موسسه بابان

انتشارات بابان

درس و کنکور ارشد

مهندسی نرم‌افزار

(متدولوژی‌های چابک)

Agile Methodology: XP , Scrum

ویژه‌ی داوطلبان کنکور کارشناسی ارشد مهندسی IT

براساس کتاب مرجع

راجر اس. پرسمن ویراست هشتم ۲۰۱۴

ارسطو خلیلی‌فر

۱۱ متدولوژی‌های چابک

چابکی در مقابل لَختی و سستی قرار دارد. در یک بیان ساده فرز و چابک بودن در انجام پروژه های نرم‌افزاری به معنی سریع‌بودن در انجام کارها است، اما نه به معنی نادیده گرفتن معیارهای موفق پروژه‌های نرم‌افزاری! (برآورده شدن نیازمندی‌های مشتری، مقرون به صرفه بودن و در زمان مورد انتظار آماده شدن)

پشت مجلل‌ترین اتومبیل هم که باشی، اما چابک نباشی، تصادف ممکن است، مرگبار باشد. چابکی یک هنر است، هنر استفاده از امکانات موجود به عالی‌ترین شیوه‌ی ممکن، هنر انجام صحیح یک راه‌حل، اما به شیوه‌ای هوشمندانه و پُرسرعت، بدون آنکه چیزی از قلم بیفتد.

مهندسان نرم‌افزار، ربات نیستند، آن‌ها تفاوت و شباهت‌هایی دارند، بعضی از آن‌ها نسبت به بعضی دیگر چابک‌تر هستند، و این نشأت گرفته از ماهیت انسانی بودن آن‌ها است، برخی باهوش تر و سریع‌تر هستند و این کاملاً مشهود است. در یک بیان ساده، چابکی، کاتالیزور فرآیند تولید نرم‌افزار است. بنابراین، چابکی، مقدمه می‌خواهد. یعنی باید مقدماتی فراهم گردد تا پروژه چابک پیش رود. برای‌مثال سازنده و مشتری هر دو چابک باشند.

پروژه چابک	=	مشتری چابک	+	سازنده چابک

به طور کلی شرایط چابکی و مقدمات چابکی عبارتند از:

مدل فرآیند تولید چابک: این مدل باید براساس رویکرد تکرار و تکامل و مبتنی بر مؤلفه شیء گرا باشد تا سازنده و مشتری روزانه و پیوسته با یکدیگر در ارتباط باشند و همچنین سبب شود تا رضایت مشتری از طریق تحویل نسخه‌های محصول به طور پیوسته و سریع با بالاترین اولویت برآورده گردد. همچنین مدل فرآیند تولید نرم‌افزار، شامل فعالیت‌های چارچوبی به عنوان یک نقشه ی حرکت، جهت هدایت و پیشرفت پروژه استفاده می‌گردد، و اینکه هر یک از فعالیت‌های چارچوبی شامل مراحل ارتباط، برنامه‌ریزی، مدل‌سازی (تحلیل و طراحی)، ساخت (پیاده‌سازی و

تست) و استقرار به چه شکل و با چه حدی از چابکی انجام گردد، در چابک بودن روند پیشرفت پروژه مؤثر است و نه ادغام و ترکیب فعالیت‌های چارچوبی همچون ادغام فعالیت‌های طراحی و ساخت.

روش چابک: روش شیءگرایی به دلیل استفاده از مفاهیمی همچنین کلاس به عنوان یک مؤلفه قابل استفاده مجدد، وراثت به عنوان یک عامل بازدارنده از دوباره‌نویسی کد، چندریختی به عنوان یک عامل بالابرنده خوانایی در برنامه، می‌تواند شکل و تمایل چابکی را به خود بگیرد. در روش ساخت‌یافته، چابکی خیلی کم‌تر دیده می‌شود.

توجه کنید که شیءگرایی و مفاهیم کلاس باعث می‌شود تا بتوان مؤلفه‌هایی با طراحی ایده‌آل با همبستگی بالا (Cohesion) و پیوستگی یا اتصال پایین (Coupling) در فعالیت طراحی مربوط به فعالیت‌های چارچوبی ایجاد کرد. هرچه قدر در طراحی مؤلفه‌ها همبستگی بالا و پیوستگی پایین باشد، فرآیند توسعه‌ی نرم‌افزار سریع‌تر خواهد بود و این یعنی چابکی.

ابزار چابک: UML، Nassi–Shneiderman diagram، Warnier-Orr Diagram، Petri net و SysML به دلیل مدل‌سازی خاص و منحصر به فرد خود می‌توانند شکل و شمایل چابکی را به خود بگیرد. مانند تولید کد سریع از مدل‌های طراحی ایجاد شده توسط UML به کمک رشنال رُز.

همانطور که پیش‌تر نیز گفتیم، متدولوژی شیءگرا یا مهندسی نرم‌افزار شیءگرا نظامی است یکپارچه شامل مدل فرآیند شیءگرا (مدرن)، روش شیءگرا (مبتنی بر مفاهیم کلاس، وراثت و چندریختی) و ابزارهای شیءگرا که منجر به ایجاد نرم‌افزاری در بازه‌ی زمانی از قبل برنامه‌ریزی شده، بودجه‌ای از قبل پیش‌بینی شده و دقیقاً مطابق با نیازمندی‌های واقعی مشتری می‌گردد.

توجه: متدولوژی RUP متداول‌ترین نمونه از متدولوژی شیءگرا براساس روش شیءگرا (مبتنی بر مفاهیم کلاس، وراثت و چندریختی)، مدل فرآیند مبتنی بر مؤلفه‌ی شیءگرا با رویکرد تکرار و تکامل و ابزارهای شیءگرا (مثل ابزار مدل‌سازی UML و زبان برنامه‌نویسی C++) می باشد.

توجه: نسبت نمونه متدولوژی شیءگرای RUP به متدولوژی شیءگرا، مثل نسبت سیستم عامل ویندوز مدرن به مفاهیم مدرن سیستم عامل است.

اما متدولوژی شیءگرای چابک یا مهندسی نرم‌افزار شیءگرای چابک نظامی است یکپارچه شامل مدل فرآیند شیءگرای چابک (مدرن‌تر و سریع‌تر)، روش شیءگرای چابک (مبتنی بر مفاهیم کلاس، وراثت و چندریختی) و ابزارهای شیءگرای چابک که منجر به ایجاد نرم‌افزاری در بازه‌ی زمانی از قبل برنامه‌ریزی شده، بودجه‌ای از قبل پیش‌بینی شده و دقیقاً مطابق با نیازمندی‌های واقعی مشتری به شکلی چابک می‌گردد. این تعریف، فرآیند تولید پروژه‌های نرم‌افزاری را به سمت موفقیت و چابکی سوق می‌دهد.

