

[Year]

# Software process

علی رضا حسن زاده  
SOFTWARE ENGINEERING

دانشگاه آیت الله خامنه‌ای



Edit with WPS Office

[Date]

0

یک روش توسعه نرم افزار است که در آن، نوشتمن تست‌های واحد Test-Driven Development (TDD) (Unit Tests) برای کد قبل از خود کد انجام می‌شود. این روش بخشی از متداول‌ترین طرزات است. در TDD، توسعه‌دهنده ابتدا تستی برای یک ویژگی می‌نویسد که طبیعتاً در ابتدا شکست می‌خورد. سپس کد لازم برای موفقیت تست نوشته می‌شود و در نهایت، کد برای بهبود کارایی و خوانایی بهینه‌سازی می‌شود.

#### ویژگی‌های کلیدی TDD :

- نوشتمن تست قبل از کد: در TDD، توسعه‌دهنده قبل از نوشتمن کد اصلی، تست را برای آن می‌نویسد.
- چرخه سه مرحله‌ای (Red-Green-Refactor) شامل سه مرحله‌ی مهم "Red" (شکست تست)، "Green" (موفقیت تست)، و "Refactor" (بهینه‌سازی کد).
- پوشش بالا برای تست‌های واحد: این روش تضمین می‌کند که بخش‌های مختلف کد به خوبی تست شده و از بروز خطاهای ناخواسته جلوگیری شود.
- تمرکز بر رفع مشکلات کوچک در هر مرحله: توسعه‌دهنده‌گان با نوشتمن کدهای کوچک برای هر تست، مشکلات را به صورت قدم‌به‌قدم حل می‌کنند.

#### مزایای TDD :

1. کاهش خطأ و باگ:
  - TDD به شدت به کاهش خطاهای کمک می‌کند، زیرا تست‌ها کد را پیش از نوشتمن و در زمان اجرا بررسی می‌کنند.
2. بهبود خوانایی و کیفیت کد:
  - با روش TDD، کد به بخش‌های کوچک‌تر و خواناتری تبدیل می‌شود که به راحتی قابل تست و نگهداری است.
3. افزایش اعتماد به تغییرات:
  - به دلیل وجود تست‌های واحد دقیق، توسعه‌دهنده‌گان با اطمینان بیشتری تغییرات را انجام می‌دهند زیرا مطمئن هستند تست‌ها مشکلات احتمالی را شناسایی خواهند کرد.
4. مستندسازی کد به صورت غیررسمی:
  - تست‌های واحد در TDD به نوعی مستندات غیررسمی هستند که عملکرد کد را شرح می‌دهند.

#### معایب TDD :

1. نیاز به زمان و هزینه بیشتر:
  - پیاده‌سازی TDD نیاز به زمان بیشتری برای نوشتمن تست‌ها دارد و ممکن است در پروژه‌های با بودجه محدود چالش برانگیز باشد.
2. نیاز به تسلط بالا به تست‌نویسی:
  - توسعه‌دهنده‌گان باید در تست‌نویسی و ابزارهای آن مهارت کافی داشته باشند تا تست‌های موثری ایجاد کنند.
3. مناسب نبودن برای پروژه‌های کوچک و سریع:
  - در پروژه‌های کوتاه‌مدت که زمان تحویل محدود است، TDD ممکن است باعث افزایش زمان شود و بهمین دلیل کارایی نداشته باشد.
4. پوشش همه‌ی جنبه‌ها دشوار است:
  - در عمل، پوشش تمام موارد پیچیده توسط تست‌ها امکان‌پذیر نیست و برخی مشکلات



ممکن است بدون پوشش باقی بمانند.

