مایکروسافت گزارش داده اند در پروژه های سبک تر اشاره شده است.

3.پیچیدگی[2]: پیچیدگی در واقع صفتی است که برای بیان مولفه های کد به تنهایی و ساختار داخلی آن ها استفاده می شود. بعنوان مثال پیچیدگی توسط فرمول مک کیب، در [McCabe1976] به این صورت است که میزان انشعابات کد، محاسبه می شود. در حالی که در روش های مدرن برای محاسبه پیچیدگی، ساختار متد ها و کلاس ها را نیز مد نظر قرار می دهند [Chidumber1994]. بعلت آنکه در این روش کد در تکه های کوچک تری توسعه می یابد، کاهش میزان پیچیدگی متصور است. کاهش میزان پیچیدگی در کلاس های برنامه نویسی در مطالعاتی که چه در صنعت و چه بصورت آکادمیک صورت گرفته ، مشاهده شده است.

[1] Defect
[2] Complexity

اما در مطالعاتی که در صنعت انجام شده، کلاس های داخل یک مولفه همبستگی[1] بالاترهستند و کمتر شامل رفتار[2]های پیچیده هستند. که باعث می شود در نهایت محصول نرم افزاری پیچیدگی کمتری داشته باشد [Janzen2008].

4.وابستگی[3] و همبستگی: همبستگی و وابستگی دو پارامتر مهم در ساختار کد نویسی شی گرا هستند و عبارتست از ارتباط میان کلاس ها و سازگاری داخلی خود کلاس ها[Chidumber1994]. زمانی که ازروش توسعه مبتنی برتست استفاده می شود، عملا مجبور به تولید کلاس هایی هستیم که وظایف محدود و مشخصی دارند. مقالات کمی در مورد مقدار این دو پرامتر گزارش داده اند. در صنعت در [Janzen2008] گزارش داده شده است که میزان وابستگی در پروژه های که ازاین روش استفاده شده بود، بیشتر شده است و همین نتیجه نیزدر یک مطالعه دانشگاهی نیز اعلام شده است[Janzen2006]. با توجه به [Janzen2008] میزان همبستگی در کلاس ها با این روش کاهش یافته که این نتایج دقیقا برخلاف آن چیزی است که ازمیزان وابستگی و همبستگی در شی گرایی انتظار داریم.

5.سایز: در روش توسعه مبتنی برتست از آنجایی که مبتنی است برنوشتن میزان حداقلی کد برای آنکه آزمایه ها را با موفقیت پشت سر بگذارد، باید انتظار داشت این روش بر روی سایز کد تاثیر گذار باشد. نسبت بین کدهای تست و کد های غیر تست راهی برای محاسبه سایز کد های تست است. نتایج مطالعاتی نشان دهنده مقادیر متفاوتی برای سایز کد های تست در این روش است. در مایکروسافت بالاترین نسبت کد تست به کد غیر تست 89%عنوان شده است. ضمن اینکه پایین ترین عدد گزارش شده نیز 39% است که برای پروژه های بزرگ بیان شده است و اعداد بین این دو عدد برای سایر پروژه هاست [Nagappan2008]. عدد 48% که IBM گزارش کرده نیز در این بازه قرار می گیرد. بدیهی است بدون استفاده ازاین روش و با بکارگیری روش های تست درآخرمی توان به نسبت های خیلی خوبی دست یافت.

6-تلاش[4]: تلاش یک صفت کیفی فرایندی است.در اینجا می توان آن را به این صورت تعریف کرد که میزان اعمال انجام شده بر روی یک وظیفه مشخص در یک محصول نرم افزاری. معمولا یک رابطه بین میزان تلاش و طول مدت و با توجه به محتوا وجود دارد. طول مدت یک وظیفه می تواند بعنوان یک نشانگر، برای میزان تلاش صرف شده تلقی شود. روش توسعه مبتنی برتست فرایند طراحی و پیاده سازی یک محصول را تغییر می دهد بنابراین تلاشی که برای این فرایند باید انجام شود، می تواند متفاوت باشد. تاثیرات ممکن است تنها محدود به این حوزه ها نباشد از آنجایی که دسترس پذیری تست های اتوماتیک وابستگی زیادی به فعالیت های تایید و اعتبار سنجی دارد. نوشتن تست ها زمان بر است یا فاکتور های دیگری نیز وجود دارد که برروی فرایند توسعه تاثیر گذار است. به این علت است که استفاده ازاین روش باعث افزایش میزان تلاش می شود. در مایکروسافت و IBM متخصصان تخمین زده اند زمان توسعه بین 15 تا 30 درصد طولانی تر می شود. در [Dogsa2011]گزارش شده که توسعه در یک پروژه 6 ماهه درحدود 3000 تا 4000 نفر-ساعت افزایش یافته است واین بعلت استفاده از این روش بوده است. در [George2003] و [Canfora2006] نیز به نتایجی مشابه اشاره شده است.