توجه: متدولوژی‌های XP[۱]، Scrum، Crystal، ASD[۲]، DSDM[۳] و AUP[۴] متداول‌ترین نمونه از متدولوژی‌های شیءگرای چابک براساس روش شیءگرای چابک (مبتنی بر مفاهیم کلاس، وراثت و چندریختی)، مدل فرآیند مبتنی بر مؤلفه‌ی شیءگرای چابک با رویکرد تکرار و تکامل و ابزارهای شیءگرای چابک (مثل ابزار مدل‌سازی UML، Nassi–Shneiderman diagram، Warnier-Orr Diagram، Petri net، SysML و زبان برنامه‌نویسی C++ و C#) می‌باشد.

توجه: نسبت نمونه متدولوژی‌های شیءگرای چابک XP، Scrum، Crystal، ASD، DSDM و AUP به متدولوژی شیءگرای چابک، مثل نسبت سیستم عامل ویندوز مدرن‌تر و سریع‌تر به مفاهیم مدرن‌تر و سریع‌تر سیستم عامل است.

متدولوژی‌های چابک از طریق یک فرآیند تکرار شونده، به تحویل زودهنگام نرم‌افزار در قالب افزایش‌های مختلف و جلب حداکثری رضایت مشتری می‌پردازند. هر افزایش یک نسخه اجرایی از نرم‌افزار است که در طی فرآیند نرم‌افزار به شکل تکرارشونده به مشتری تحویل می‌شود. با گذشت زمان افزایش‌های نرم‌افزار، کامل و کامل‌تر شده تا به شکل نرم‌افزار نهایی تبدیل شود. در رویکرد مبتنی بر تکرار و تکامل، فرصت یادگیری و بهبود تدریجی در سرتاسر چرخه‌ی تولید فراهم است. بدین ترتیب، در طول پروژه، امکان تصحیح به موقع اشتباهات وجود خواهد داشت. در صورت بروز اشتباه در یک تکرار، امکان جبران آن در تکرار بعدی وجود دارد. رویکرد مبتنی بر تکرار و تکامل به برنامه‌ریزی مستمر و پویا در طول پروژه نیازمند است. رویکرد مبتنی بر تکرار و تکامل در هر تکرار نسخه‌هایی از نرم‌افزار را ارائه می‌دهد که هریک از قبلی کامل‌تر است. به بیان دیگر در مدل‌های تکاملی در هر دور از تکرار نسخه‌ی کامل‌تری از نرم‌افزار تولید می‌شود. در متدولوژی‌های چابک، تحویل سریع‌تر نرم‌افزار ترجیح بیشتری به انجام مراحل تحلیل و طراحی دارد، اگرچه به هیچ وجه نباید از انجام مراحل تحلیل و طراحی صرف نظر کرد. علت اصلی استفاده از متدولوژی‌های چابک این است که نرم‌افزارهای جدید مدام درحال تغییر و تحول هستند، علت این تغییرات مکرر، نیاز نرم‌افزار جهت تطابق با قوانین و نیازهای محیط عملیاتی خودش و یا پیشرفت تکنولوژی است. متدولوژی‌های چابک ضمن ارائه‌ی زودهنگام نرم‌افزار، نسبت به تغییرات نرم‌افزار نیز به دلیل رویکرد تکرار و تکامل بسیار منعطف است. در متدولوژی‌های چابک فعالیت‌های توسعه‌ی نرم‌افزار با جزئیات کمتری نسبت به متدولوژی‌های قدیمی‌تر انجام می‌شود. در واقع تنها وظایف ضروری حاضر در فعالیت‌های اصلی یا framework activity مورد توجه قرار گرفته و از انجام دیگر وظایف غیرضروری چشم پوشی

---

[۱] Extreme Programming

[۲] Adaptive Software Development

[۳] Dynamic System Development Method

[۴] Agile Unified Process

می‌شود. به همین دلیل نام دیگری که برای متدولوژی‌های چابک پیشنهاد می‌شود، مهندسی نرم‌افزار سبک یا software engineering lite است.

در بحث مدیریت منابع انسانی و غیرانسانی نیز، این مدیریت باید چابک باشد، به عبارت دیگر پروژه‌های نرم‌افزاری به واسطه فعالیت‌های چارچوبی و فعالیت‌های چتری ایجاد می‌گردند. به بیان دیگر مطابق آنچه پیش از این نیز گفتیم نه تنها فعالیت‌های چارچوبی (فرآیند تولید نرم‌افزار) باید چابک باشد، بلکه فعالیت‌های چتری نیز باید چابک باشد، برای‌مثال یکی از فعالیت‌های چتری، پشتیبان‌گیری از منابع انسانی (مانند مدیران رده بالا) و غیرانسانی (مانند اطلاعات پروژه) است که باید به شکلی چابک و سریع و تا دیر نشده است انجام گردد.

همچنین در مورد تیم‌های متدولوژی‌های چابک باید گفت که این تیم‌ها به صورت خود سازمانده (Self-Organizing) تشکیل شده و اعضای آن‌ها با اختیارات بیشتری نسبت به دیگر تیم های نرم‌افزاری عمل می‌کنند. که این خود سازماندهی سبب چابکی تیم‌های کاری می‌گردد. تیم خودسازمانده خودش کنترل کارهایش را برعهده دارد. این تیم خودش به تعهداتش عمل می‌کند و طرح‌هایی برای انجام آنها تعریف می‌کند. هیچ چیز به این اندازه انگیزه را در یک تیم از بین نمی‌برد که آدم دیگری برایش تعیین تکلیف کند و هیچ چیز به اندازه‌ی پذیرش مسئولیت برای تعیین وظایف و انجام تعهدات در تیم ایجاد انگیزه نمی‌کند.

باز هم تأکید می‌کنیم که چابکی، مقدمه می‌خواهد، مقدمات چابکی باید فراهم گردد، تا پیشرفت پروژه چابک باشد، در یک بیان ساده، طوری تیم‌ها و کارها را سازماندهی کنید که همه چیز چابک و سریع باشد و عاملی نتواند روند رو به جلوی پیشرفت پروژه را مختل نماید، به هر دلیلی.