### کاربردهای TDD :

- پروژه‌های بزرگ و بلندمدت TDD بیشتر در پروژه‌هایی که نیاز به نگهداری و تغییرات مکرر دارند، کارآمد است.
- توسعه سیستم‌های حساس و حیاتی TDD در پروژه‌هایی که باید خطاهای خداقل بررسند و کیفیت بالا حیاتی است، مانند سیستم‌های پزشکی، مالی و حمل و نقل.
- محیط‌های چابک (Agile) به دلیل هماهنگی بالا بین TDD و متدهای چابک، این روش در تیم‌های چابک بسیار کاربردی است.
- توسعه نرم‌افزارهای تست‌پذیر TDD مناسب پروژه‌هایی است که قابلیت تست‌پذیری بالایی دارند و به طور مداوم تغییر می‌کنند.

Feature-Driven Development (FDD) یا توسعه بر مبنای ویژگی‌ها، یک متدولوژی توسعه نرم‌افزار است که بر ساخت ویژگی‌های کاربردی در قالب بخش‌های کوچک و قابل تحويل تمرکز دارد. این روش که بخشی از متدهای چابک است، ابتدا به شناسایی و مدل‌سازی ویژگی‌ها می‌پردازد و سپس با توسعه و تحويل مداوم، به ایجاد نرم‌افزار کامل می‌انجامد FDD. معمولاً برای تیم‌های بزرگ و پروژه‌های پیچیده مناسب است و مراحل مشخصی دارد.

### ویژگی‌های کلیدی FDD :

- مدل‌سازی کلان و ویژگی‌محور: پروژه در ابتدا به مدل‌های کلی و کوچکتری تقسیم می‌شود تا ویژگی‌های مختلف در قالب مراحل کوچک پیاده‌سازی شوند.
- تحویل‌های کوتاه و تدریجی FDD: بر اساس تحویل‌های کوچک و پیوسته کار می‌کند و در هر مرحله ویژگی‌های جدیدی را به پروژه اضافه می‌کند.
- ساختار تیمی و نقش‌های مشخص: این متد دارای نقش‌های مشخصی مانند توسعه‌دهنده‌ی اصلی، مالک ویژگی و مدیر پروژه است که وظایف هر نقش تعریف شده است.
- پیشرفت قابل اندازه‌گیری: در FDD، معیارهای دقیقی برای اندازه‌گیری پیشرفت وجود دارد و این متد به صورت کاملاً ساختاریافته مراحل پروژه را پیگیری می‌کند.

### مزایای FDD :

- قابلیت پیش‌بینی بالا در پروژه‌های بزرگ:
  - با توجه به ساختار مشخص و قابلیت تحويل تدریجی ویژگی‌ها، FDD برای پروژه‌های بزرگ و پیچیده امکان برنامه‌ریزی و پیش‌بینی دقیق‌تر را فراهم می‌کند.
- مدیریت بهتر تیم‌های بزرگ:
  - ساختار تیمی و نقش‌های تعریف شده در FDD به ایجاد هماهنگی و کارایی بالاتر در تیم‌های بزرگ کمک می‌کند.
- تمرکز بر نیازهای مشتری:
  - با تقسیم پروژه به ویژگی‌های کوچک و قابل تحويل، FDD به مشتری این امکان را می‌دهد که به طور تدریجی خروجی‌ها را بررسی کند و بازخورد دهد.
- سازگار با متدهای چابک:
  - اصول چابک را دنبال می‌کند و می‌تواند به راحتی با سایر متدهای چابک ترکیب شود.



## معایب FDD :

۱. مناسب نبودن برای پروژه‌های کوچک:
  - ۰ به دلیل ساختار سنگین و مراحل برنامه‌ریزی طولانی، FDD برای پروژه‌های کوچک و کوتاه‌مدت بهینه نیست.
۲. نیاز به تیم‌های تخصصی:
  - ۰ در FDD نقش‌ها و مسئولیت‌ها به دقت تعریف شده‌اند که نیاز به تیم‌های تخصصی و توانمند دارد.
۳. نیاز به زمان بیشتر برای برنامه‌ریزی و مدل‌سازی:
  - ۰ مراحل اولیه شامل زمان قابل توجهی برای مدل‌سازی و برنامه‌ریزی است که ممکن است در برخی پروژه‌ها مناسب نباشد.
۴. احتمال ناکارآمدی در صورت عدم تعریف دقیق ویژگی‌ها:
  - ۰ در صورت عدم تعریف دقیق ویژگی‌ها، ممکن است تحويل تدریجی ویژگی‌ها به نتیجه نهایی دلخواه منجر نشود.