[1] Cohesion
[2] Method
[3] Coupling
[4] Effort

7.صفات کیفی خارجی: کیفیت خارجی در این مطالعه عبارتست از موارد کیفی، ذهنی و اطلاعات خاص محیطی از مطالعات تجربی که به آسانی قابل تبدیل به معیارهای قابل سنجش نیست. بعنوان مثال روش های تحقیقی مبتنی برترکیب چند روش در تعدادی از مقالات، پیاده سازی شده است. که هریک از آنها از مقالات مروری دیگر و مصاحبه هایی که با ذینفعان انجام داده اند، دیدگاه های ذهنی ای نسبت به روش توسعه مبتنی برتست را جمع آوری کرده اند. بعنوان مثال در [Huang2009] در مصاحبه ای که با مشتریان یک پروژه پیاده سازی شده به روش توسعه مبتنی برتست انجام داده بود و از آنها در مورد کیفیت محصول پس از یک ماه استفاده پرسیده بود، کاربران نهایی تغییر و تاثیر خاصی نسبت به زمانی که از سایر روش های توسعه نرم افزار استفاده می شود، گزارش نکرده اند. این یعنی آنکه توسعه مبتنی برتست بر روی صفات کیفی که توسط مشتری قابل مشاهده است تاثیر زیادی ندارد. احتمالا توسعه دهندگانی که درفازهای طراحی و پیاده سازی محصول بوده اند، بتوانند توضیحات بهتری در ارتباط با کیفیت پیاده سازی محصول با این روش بدهند. بعنوان مثال در [Dogsa2011] و[Gupta2007] نیز همین پرسش از توسعه دهندگان شده است که پاسخ آنها بهبود کیفیت کد[1] و قابلیت تست[2] بالای محصول بوده است.

8.میزان بهره وری[3]: بهره وری یک پارامتر محاسباتی براساس رابطه بین مقدار یک سری آیتم ملموس تولید شده و تلاشی که لازم است تا نتیجه خروجی را تولید کند، می باشد. بهره وری یک پارامتر کیفی خارجی است که برای محیط خارج از محیط توسعه قابل لمس و مشاهده است. توسعه مبتنی برتست بنظر می رسد یک فاکتور در جهت افزایش میزان تلاش توسعه دهندگان همانطور که اشاره شد، باشد. اما آیا این روش باعث افزایش سرعت پیاده سازی داستان های کاربر[4] می شود یا بر روی توسعه دهندگان که کد می نویسند تاثیر گذار است؟ چندین مورد مطالعه در مورد ارتباط بین توسعه مبتنی برتست و میزان بهره وری صحبت کرده اند. در [Dogsa2011] اشاره شده در این روش توسعه دهندگان با سرعت آرام تری اقدام به تولید کد اجرایی می کنند و معتقدند در اینجا، نسبت به سایر تیم های توسعه از بهره وری بالاتری برخوردار است.

9.قابلیت نگداری: قابلیت نگهداری یک ویژگی است که مرتبط است با جنبه های تکامل یک نرم افزار بعنوان مثال، چقدر ساده است تا بفهمیم در کدمان باید چه قسمت هایی را تغییر دهیم و البته این باید همراه با مکانیزم های مناسب برای ارزیابی میزان تاثیرات باشد[Cook2001]. آرایه ای از تست های اتوماتیک در این روش می تواند حداقل باعث افزایش میزان قابلیت تست و پایداری نرم افزار شود که خود دلالت برافزایش میزان قابلیت نگهداری دارد. مقالات بسیار کمی در این حوزه گزارش داده اند و این بخش نیازمند کار بیشتر است. ولی همانطور که اشاره در خواست تغییر برای کدی که به این روش توسعه یافته، زمان کمتری را می برد و تلاش کمتری را نیز طلب می کند.

یک جمع بندی از مطالعات انجام شده با در نظر گرفتم پارامتر هایی که در بالا بیان شد در جدول 3 آورده شده است. در این جدول نه عدد پارامتر بخش 7 را در سه بخش تقسیم بندی کرده ایم که شامل 1. تاثیر شدید مثبت روش بر روی پارامتر 2.تاثیر خنثی یا غیر قابل اندازه گیری روش برروی پارامتر و 3.تاثیر شدید منفی روش توسعه مبتنی بر تست بر روی پارامتر است. تتیجه نوشته شده در جدول 3 متاثر از میانگین تمامی مطالعات انجام شده در مورد پارامتر خاص در این مقاله است. اما باید این موضوع را هم در نظر گرفت

[1] Code Quality
[2] Testability
[3] Productability
[4] User Story

که تقسیم بندی جدول 3 یک دانه بندی درشت است و ممکن است در یک مطالعه مشابه نتایج متفاوت باشد.