در اقتصاد نوین، پیش‌بینی چگونگی تکامل یافتن یک سیستم کامپیوتری، برای مثال یک برنامه‌ی کاربردی مبتنی بر وب، در گذر زمان، غالبا کاری دشوار است. شرایط بازار به سرعت تغییر می‌کند، نیازهای کاربران نهایی تکامل می‌یابد و تهدیدهای رقابتی بدون هشدار قبلی ظهور می‌کند. در بسیاری شرایط، قادر به تعریف کامل خواسته‌ها قبل از شروع پروژه نخواهید بود. باید به قدر کافی چابک باشید تا بتوانید به یک محیط عملیاتی متغییر پاسخ دهید.

تغییر، هزینه‌بردار است. به ویژه اگر کنترل نشده باشد یا مدیریت ضعیفی بر آن اعمال شود. یکی از بارزترین ویژگی‌های متدولوژی‌های چابک، توانایی آن در کاهش دادن هزینه‌های ناشی از تغییر در سرتاسر فرآیند نرم‌افزار است.

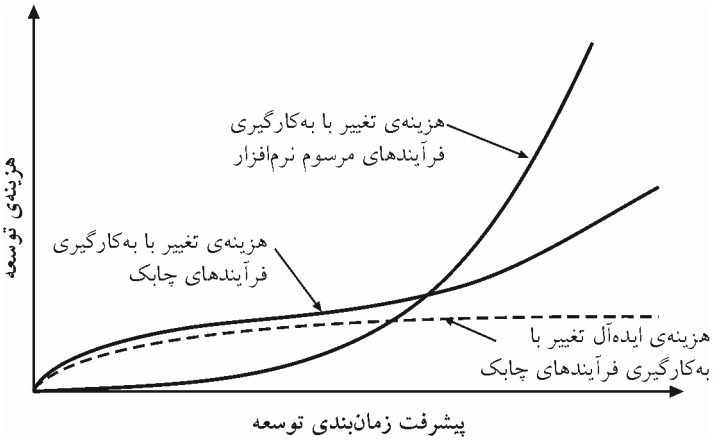
آیا این شرایط بدان معناست که شناخت چالش‌های پدید آمده از یک واقعیت جدید، باعث می‌شود که همه‌ی اصول، مفاهیم، روش‌ها و ابزارهای ارزشمند مهندسی نرم‌افزار را نادیده بگیرید؟ مطقا خیر! مهندسی نرم‌افزار نیز همانند کلیه رشته‌های مهندسی به تکامل خود ادامه می‌دهد و می‌توان آن را طوری تطبیق داد که چالش‌های ناشی از تقاضا برای سرعت را نیز پاسخگو باشد.

این روزها هنگام توصیف یک فرآیند نرم‌افزار مدرن‌تر، **چابکی** واژه‌ای است که به وفور به

گوش می‌رسد. تیم چابک تیمی فرز و چالاک است که قادر است به تغییرات پاسخ مناسب بدهد. تغییر چیزی است که در توسعه نرم‌افزار، بسیار با آن مواجه می‌شویم. تغییرات در نرم‌افزارهایی که درحال ساخته‌شدن هستند، تغییرات در اعضای تیم، تغییرات به دلیل تکنولوژی جدید، و یا هر نوع تغییری که ممکن است بر محصول درحال ساخت یا محصولی که محصول نهایی را ایجاد می‌کند، تاثیرگذار باشند. هرآنچه در نرم‌افزار انجام می‌دهیم باید خود حاوی ویژگی‌هایی برای پشتیبانی از تغییرات باشد، چیزی که قلب و روح نرم‌افزار است. تیم چابک می‌داند که نرم‌افزار توسط افرادی توسعه می‌یابد که در قالب تیمی کار می‌کنند و مهارت‌های این افراد و توانایی ایشان در **همکاری**، هسته‌ی اصلی موفقیت پروژه است. چابکی، بیشتر خاصیت خود را مرهون دانش نهفته در تیم است و نه در دانش نوشته شده روی کاغذ.

چابکی و هزینه‌های تغییر

عقل سلیم در توسعه نرم‌افزار که چند دهه تجربه پشتیبان آن است، حکایت از آن دارد که هزینه‌ی تغییر به صورت غیرخطی با پیشرفت پروژه افزایش می‌یابد. پاسخگویی به تغییر در فعالیت ارتباطات یعنی هنگامی‌که تیم نرم‌افزاری مشغول جمع‌آوری خواسته‌هاست، نسبتا آسان و با هزینه اندک است. و زمان مورد نیاز برای پاسخ به این نوع تغییرات تاثیری عمیق بر زمان‌بندی پروژه نخواهد داشت. اما اگر چند ماه جلوتر برویم، چطور؟ فرض کنید تیم درحال انجام فعالیت تست است، چیزی که نسبتا در اواخر پروژه رخ می‌هد و ناگهان مشتری درخواست تغییری عمده در قابلیت‌های سیستم را دارد. این تغییر نیاز به اصلاح لیست نیازمندی‌ها، مدل تحلیل، مدل طراحی، پیاده‌سازی و تست دارد، برای‌مثال ساخت سه قطعه جدید و اصلاح و ویرایش پنج قطعه قدیم، و به تبع موردهای تست جدید نیز باید طراحی شوند. در این شرایط هزینه‌ها به سرعت بالا می‌رود و زمان لازم برای حصول اطمینان از اینکه تغییرات بدون برجای گذاشتن اثرات جانبی اعمال شود، قابل چشم پوشی نخواهد بود.



یک فرآیند چابک با رعایت اصول طراحی معماری ایده‌آل (پنهان‌سازی اطلاعات، افزایش انسجام، کاهش اتصال و استقلال عملیاتی) می‌تواند، هزینه تغییر را **تسطیح** کند. به این ترتیب، تیم نرم‌افزاری قادر به پاسخگویی به تغییرات در اواخر پروژه خواهد بود، بدون اینکه ضربه‌ای قابل ملاحظه از نظر زمان و هزینه به پروژه وارد آید. همانطور که پیش‌تر گفتیم، متدولوژی‌های چابک شامل تحویل افزایشی محصول می‌شود. هنگامی‌که تحویل افزایشی با سایر توصیه‌های چابک از قبیل تست واحد مستمر و برنامه‌نویسی زوجی (دو نفری بر روی یک کامپیوتر) تلفیق شود، زمان و هزینه اعمال تغییرات بیشتر هم کاهش می‌یابد.