## کاربردهای FDD :

- پروژه‌های بزرگ و پیچیده FDD: به دلیل ساختار دقیق و امکان برنامه‌ریزی برای تیم‌های بزرگ، در پروژه‌های بزرگ و سازمانی کارآمد است.
- محیط‌های تجاری و مالی: به دلیل نیاز به ساختار و کنترل دقیق در محیط‌های مالی، این متداول‌وزیر مناسب این محیط‌هاست.
- سازمان‌های با تمرکز بر خروجی‌های تدریجی: سازمان‌هایی که نیازمند ارائه و ارزیابی خروجی‌های تدریجی هستند، می‌توانند از FDD بهره‌مند شوند.
- پروژه‌هایی با نیاز به قابلیت اندازه‌گیری پیشرفت: در پروژه‌هایی که نظارت دقیق بر پیشرفت هر مرحله اهمیت دارد، FDD می‌تواند موثر باشد.

Behavior-Driven Development (BDD) یا توسعه‌ی رفتار محور، یک متداول‌وزیر توسعه نرم‌افزار است که با تمرکز بر رفتار سیستم و تعاملات مختلف مختلف آن با کاربران به طراحی تست‌های نرم‌افزاری کمک می‌کند. این روش به توسعه‌دهندگان و ذینفعان پروژه اجازه می‌دهد تا با استفاده از زبانی مشترک، نیازمندی‌ها و رفتارهای مورد انتظار سیستم را مشخص کنند BDD. به‌طور معمول بر اساس توسعه‌ی تست‌ها در چارچوب‌های قابل خوانش توسط انسان‌پیاده‌سازی می‌شود و از ابزارهایی مانند Cucumber، JBehave و SpecFlow پشتیبانی می‌گیرد

## ویژگی‌های کلیدی BDD :

۱. استفاده از زبان مشترک و قابل فهم برای همه: در BDD، از زبان ساده‌ای استفاده می‌شود که تمامی ذینفعان پروژه، از جمله مدیران و کاربران نهایی، بتوانند آن را درک کنند.
۲. نوشن سناریوهای رفتاری: تست‌ها به صورت سناریوهای رفتاری نوشته می‌شوند که توصیف کننده ی چگونگی رفتار سیستم در موقعیت‌های مختلف هستند.
۳. تعریف ویژگی‌ها با استفاده از قالب "Gherkin": در این روش، ویژگی‌ها عموماً در قالب Given-When-Then نوشته می‌شوند تا نیازمندی‌ها و رفتارها دقیقاً مشخص شوند.
۴. تأکید بر تعاملات و رفتارها BDD: بیشتر به تعاملات و رفتار سیستم در برابر ورودی‌های مختلف می‌پردازد تا جزئیات پیاده‌سازی کد.

## مزایای BDD :



۱. افزایش تعامل و هماهنگی بین تیمها:
  - با استفاده از زبان مشترک، BDD باعث می‌شود که توسعه‌دهندگان، تست‌ها و ذینفعان پروژه بتوانند بهتر با هم همکاری کنند.
۲. بهبود فهم نیازمندی‌ها:
  - در BDD، نیازمندی‌ها به طور شفاف و واضح با سناریوهای رفتاری تعریف می‌شوند که این امر به درک بهتر نیازمندی‌ها و انتظارات مشتریان کمک می‌کند.
۳. افزایش تست‌پذیری و کیفیت کد:
  - به نوشتن تست‌های جامع و سناریوهای واقعی کمک می‌کند که موجب بهبود کیفیت کد و کاهش باگ‌ها می‌شود.
۴. قابلیت خواندن تست‌ها توسط افراد غیرفنی:
  - از آنجا که سناریوها به زبان ساده‌ای نوشته می‌شوند، ذینفعان غیرفنی نیز می‌توانند آن‌ها را مرور و تأیید کنند.