جدول 3 : پارامتر های مورد بررسی در روش توسعه مبتنی بر تست و تاثیرات روش بر روی آنها

نام پارامتر	نوع تاثیر بر روی پارمتر	نام پارامتر	نوع تاثیر بر روی پارامتر
چگالی خطا و تعداد خطا	تاثیر مثبت	میزان تلاش	تاثیر منفی
پوشش کد	تاثیر مثبت	صفات کیفی	تاثیر مثبت
پیچیدگی	تاثیر مثبت	میزان بهره وری	تاثیر منفی
وابستگی و همبستگی	بدون تاثیر	قابلیت نگهداری	بدون تاثیر
سایز	بدون تاثیر		

8-نتیجه گیری

در این مقاله با بررسی مطالعات موردی انجام شده بر روی روش توسعه مبتنی برتست دیدیم که چرخه حیات این روش چگونه عمل می کند. در مورد تفاوت های این ورش با سایر فرایند های توسعه نرم افزار صحبت شد و دیدیم که می توان این روش را نیز یک فرایند توسعه قلمداد کرد البته همانطور که از مرور مقالات می توان نتیجه گرفت در حال حاضر بیشتر روش های چابک به این سمت سوق پیدار کرده اند که از چرخه حیات این روش در متدولوژی خود استفاده کنند که مثال بارز آن روش توسعه مبتنی بر تست پذیرش شده در متدولوژی های چابک است. در ادامه به مشکلات و معضلات گزارش شده این ورش در مقالات پرداختیم و راه حل های پیشنهادی برای این مشکلات را دسته بندی نمودیم. در نهایت اقدام با اندازه گیری میزان بهبود در 9 صفت کیفی و پارمتر های مختلف ازدیدگاه مقالات مروری پرداخته شد. با توجه به نتایج بدست آمده در بخش 7 که بیشتر مبتنی برگزارشات صنعتی و عملیاتی استفاده از این روش است می توان به این نتیجه رسید که روش توسعه مبتنی برتست ممکن است باعث افزایش تلاش یا زمان اجرای پروژه شود ولی پارامتر های مهمی از جمله بهبود میزان بهره وری کد نهایی تولید شده و همچنین بهبود در کیفیت کد و قابلیت استفاده مجدد را به همراه دارد.

9- مراجع:

[1] Beck, K. (2000). ***Extreme Programming Explained***. Addison- Wesley.

[2] Beck, K. (2001). ***Aim, Fire***. *Software*, 18(5):87-89, Sept.-Oct. 2001.

[3] Larman, C., & Basili, V. R. (2003). ***Iterative and Incremental Development: A Brief History***. *Computer* (June 2003), 47-56.

[4] Beck, K. (2003). ***Test-Driven Development: By Example***. Addison-Wesley Professional.

[5] Astels, D. (2003). ***Test-Driven Development: A Practical Guide*** . Pearson Education, Inc., Upper Saddle River, NJ.

[6] Wake, W. (2001). ***The Test-First Stoplight***. http://xp123.com/articles/the-test-first-stoplight/

[7] Ambler, S. (2010). ***How Agile Are You?*** 2010 Survey Results. http://www.ambysoft.com /surveys /howAgileAreYou2010.html

[8] Beck, K. (2010). ***CD Survey: What practices do developers use?*** http://www .threeriversinstitute.org /blog/?p=541

[9] West, D., & Grant, T. (2010). ***Agile Development: Mainstream Adoption Has Changed Agility.*** Cambridge: Forrester.

[10] S. Kumar and S. Bansal (2013). Comparative ***Study of Test Driven Development with Traditional Techniques***, International Journal of Soft Computing and Engineering, vol. 3, pp. 352-360.

[11] M. Pancur and M. Ciglaric (2011). ***Impact of Test-driven Development on productivity, code and tests: A controlled experiment***, Information and Software Technology, vol. 53, pp. 557-573,

[12] D. Janzen and H. Saiedian (2006). ***On the Influence of Test-Driven Developement on Software Design,*** presented at the Software Engineering Education and Training, Turtle Bay, HI,

[13] A. Cauevic, D. Sundmark, and S. Punnekkat (2012). ***Quality of Testing in Test Driven Development***, presented at the Quality of Information and Communications Technology, Lisbon.

[14] C. Consulting***. Agile Test Driven Development.*** Available: http://centricconsulting.com/ agile-test-driven- development/

[15] S. H. Edwards (2004). ***Using software testing to move students from trial-anderror to reflection-in-action,*** in Proceedings of the 35th SIGSCE technical symposium on Computer science education New York, NY, USA, pp. 26-30.