فرآیند چابک

فرآیند چابک باید از انطباق‌پذیری برخوردار باشد. ولی انطباق پیوسته و بدون پیشروی از موفقیت چندانی برخوردار نیست. بنابراین، در یک فرآیند نرم‌افزار چابک، باید روند انطباق به طور افزایشی انجام پذیرد. برای دستیابی به انطباق افزایشی و تدریجی، تیم نرم‌افزاری چابک نیاز به بازخورد از مشتری دارد، تا بتواند انطباق‌های لازم را انجام دهد. یک عامل شتاب‌دهنده به دریافت بازخورد مشتریان، نمایش نمونه‌ای اولیه از سیستم عملیاتی به مشتری است. از این رو، راهبرد توسعه افزایشی را باید نهادینه ساخت. نسخه‌های نرم‌افزار یعنی نمونه‌های اولیه قابل اجرا یا بخش‌هایی از سیستم عملیاتی باید در دوره‌های زمانی کوتاه مدت به مشتری تحویل داده شوند تا روند انطباق بتواند همگام با روند تغییرات ادامه یابد. مشتری به کمک این رویکرد مبتنی بر تکرار و تکامل می‌تواند، هر نسخه از نرم‌افزار را مرتبا ارزیابی کند، بازخوردهای لازم را برای تیم نرم‌افزاری فراهم سازد و بر انطباق‌های به عمل آمده جهت پاسخگویی به بازخوردها تاثیر بگذارد.

اصول چابکی

در پیمان چابک (Agile Alliance) دوازده اصل برای کسانی که می‌خواهند به چابکی دست پیدا کنند، تعریف شده است.

۱- جلب رضایت مشتری از طریق تحویل زودهنگام و پیوسته نرم‌افزارهای ارزشمند، بیشترین اولویت را نزد ما دارد.
۲- پذیرا بودن تغییرات در خواسته‌ها حتی در اواخر فرآیند توسعه. که مزیتی رقابتی مابین شرکت‌های سازنده نرم‌افزار است.
۳- تحویل پیوسته‌ی نرم‌افزارهای کاری از دو هفته گرفته تا دو ماه، که بازه‌های زمانی کوتاه‌تر باید در اولویت قرار داده شوند.
۴- دست‌اندرکاران و افراد تجاری باید در سرتاسر پروژه هر روز باهم کار کنند.
۵- سپردن پروژه به افراد باانگیزه، فراهم‌سازی محیط و پشتیبانی مورد نیاز آنها و اطمینان‌کردن به آنها در انجام کارها.

۶- اثربخش‌ترین و موثرترین روش انتقال اطلاعات به درون و بیرون تیم توسعه، گفتگوی رو در رو است.
۷- نرم‌افزاری که به درستی کار کند، میزان اصلی در سنجش پیشرفت کاری است.
۸- فرآیندهای چابک، توسعه پایدار را ارتقا می‌بخشند. سازندگان و مشتریان باید قادر به حفظ سرعت ثابت در مسیر پیشرفت کار باشند.
۹- توجه پیوسته به اعتلای فنی و طراحی معماری ایده‌آل (پنهان‌سازی اطلاعات، افزایش انسجام، کاهش اتصال و استقلال عملیاتی)، باعث افزایش چابکی می‌شود.
۱۰- ساده‌سازی ضروری است، یعنی هنر به انجام رساندن کارهای انجام ناپذیر.
۱۱- بهترین دستاورد از فعالیت‌های چارچوبی(ارتباطات، برنامه‌ریزی، مدل‌سازی، ساخت و استقرار) از تیم‌های خود سازمانده ظهور می‌کنند. به بیان دیگر بهترین تحلیل‌ها، طراحی ها، معماری‌ها و شناسایی نیازها از تیم‌های خود سازمانده منتج می‌شود.
۱۲- تیم در بازه‌های زمانی منظم، بازخوردی از میزان بهبود اثربخشی خود ارائه می‌دهد و سپس رفتار خود را مطابق این بازخورد تنظیم می‌کند.

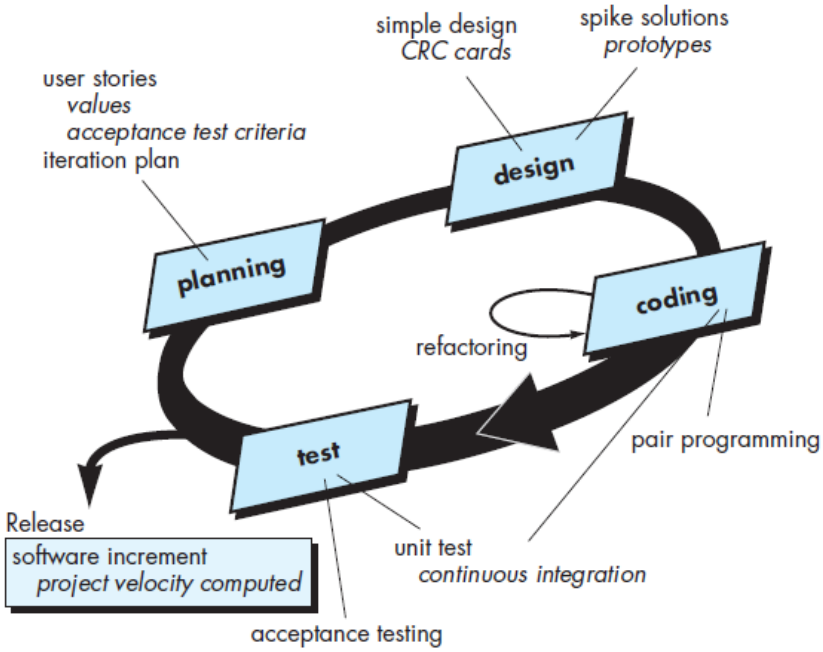
این دوازده اصل در تمامی فرآیندهای چابک با وزن مساوی به کار برده نمی‌شوند و در برخی مدل‌ها از اهمیت یک یا چند اصل چشم پوشی می‌شود یا دست کم نقش آنها کمرنگ‌تر می‌شود. این اصول تعیین کننده هسته و جوهره چابکی هستند که در مدل‌های فرآیند چابک اغلب حفظ خواهند شد.