: BDD معایب

۱. نیاز به زمان بیشتر برای نوشتن سناریوها:
  - پیاده‌سازی BDD نیاز به صرف زمان بیشتری برای نوشتن سناریوهای رفتاری دقیق و تست‌های قابل فهم دارد.
۲. نیاز به یادگیری ابزارها و تکنیک‌های جدید:
  - اعضای تیم باید ابزارها و چارچوب‌های BDD را فراگیرند که ممکن است به هزینه و زمان بیشتری نیاز داشته باشد.
۳. پیچیدگی در پروژه‌های بزرگ و پیچیده:
  - برای پروژه‌های بزرگ و پیچیده، ممکن است تعداد سناریوها زیاد و مدیریت آن‌ها دشوار شود.
۴. نیاز به همکاری نزدیک تیمها:
  - موفقیت در پیاده‌سازی BDD وابسته به همکاری و ارتباط قوی بین تیم‌های توسعه، تست و ذینفعان است که در صورت نبود همکاری، این روش ممکن است کارایی لازم را نداشته باشد.

: کاربردهای BDD

- پروژه‌های با تعاملات پیچیده و متعدد BDD: در پروژه‌هایی که نیاز به سناریوهای تعاملی پیچیده دارند و رفتارهای کاربر را در موقعیت‌های مختلف پیش‌بینی می‌کنند، بسیار کارآمد است.
- محیط‌های چاپک و تیم‌های چندتخصصی: به دلیل تأکید بر همکاری و شفافیت، BDD در تیم‌های چاپک که اعضای چندتخصصی دارند، بسیار مؤثر است.
- پروژه‌های با نیاز به مستندسازی شفاف و قابل فهم: در پروژه‌هایی که مشتری یا ذینفعان نیاز به دید کلی و شفاف از عملکرد و رفتار سیستم دارند، BDD می‌تواند مفید باشد.
- سیستم‌های مالی، تجاری و بهداشتی: به دلیل اهمیت بالای رفتارهای دقیق و صحیح در این سیستم‌ها، BDD به ویژه در این حوزه‌ها مورد استفاده قرار می‌گیرد.

## ۱. Component-Driven Development (CDD)

Component-Driven Development (CDD) یا توسعه بر مبنای مؤلفه‌ها، متداول‌تر است که به ویژه در توسعه رابطه‌ای کاربری مدرن و سیستم‌های مقیاس‌پذیر و چندلایه کاربرد دارد. در CDD، برنامه‌ها از مجموعه‌ای از مؤلفه‌های جداگانه ساخته می‌شوند که هر کدام به صورت مستقل تست، توسعه و نگهداری



Edit with WPS Office

[Date]

4

می‌شوند. این روش به تیم‌ها اجازه می‌دهد که مؤلفه‌های نرم‌افزار را به صورت مجزا توسعه دهند و سپس آن‌ها را به صورت یکپارچه ترکیب کنند.

#### ویژگی‌های کلیدی CDD :

۱. جداسازی مؤلفه‌ها و استقلال توسعه: در CDD هر مؤلفه به صورت مستقل توسعه می‌یابد و وابستگی کمتری به سایر مؤلفه‌ها دارد.
۲. سهولت در تست و نگهداری: هر مؤلفه به طور مستقل تست می‌شود، که مدیریت تست و نگهداری سیستم را ساده‌تر می‌کند.
۳. قابلیت استفاده مجدد: مؤلفه‌ها قابل استفاده مجدد هستند و می‌توان آن‌ها را در پروژه‌های دیگر نیز به کار برد.
۴. مناسب برای رابطه‌های کاربری پیچیده CDD: به ویژه در توسعه UI پیچیده و اپلیکیشن‌های مبتنی بر وب که به سرعت توسعه و مقیاس‌پذیری نیاز دارند، کارآمد است.