[16] H. Erdogmus, M. Morisio, and M. Torchiano (2005). ***On the effectiveness of the test-first approach to programming,*** Software Engineering, IEEE Transactions, vol. 31, pp. 226-237.

[17] D. Janzen and H. Saiedian (2008). ***Does test-driven development really improve software design quality?***, IEEE 77-84. Software, vol. 25, pp

[18] R. Kaufmann and D. Janzen (2003). ***Implications of test-driven development: a pilot study,*** in OOPSLA '03: Companion of the 18th annual ACM SIG-PLAN conference on Object-oriented programming, systems, languages, and applications, New York, NY, USA, pp. 298-299.

[19] T. Bhat and N. Nagappan (2006). ***Evaluating report of the TDD process. Using this approach, the the efficacy of test-driven development: industrial case studies,*** in ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on Empirical software pp. 356-363. engineering, New York, NY, USA.

[20] R. Amlani (2013). ***Comparison of Different SDLC Models,*** Journal of Computers Applications & Information Technology, vol. 2.

[21] S. Kollanus (2010). ***Test-Driven Development- Still a Promising Approach?,*** presented at the Quality of Information and Communications Technology, Porto.

[22] Aniche, M. F., & Gerosa, M. A. (2010). ***Most Common Mistakes in Test-Driven Development Practice: Results from an Online Survey with Developers***. Third International Conference on Software Testing, Verification, and Validation Workshops , 469-478.

[23] Bajaj.K , patel.H , Patel. J (2015). ***Evolutionary Software Development using Test Driven Approach,*** Computing and Communication (IEMCON), 2015 International Conference and Workshop on, Vancouver, BC, Canada.

[24] North, D. (2006). ***Introducing BDD.*** Better Software (March 2006).

[25] Astels, D. (2006). ***Google TechTalk: Beyond Test Driven Develoment: Behavior Driven Development.*** http://www.youtube.com/watch?v=XOkHh8zF33o

[26] Rimmer, C. (2010). ***Introduction. Behaviour-Driven Development***: http://behaviour-driven.org/Introduction

[27] Madeyski, L. (2010). ***Test-Driven Development: An Empirical Evaluation of Agile Practice.*** Berlin: Springer-Verlag.

[28] Nanthaamornphong.A , Carver . C.J, (2015), ***Test-Driven Development in scientific software: a survey***, Software Quality Journal , Volume 25, Issue 2, pp 343–372.

[29] Pezze, M., Young, M. (2008). ***Software Testing and Analysis: Process, Principles and Techniques***. Wiley, Chichester.

[30] George, B., Williams, L. (2003), ***An Initial Investigation of Test Driven Development in Industry***. In: Proceedings of the 2003 ACM Symposium on Applied Computing, SAC 2003, pp. 1135–1139. ACM, New York .

[31] Williams, L., Maximilien, E.M., Vouk, M. (2003***), Test-Driven Development as a Defect Reduction Practice.*** In: Proceedings of the 14th International Symposium on Software Reliability Engineering, ISSRE 2003, pp. 34–45 .

[32] Nagappan, N., Maximilien, E., Bhat, T., Williams, L.(2008), ***Realizing Quality Improvement through Test Driven Development: Results and Experiences of Four Industrial Teams.*** Empirical Software Engineering 13, 289–302

[33] McCabe, T. (1976). ***A Complexity Measure.*** IEEE Transactions on Software Engineering SE 2(4), 308–320.

[34] Chidamber, S., Kemerer, C. (1994), ***A Metrics Suite for Object Oriented Design***. IEEE Transactions on Software Engineering 20(6), 476–493.

[35] Dogša, T., Batič, D. (2011***). The Effectiveness of Test-Driven Development: An Industrial Case Study.*** Software Quality Journal 19, 643–661.

[36] Canfora, G., Cimitile, A., Garcia, F., Piattini, M., Visaggio, C.A. (2006). ***Evaluating Advantages of Test Driven Development: A Controlled Experiment with Professionals.*** In: Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering, ISESE 2006, pp. 364–371. ACM, New York

[37] Huang, L., Holcombe, M. (2009). ***Empirical Investigation Towards the Effectiveness of Test First Programming***. Information and Software Technology 51(1), 182–194.

[38] Gupta, A., Jalote, P.(2007). ***An Experimental Evaluation of the Effectiveness and Efficiency of the Test Driven Development***. In: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement, ESEM 2007, pp. 285–294.

[39] Cook, S., He, J., Harrison, R. (2001). ***Dynamic and Static Views of Software Evolution.*** In: Proceedings of the IEEE International Conference on Software Maintenance, pp. 592–601.