توجه: چابکی به معنی نادیده‌گرفتن مهندسی نرم‌افزار نیست. نباید بین چابکی و مهندسی‌نرم‌افزار یکی را انتخاب کرد، بلکه باید یک رویکرد مهندسی نرم‌افزار انتخاب شود که چابک هم باشد! اما روش‌شناسان **سنتی** یک مُشت آدم‌های فاقد خلاقیت هستند که ترجیح می‌دهند مستنداتی بدون نقص تهیه کنند تا اینکه سیستمی کاری ارائه دهند که نیازهای مشتری را برآورده سازد. مکتب چابکی به معنی خروج از مکتب مهندسی نرم‌افزار نیست، بلکه اتفاقا چابکی به معنی ورود و دقت بیشتر بر اصول اساسی و طراحی معماری ایده‌آل مهندسی نرم‌افزار است اما اینبار با سرعت و چابکی بیشتر. با در نظر گرفتن بهترین ایده‌ها از هر دو مکتب، بیشترین بهره عاید خواهد شد و چیزی از تخریب دیگری، به‌دست نخواهد آمد.

متدولوژی XP

متدولوژی XP یا متدولوژی برنامه‌نویسی حدی (Extreme Programming) پرکابردترین رویکرد در توسعه نرم‌افزار به شیوه چابک است. این متدولوژی با نام‌های XP و یا X-programming نیز شناخته می‌شود. علت نامگذاری این متدولوژی به برنامه‌نویسی حدی این است که نسبت به متدولوژی‌های دیگر، مرحله‌ی برنامه‌نویسی را با تاکید بیشتری انجام می‌دهد. متدولوژی XP از روش شیءگرا جهت توسعه برنامه استفاده می‌کند. در این متدولوژی فعالیت‌های

چارچوبی (framework activities) شامل چهار فعالیت برنامه‌ریزی، طراحی، برنامه‌نویسی و تست می‌باشد. به تازگی، شکل دیگری از XP موسوم به IXP (Industrial Extreme Programming) به معنی XP صنعتی پیشنهاد شده است. IXP پالایشی از XP است که فرآیند چابک را مشخصا برای استفاده در سازمان‌های بزرگ هدف قرار داده است. شکل زیر گویای روند کارکرد متدولوژی XP است:



برنامه‌ریزی (Planning)

فعالیت برنامه‌ریزی با گوش‌سپردن (listening) به خواسته‌های مشتری توسط سازنده طی جلساتی غیررسمی و شفاهی مابین سازنده و مشتری آغاز می‌شود. این جلسات منجر به بیان خواسته‌ها توسط مشتری و درک و جمع‌آوری خواسته‌ها توسط سازنده می‌شود. خواسته‌های شفاهی مشتری در قالب use caseها قرار می‌گیرند، سپس در ادامه برای هر use case (مورد کاربرد یا نیاز) سناریو یا شرح حال نوشته می‌شود.

سناریو بر دو طبقه اصلی و فرعی می‌باشد:

سناریوی اصلی: بیان روال حرکت قدم به قدم کارها داخل یک use case از نقطه شروع تا رسیدن به یک نتیجه مطلوب (موفق). (حرکت از خود موجود و رسیدن به خود مطلوب).

سناریوی فرعی: بیان روال حرکت قدم به قدم کارها داخل یک use case از نقطه شروع تا

رسیدن به یک نتیجه نامطلوب (ناموفق). (حرکت از خود موجود و رسیدن به خود نامطلوب).

توجه: هر use case، فقط و فقط یک سناریوی اصلی دارد و می‌تواند چندین سناریوی فرعی داشته باشد. سناریوها بر روی کارت شاخص (Index Card) نوشته می‌شوند.

در سناریوهای اصلی، خروجی‌ها و ورودی‌های نرم‌افزار، خصوصیات نرم‌افزار و کارکردهای نرم‌افزار مشخص می‌شود. مشتری باید برای هر سناریو اولویتی را مشخص کند. از آنجاکه متدولوژی XP بر پایه رویکرد تکرار و تکامل است، این اولویت‌بندی سناریوها توسط مشتری، منجر به شناخت اولویت پیاده‌سازی سناریوها توسط سازنده در هر تکرار و افزایش می‌شود، ضمن اینکه در هر تکرار می‌توان سناریوهای جدیدی به پروژه اضافه نمود و یا سناریوهایی از پروژه حذف کرد. پس از درک سناریوهای مشتری توسط سازنده و اولویت‌بندی آنها توسط مشتری، سازنده هزینه و زمان لازم جهت پیاده‌سازی هر سناریو را بر حسب تعداد هفته‌های لازم مشخص می‌کند. اگر تعداد هفته‌های لازم جهت توسعه هر سناریو بیش از سه هفته بود، از مشتری خواسته می‌شود که سناریوی مورد نظر را به سناریوهای کوچکتر تقسیم کند و دوباره آنها را اولویت‌بندی کند. سازنده و مشتری با توافق یکدیگر سناریوهایی که در یک تکرار باید توسعه یابد را اولویت‌بندی و مشخص می‌کنند، اما ترتیب توسعه سناریوها در یک تکرار بر حسب شرایط پروژه بر اساس یکی از سه رویکرد زیر تعیین می‌شود:

۱- همه سناریوهای موجود در یک تکرار پیاده‌سازی شود.

۲- برخی از سناریوهای موجود در یک تکرار پیاده‌سازی شود، به این شکل که ابتدا سناریوهایی با اولویت بالاتر پیاده‌سازی شوند. بنابراین اگر یک تکرار یا افزایش تحت فشار زمان‌بندی قرار گرفت، این مزیت وجود دارد که سناریوهای پراولویت‌تر پیاده‌سازی شده‌اند.

۳- برخی از سناریوهای موجود در یک تکرار پیاده‌سازی شود، به این شکل که ابتدا سناریوهایی که ریسک بیشتری دارند پیاده‌سازی شوند. بنابراین اگر یک تکرار یا افزایش تحت فشار زمان‌بندی قرار گرفت، این مزیت وجود دارد که سناریوهای پرریسک‌تر پیاده‌سازی شده‌اند.

همانطور که گفتیم، متدولوژی XP بر اساس رویکرد تکرار و تکامل پیش می‌رود، بنابراین اندازه‌گیری‌های انجام‌شده بر روی سناریوهای انجام شده در تکرار و افزایش قبلی، مبنایی برای کار برنامه‌ریزی شامل فعالیت‌های «برآورد میزان کار»، «برآورد زمان لازم برای انجام کار»، «برآورد هزینه لازم برای انجام کار»، «مدیریت ریسک» و «زمان‌بندی» در افزایش و تکرار بعدی است. آینده نزدیک، بسیار شبیه به گذشته نزدیک است.