#### مزایای CDD :

- بهبود بهره‌وری تیم‌ها: با جداسازی مؤلفه‌ها، تیم‌ها می‌توانند به طور مستقل روی مؤلفه‌های مختلف کار کنند.
- افزایش مقیاس‌پذیری CDD: به افزایش مقیاس‌پذیری پروژه کمک می‌کند زیرا هر مؤلفه می‌تواند به طور مستقل به روز و توسعه یابد.
- سهولت در تست و عیب‌یابی: تست‌های هر مؤلفه مستقل از سایر بخش‌ها انجام می‌شود، که خطایابی را ساده‌تر می‌کند.
- قابلیت استفاده مجدد بالا: مؤلفه‌های یکبار نوشته شده می‌توانند در پروژه‌های مختلف مورد استفاده قرار گیرند.

#### معایب CDD :

- پیچیدگی در هماهنگی مؤلفه‌ها: با افزایش تعداد مؤلفه‌ها، هماهنگی و یکپارچگی مؤلفه‌ها ممکن است دشوار شود.
- نیاز به مدیریت مؤلفه‌ها و وابستگی‌ها: برای مدیریت وابستگی‌های بین مؤلفه‌ها، نیاز به راهکارهای مناسبی مانند مدیریت وابستگی‌ها و کنترل نسخه است.
- مناسب نبودن برای پروژه‌های کوچک CDD: بیشتر برای پروژه‌های بزرگ و پیچیده مناسب است و در پروژه‌های کوچک ممکن است هزینه بیشتری در برداشته باشد.

#### کاربردهای CDD :

- توسعه رابطه‌های کاربری پیچیده: برای UI‌های پیچیده و اپلیکیشن‌های مبتنی بر وب.
- سیستم‌های مقیاس‌پذیر: مناسب سیستم‌هایی با نیاز به مقیاس‌پذیری بالا.
- پروژه‌های با مؤلفه‌های چندگانه: در پروژه‌هایی که مؤلفه‌ها باید به صورت مستقل توسعه و تست شوند.

## 2. Data-Driven Development (DB)

Data-Driven Development یا توسعه داده‌محور، یک متداول‌تر توسعه نرم‌افزار است که به طور عمده بر اساس داده‌ها و تحلیل داده‌ها برنامه‌ریزی می‌شود. در این روش، داده‌های جمع‌آوری شده از کاربران،



Edit with WPS Office

[Date]

5

بازار و دیگر منابع به بهینه‌سازی فرآیند توسعه و بهبود کیفیت و عملکرد نرم‌افزار کمک می‌کنند.

ویژگی‌های کلیدی DB :

۱. استفاده از داده‌ها به عنوان ورودی‌های تصمیم‌گیری: تصمیمات توسعه بر اساس داده‌ها و تجزیه و تحلیل‌های واقعی گرفته می‌شوند.
۲. تست و اندازه‌گیری مداوم: نرم‌افزار به طور مداوم تست و داده‌های عملکرد آن اندازه‌گیری می‌شوند تا تغییرات مورد نیاز اعمال شوند.
۳. بهینه‌سازی براساس رفتار کاربران: بازخورد و رفتار کاربران در بهینه‌سازی و تغییرات نرم‌افزار نقش اصلی را ایفا می‌کند.

مزایای DB :

- تصمیم‌گیری مبتنی بر شواهد: این روش امکان تصمیم‌گیری‌های دقیق و مبتنی بر داده‌ها را فراهم می‌کند.
- افزایش کیفیت و عملکرد: با تجزیه و تحلیل داده‌ها، می‌توان کیفیت و عملکرد نرم‌افزار را بهبود داد.
- کاهش ریسک: استفاده از داده‌ها به شناسایی مشکلات زودتر از موعد و کاهش ریسک توسعه کمک می‌کند.

معایب DB :

- نیاز به داده‌های دقیق و بزرگ: در صورتی که داده‌های کافی و دقیق در دسترس نباشد، این روش ممکن است ناکارآمد باشد.
- پیچیدگی در تحلیل داده‌ها: تجزیه و تحلیل داده‌ها نیازمند دانش و ابزارهای تخصصی است.
- مناسب نبودن برای پروژه‌های کوچک: این روش به داده‌های حجمی و تیم‌های متخصص در داده نیاز دارد که ممکن است در پروژه‌های کوچک وجود نداشته باشد.