طراحی (Design)

طراحی در متدولوژی XP برپایه‌ی سادگی بنا شده است، این اصل که سادگی را حفظ‌کن

(keep it simple). در متدولوژی XP همواره یک طراحی ساده بر یک طراحی پیچیده برتری دارد. در این متدولوژی صرفا طراحی سناریوهایی انجام می‌شود که نیاز وضع موجود است و قرار به پیاده‌سازی قطعی آن است، بنابراین طراحی سناریوهایی که نیاز به آن در آینده محتمل است و قرار به پیاده‌سازی آن قطعی نیست، در این متدولوژی جایگاهی ندارد. متدولوژی XP در جهت حفظ سادگی در فعالیت طراحی خود فقط از کارت‌های CRC استفاده می‌کند. پس از شناسایی موارد کاربرد و سناریونویسی برای هر یک از موارد کاربرد، زمان تعریف کلاس‌ها، صفات، متدها و ارتباطات میان کلاس‌های همکار برای هر یک از موارد کاربرد می‌رسد. برای شناسایی کلاس‌ها، صفات، متدها و ارتباطات میان کلاس‌ها برای هر یک از موارد کاربرد، از تکنیکی موسوم به CRC که سرواژه‌ی عبارت Class – Responsibility Collaborator و به معنی «مدل همکاری مسئولیت های کلاس‌ها» می‌باشد، استفاده می‌شود. مدل‌سازی CRC روشی ساده جهت تعیین و سازماندهی کلاس‌های داخل هر مورد کاربرد است. اگر در بخشی از طراحی یک سناریو، سختی ایجاد شود، متدولوژی XP، ایجاد فوری یک نمونه اولیه عملیاتی (operational prototype) را برای آن بخش از طراحی توصیه می‌کند. استفاده از نمونه اولیه عملیاتی که موسوم به راهکار خیزشی (spike solution) نیز می‌باشد، دو پیامد دارد، اول اینکه، این راهکار منجر به کاهش ریسک فنی در پیاده‌سازی واقعی سناریوی مورد نظر می‌شود و دوم اینکه، این راهکار منجر به اعتبارسنجی برآوردهای اولیه یعنی «برآورد میزان کار»، «برآورد زمان لازم برای انجام کار»، «برآورد هزینه لازم برای انجام کار»، «مدیریت ریسک» و «زمان‌بندی» مربوط به سناریوی مورد نظر می‌شود.

کدنویسی (Coding)

پس از کشف سناریوهای مشتری و انجام‌شدن کارهای طراحی مقدماتی، تیم توسعه به کدنویسی نمی‌پردازد، بلکه یک سری «آزمون واحد» تهیه می‌کند تا هریک از سناریوهایی که قرار است در نسخه جاری یعنی تکرار فعلی گنجانده شود، مورد تست و بررسی قرار گیرد. هنگامی‌که آزمون واحد تهیه شد، سازنده بهتر می‌تواند توجه خود را به آن چیزی معطوف کند که باید پیاده‌سازی شود تا آزمون را با موفقیت پشت سر بگذارد. هنگامی‌که کدها کامل شدند، می‌توان تست واحدی را بلافاصله انجام داد و در نتیجه، بازخوردی فوری را حاصل کرد.

یکی از اصلی‌ترین مفاهیم متدولوژی XP، برنامه‌نویسی زوجی (pair programming) است که در مرحله کدنویسی قرار دارد. متدولوژی XP توصیه می‌کند که برای تولید کد هر سناریو، دو نفر از اعضای تیم توسعه بر روی یک کامپیوتر اقدام به کدنویسی نمایند. زیرا دو فکر غالباً بهتر از یک فکر است، مانند حضور مؤلف، هنگام تایپ یک کتاب، که به کشف زود هنگام خطاها کمک می کند. درواقع زمانی‌که برنامه توسط یکی از برنامه‌نویسان نوشته می‌شود، برنامه‌نویس دیگر در همان لحظه کد را با هدف کشف خطاهای آن بازبینی (review) می‌کند. که به آن تضمین کیفیت بی درنگ نیز گفته می‌شود.

به موازاتی که برنامه‌نویسان زوجی کار خود را کامل می‌کنند، کدی که می‌نویسند در کنار کار دیگران قرار داده می‌شود. در برخی موارد، این کار به صورت روزانه توسط تیم یکپارچه‌کننده (جامعیت) انجام می‌شود. در موارد دیگر، برنامه‌نویسان زوجی، مسئولیت جامعیت تدریجی (افزایشی) را نیز برعهده دارند. این راهبرد یعنی «جامعیت تدریجی پیوسته» به پرهیز از مشکلات سازگاری و ایجاد واسط کمک می‌کند و یک محیط تست دود (smoke testing) فراهم می‌سازد که به کشف زودهنگام خطاها کمک می‌کند.

همانطور که پیش‌تر گفتیم، طراحی در متدولوژی XP برپایه‌ی سادگی بنا شده است، بنابراین این دغدغه وجود دارد که گاهی طراحی‌های XP، از کیفیت الگوریتمی (غیروظیفه‌مندی یا مزه) مناسبی برخوردار نباشد. متدولوژی XP برای مرتفع نمودن این دغدغه، روش فاکتورگیری مجدد یا بازآرایی (refactoring) را پیشنهاد می‌کند. بنابراین با وجود اینکه روش فاکتورگیری مجدد در مرحله کدنویسی انجام می‌شود اما ماموریت آن ارتقاء کیفیت الگوریتمی و خوشمزه‌سازی طراحی است. فاکتورگیری مجدد یا بازآرایی، فرآیند تغییر، بهبود و ارتقاء کیفیت کد است، بازآرایی، کد را خواناتر می‌کند، بازآرایی، کد را تغییرپذیرتر می‌کند، بازآرایی، ساختار داخلی کد را بهبود می‌دهد اما بدون آنکه بازآرایی، رفتار بیرونی کد را اصلاح کند، به عبارت دیگر بازآرایی، رفتار بیرونی کد را اصلاح نمی‌کند یعنی ورودی‌ها و خروجی‌های کد را تغییر نمی‌دهد.

بازآرایی، یک روش مناسب جهت اصلاح، ساده‌سازی و بهبود کد است. این کار احتمال وجود خطا در کد را کاهش می‌دهد. واضح است که عمل بازآرایی پس از تولید کد باید انجام شود. بازآرایی منجر به این می‌شود که علاوه بر بهبود ساختار داخلی کد، فعالیت طراحی نیز در طول فعالیت کدنویسی دستخوش تغییر، تحول و بهبود شود. یعنی هم قبل از کدنویسی و هم بعد از کدنویسی طراحی درحال انجام است. شعار طراحی قبل از کدنویسی «سادگی» است، اما شعار طراحی بعد از کدنویسی «کیفیت» است. و این کدنویسی است، که راهنمایی لازم برای چگونگی بهبود بخشیدن به طراحی را فراهم می‌سازد.

تست (Testing)

شروع زودهنگام و غیررسمی آزمون واحد و آزمون جامعیت (یکپارچگی) در مرحله کدنویسی است، اما ادامه آن و شروع رسمی آزمون واحد، آزمون جامعیت (یکپارچگی) و آزمون اعتبارسنجی در مرحله تست است. آزمون واحد **صحت عملکرد یا روند اجرای** یک بخش، یک جزء یا یک مولفه از نرم‌افزار را مستقل از سایر مولفه‌های دیگر برنامه مورد تست و ارزیابی قرار می‌دهد. آزمون جامعیت یا یکپارچه‌سازی، اعتبارسنجی عملکرد یا روند اجرای چندین مولفه در کنار یکدیگر و یا کل مولفه‌های نرم‌افزار ایجاد شده در کنار یکدیگر به همراه **نقاط اتصال** مابین آن‌ها را مورد تست و ارزیابی قرار می‌دهد. در یک نرم‌افزار ممکن است هر یک از مؤلفه‌های برنامه به تنهایی درست عمل کنند اما ترکیب آنها با یکدیگر به درستی عمل نکند. زیرا ممکن است به دلیل

عدم تنظیم درست نقاط اتصال و ارتباطی مابین مولفه‌ها، این مولفه‌ها قادر نباشند در کنار یکدیگر، کارکرد مورد انتظار را نشان دهند. بنابراین آزمون جامعیت، جهت تست تدریجی یکپارچه‌شدن مولفه‌های برنامه در کنار یکدیگر مورد استفاده قرار می‌گیرد. منظور از نقاط اتصال مابین مولفه‌های مرتبط با هم، همان کانال یا محل تبادل داده مابین مولفه‌های مرتبط با هم هست، مانند فراخونی با مقدار یا فراخوانی با ارجاع. اگر این نقاط اتصال مابین مولفه‌های مرتبط با هم درست تنظیم نشوند ممکن است داده‌ها در گذر از این نقاط اتصال از بین بروند مانند حالتی که یک متغیر دوبایتی به یک متغیر یک بایتی در روش فراخوانی با مقدار پاس داده می‌شود، در واقع در این حالت بخشی از داده‌ها از دست رفته است. این نقاط اتصال به واسط مولفه نیز موسوم است. واضح است که، برطرف کردن مشکلات کوچک در هر چند ساعت یکبار، نسبت به برطرف‌کردن مشکلات بزرگ درست قبل از پایان مهلت، زمان کمتری می‌برد، بنابراین بهتر است آزمون جامعیت به شکل روزانه و به روش آزمون دود انجام شود. پس از انجام بازآرایی کد (refactoring) نیز می‌بایست آزمون جامعیت به روش آزمون رگرسیون انجام شود. آزمون اعتبارسنجی، اعتبارسنجی عملکرد یا روند اجرای کل مولفه‌های نرم‌افزار ایجاد شده در کنار یکدیگر را در شرایط **سهل و آسان** مورد تست و ارزیابی قرار می‌دهد. هنگامی‌که یک **نرم‌افزار سفارشی** تنها برای یک مشتری توسعه می‌یابد، «**آزمون پذیرش**[1]» اجرا می‌شود تا مشتری را قادر به اعتبارسنجی کلیه‌ی خواسته‌ها کند. آزمون پذیرش، که به جای مهندس نرم‌افزار، توسط مشتری انجام می‌شود، می‌تواند از یک آزمون غیررسمی تا یک سری آزمون‌های برنامه‌ریزی شده‌ی سیستماتیک را شامل شود. درواقع، آزمون پذیرش را می‌توان در عرض یک هفته یا یک ماه انجام داد و لذا کشف خطاهایی که ممکن است سیستم را با گذشت زمان تنزل دهد، امکان‌پذیر می‌شود. در متدولوژی XP، آزمون اعتبارسنجی توسط آزمون پذیرش انجام می‌شود. مبنای آزمون پذیرش توسط مشتری، کنترل کارکرد همان سناریوهایی است که توسط مشتری در هر تکرار یا افزایش بیان شده‌است.

ارزش‌های متدولوژی XP

در متدولوژی XP پنج ارزش تعریف شده است که مبنایی برای انجام صحیح همه کارهای موجود در متدولوژی XP است. هرکدام از این ارزش‌ها به عنوان محرکه‌ای برای فعالیت‌ها، کنش‌ها و وظایف XP به کار می‌رود، این ارزش‌ها به صورت زیر است:

۱- **ارتباطات (communication)**: برقراری ارتباطات نزدیک به شکل رو در رو و غیررسمی مابین سازنده و مشتری جهت مشخص‌سازی خواسته‌های دقیق و واضح مشتری و پرهیز از برقراری ارتباط با مشتری برپایه تهیه مستندات پرحجم. هدف اصلی این ارتباطات

[1] Acceptance Test

شناخت نیازها و تعیین سناریوهای مشتری (customer stories) است.

۲- **سادگی (Simplicity)**: متدولوژی XP برای دستیابی به سادگی، به سازندگان توصیه می‌کند که فقط نیازهای فعلی بررسی و طراحی شوند و بررسی و طراحی نیازهای آینده به آینده واگذار شود و به حال آورده نشود. طراحی نیز باید ساده باشد و اگر نیاز به بهبود بود، بازآرایی می‌شود.

۳- **بازخورد (Feedback)**: در متدولوژی XP تیم توسعه باید توجه ویژه‌ای به بازخورد محصول، مشتری و حتی خود تیم توسعه داشته باشد. با ایجاد هر کلاس مربوط به یک use case آزمون واحد بر روی هر کلاس مطابق قابلیت‌های مورد انتظار انجام می‌شود و به تدریج کارکرد کلاس‌های همکار درون یک use case در کنار هم تحت بررسی آزمون جامعیت (یکپارچگی) قرار می‌گیرد. در نهایت نیز آزمون اعتبارسنجی توسط آزمون پذیرش بر روی نحوه پیاده‌سازی سناریوهای موجود در use caseهای تحویلی در هر افزایش انجام می‌شود. همچنین در هنگام برنامه‌ریزی افزایش بعدی، بازخورد برنامه‌ریزی افزایش قبلی مورد استفاده قرار می‌گیرد.