کاربردهای DB :

- پروژه‌های مبتنی بر کاربران نهایی: برای توسعه نرم‌افزارهایی که بازخورد کاربران در آن‌ها اهمیت دارد.
- تحلیل و بهینه‌سازی عملکرد سیستم‌ها: مناسب سیستم‌هایی که نیاز به بهینه‌سازی مداوم عملکرد دارند.
- نرم‌افزارهای مبتنی بر رفتار کاربران: در پروژه‌هایی که رفتار کاربران بخش مهمی از فرآیند توسعه است.

### 3. User-Centered Design (UCD)

User-Centered Design یا طراحی کاربر محور یک متداول‌وزی توسعه نرم‌افزار است که به طور عمده بر اساس نیازها، ترجیحات و تجربه کاربران برنامه‌ریزی می‌شود. این روش تلاش می‌کند تا نرم‌افزار را به شکلی طراحی کند که بهترین تجربه کاربری را ارائه دهد.



Edit with WPS Office

[Date]

6

## ویژگی‌های کلیدی UCD :

۱. تمرکز بر تجربه کاربر: فرآیند طراحی و توسعه بر اساس نیازها و انتظارات کاربران تنظیم می‌شود.
۲. تست و بازخورد مستمر: کاربران به صورت دوره‌ای با نرم‌افزار تعامل دارند و بازخورد آن‌ها در هر مرحله جمع‌آوری می‌شود.
۳. فرآیند تکرارپذیر: هر مرحله از طراحی به‌طور پیوسته تکرار و بهبود می‌یابد تا با نیازهای کاربران همگام شود.

## مزایای UCD :

- بهبود تجربه کاربری: تمرکز بر نیازهای کاربران باعث بهبود UX می‌شود.
- کاهش ریسک‌های پذیرش نرم‌افزار: کاربران با مشارکت در توسعه نرم‌افزار، به راحتی نرم‌افزار را می‌پذیرند.
- افزایش رضایت کاربران: بازخورد مستمر کاربران به بهبود کیفیت و افزایش رضایت آن‌ها کمک می‌کند.

## معایب UCD :

- زمان و هزینه بیشتر: جمع‌آوری بازخورد کاربران و تکرار مراحل طراحی ممکن است زمان برو و هزینه‌بر باشد.
- نیاز به تعامل مستمر با کاربران: تعامل و بازخورد کاربران باید به صورت مداوم انجام شود که نیاز به منابع دارد.
- پیچیدگی در مدیریت بازخوردها: هماهنگ کردن بازخوردهای متعدد از کاربران ممکن است پیچیده باشد.

## کاربردهای UCD :

- نرم‌افزارهای B2C : برای نرم‌افزارهایی که مستقیماً با کاربران نهایی تعامل دارند.
- سیستم‌های تعاملی: در سیستم‌هایی که نیاز به تعامل بالای کاربران دارند، مانند وبسایتها و اپلیکیشن‌های موبایل.
- محصولات با تمرکز بر UX : مناسب برای محصولات و نرم‌افزارهایی که تجربه کاربری در آن‌ها اهمیت بالایی دارد.

## 4. Use-Case Driven Development (UDD)

تمرکز اصلی بر توسعه و پیاده‌سازی بر سناریوهای کاربردی، متدولوژی‌ای است که در آن به عنوان نماینده‌ی بخشی از نیازمندی‌ها و عملکرد سیستم در نظر گرفته می‌شود و توسعه بر اساس آن‌ها انجام می‌گیرد.

## ویژگی‌های کلیدی UDD :

۱. تمرکز بر سناریوهای کاربردی: هر بخش از نرم‌افزار با سناریوهای کاربردی تعریف شده و مشخص توسعه داده می‌شود.
۲. ساختاردهی نیازمندی‌ها بر اساس سناریوها: نیازمندی‌ها به صورت مستقیم از سناریوهای کاربردی



استخراج می‌شوند.

3. توسعه به صورت تدریجی: سیستم به صورت تدریجی و سناریو به سناریو تکمیل می‌شود.

: مزایای UDD

- سهولت در پیاده‌سازی نیازمندی‌ها: با تمرکز بر سناریوهای کاربردی، پیاده‌سازی نیازمندی‌ها ساده‌تر است.
- افزایش هم‌خوانی با نیازهای کاربران: سناریوها به طور دقیق نیازهای کاربران را نمایش می‌دهند.
- قابلیت پیگیری و مدیریت آسان: هر سناریو به صورت مستقل پیگیری می‌شود که مدیریت را ساده‌تر می‌کند.

: معایب UDD

- محدودیت در پروژه‌های پیچیده: سناریوهای کاربردی ممکن است تمامی نیازهای پروژه‌های پیچیده را پوشش ندهند.
- نیاز به تعریف دقیق سناریوها: موفقیت این روش وابسته به تعریف دقیق و کامل سناریوها است.
- عدم انعطاف‌پذیری بالا: تغییرات بزرگ ممکن است نیاز به تغییر سناریوها و بازنی کل فرآیند داشته باشد.

: کاربردهای UDD

- پروژه‌های بزرگ و پیچیده: برای پروژه‌هایی که سناریوهای مختلف و گستردگی دارند.
- پروژه‌های نیازمند ساختاردهی مشخص: در پروژه‌هایی که نیازمندی‌های پیچیده‌ای دارند.
- سیستم‌های مبتنی بر فرآیندهای کسب‌وکار: مناسب برای سیستم‌هایی که به پیاده‌سازی دقیق فرآیندهای کسب‌وکار نیاز دارند.

ویژگی‌ها	TDD (Test-Driven Development)	FDD (Feature-Driven Development)	BDD (Behavior-Driven Development)	CDD (Component-Driven Development)	DB (Data-Driven Development)	UCD (User-Centered Design)	UDD (User-Driven Development)
تمرکز اصلی	نوشتن تست قبل از کدنویسی و بهبود کیفیت کد	تحویل ویژگی‌های قابل مشاهده	توصیف رفتار نرم‌افزار با تمرکز بر تعاملات کاربر	استفاده از مؤلفه‌ها و مازول‌های مستقل	توسعه مبتنی بر داده‌ها و تحلیل آن‌ها	نیازها و تجربه کاربری	پیاده‌سازی سناریوهای کاربردی
مزایا	بهبود کیفیت کد	تحویل سریع ویژ	بهبود تعاملات و	افزایش مقیاس‌پذیر	تصمیم‌گیری بهتر و	بهبود تجربه	پیاده‌سازی دقیق



	و کاهش اشکالات	گیها، گیها، افزایش همکاری تیمی	ارتباطات میان تیمها	ی و بهرهوری	دقیقتر بر اساس دادهها	کاربری و کاهش ریسکه‌ها	نیازمندی‌ها و کاربران
معایب	زمان بر بودن تست‌ها، نیاز به دقت بالا	نیاز به برنامه‌ریزی دقیق ویژگی‌ها و امکان کم‌توجهی به کیفیت کلی	نیاز به تعریف سناریوها و همکاری مستمر بین تیمها	نیاز به طراحی دقیق مؤلفه‌ها	نیاز به داده‌های دقیق و بزرگ	هزینه‌بر و زمان بر بودن برای پروژه‌های بزرگ	محدودیت در پروژه‌های پیچیده و نیاز به تعریف دقیق سناریوها
کاربردها	پروژه‌های پیچیده با نیاز به کیفیت بالا	پروژه‌های چابک و تیم‌های توسعه نرم‌افزار	پروژه‌های با نیاز به درک بهتر نیازهای کاربران	برنامه‌های کاربردی مقیاس‌پذیر و مازولار	پروژه‌های مبتنی بر داده و تحلیل کسب‌وکار	نرم‌افزار های تعامل م حور و سیستم‌های B2C	سیستم‌های مبتنی بر فرآیندهای کسب‌وکار و پروژه‌های بزرگ و پیچیده