۴- **شجاعت (Courage) یا انظباط (discipline)**: پایبندی سفت و سخت به برخی جنبه‌های متدولوژی XP به جسارت و جرات نیاز دارد. یک واژه‌ی بهتر می‌تواند انظباط باشد. اغلب تمایل و فشار برای انجام طراحی نیازهای آینده وجود دارد. اکثر تیم‌های نرم‌افزاری هم سر تسلیم فرود می‌آورند، با این استدلال که طراحی برای آینده در بلندمدت منجر به صرفه‌جویی در زمان و تلاش می‌شود. اما تیم XP چابک باید انظباط و شجاعت لازم برای طراحی در امروز و زمان حال را داشته باشد. و در عین بداند که خواسته‌های آینده ممکن است به طرز چشمگیری تغییر کند که طراحی و پیاده‌سازی آنها در زمان حال منجر به دوباره‌کاری و اتلاف زمان و هزینه‌ها می‌گردد.

۵- **احترام (Respect)**: تیم چابک، با محقق‌کردن هریک از ارزش‌های فوق، احترام متقابل میان سازنده و مشتری را برقرار می‌کند. با تحویل موفق هریک از نسخه‌های نرم‌افزار این احترام و اعتقاد بر موثر بودن متدولوژی چابک XP بیشتر و بیشتر نیز می‌شود.

متدولوژی Scrum

نام متدولوژی Scrum از بازی راگبی گرفته‌شده است. راگبی ورزش گروهی است که در یک زمین چمن مستطیل شکل با دروازه‌های H شکل در دو سوی زمین و با یک توپ بیضی شکل بازی می‌شود. اسکرام در واقع یک شروع دوباره در بازی راگبی است به طوری که وقتی خطایی در این بازی روی می‌دهد یا اینکه توپ از زمین خارج می‌گردد از روش اسکرام برای آغاز دوباره بازی استفاده می‌شود. به این شکل که همه بازیکن‌ها دور هم جمع می‌شوند، سرهایشان را پایین می‌گیرند و شروع مجدد می‌کنند، اسکرام در بازی راگبی بسیار اتفاق می‌افتد.

اسکرام یک روش گروهی برای تولید و توسعه نرم‌افزار است. اینکه در هریک از فعالیت‌های چارچوبی شامل ارتباط، برنامه‌ریزی، تحلیل، طراحی، پیاده‌سازی، تست و استقرار چه وظایفی باید انجام شود، توسط sprintها مشخص می‌شود که در ادامه به آن می‌پردازیم.

نقش‌های اسکرام (scrum roles)

- **اسکرام مستر (scrum master) :** رهبر اسکرام وظیفه دارد تا تمامی اعضای تیم را هدایت و راهنمایی نماید تا هیچ یک از اعضای تیم از چارچوب و قوانین اسکرام خارج نشوند. رهبر اسکرام نقش مدیر را ندارد بلکه تنها وظیفه رهبری تیم را بر عهده دارد تا با رفع مشکلات و موانع پیش رو، در صورتی که اعضای تیم قادر به رفع موانع نباشند، اجرای اسکرام را بهبود بخشد.
- صاحب محصول (product owner) : صاحب محصول که نماینده ذینفعان (Stakeholders) پروژه و business است، با اعلام دقیق نیازمندی‌های خود به تیم تولید، با رهبر اسکرام و تیم تولید همکاری می‌نماید. صاحب محصول باید به سوالات تیم تولید پاسخ داده و همواره در دسترس باشد.
- **تیم تولید و توسعه نرم‌افزار (development team) :** افراد این تیم در چارچوب قوانین اسکرام، به تولید آن چه که صاحب محصول درخواست کرده است، می پردازند. تعداد اعضای تیم تولید نه باید آن قدر کم باشد که همکاری گروهی و کار تیمی بی‌معنا شود و نه آن قدر زیاد باشد که هماهنگی بین اعضای تیم تبدیل به امری دشوار و وقت‌گیر گردد. تعداد اعضای تیم تولید، بستگی به پروژه دارد اما معمولا ۶ تا ۹ نفر اعضای این تیم را تشکیل می دهند.

روند کارکرد اسکرام

هسته اصلی اسکرام را sprintها تشکیل می‌دهند. در متدولوژی‌های تکرارشونده (iterative) دوره‌های زمانی تکراری (iteration) وجود دارد که در این دوره‌ها به تدریج محصول کامل می گردد. بدین صورت که در تولید یک محصول، تعدادی تکرار در نظر گرفته می‌شود که در پایان دوره‌ی زمانی هر تکرار، یک محصول قابل ارائه وجود دارد. به این دوره‌های زمانی تکرارشونده در اسکرام sprint می‌گویند. در پایان هر sprint، محصول کامل‌تر شده و در نهایت محصول نهایی تولید می‌گردد. هر sprint دارای تعریفی است که در آن باید مشخص شده باشد که چه چیزی قرار است ساخته شود، نیازمندی‌ها، راهنمای ساخت و محصول خروجی نیز باید مشخص باشند. نتیجه اینکه مثل تمام متدولوژی‌های incremental و iterative در اسکرام نیز دوره‌های زمانی یا iteration داریم که در طی آنها محصول نهایی پروژه به‌تدریج تکمیل می‌شود.

در طی یک sprint که معمولا یک دوره دو تا چهار هفته است یعنی حداکثر ۳۰ روز (طول

دوره را تیم مشخص می‌کند) اعضاء، یک محصول قابل ارائه و قابل استفاده را تدریجا تولید می‌کنند.

اصطلاح Product Backlog نامی است که به لیست نیازمندهای وظیفه‌مندی و غیروظیفه‌مندی کل یک پروژه اطلاق می‌شود و در حقیقت مجموعه‌ای اولویت‌بندی شده از نیازمندی‌های مشتری است که در نهایت بایستی تحویل داده شود.

مواردی از Product Backlog که در طی یک sprint بایستی انجام شود، در طول جلسه‌ی طراحی sprint یا Sprint Planning Meeting مشخص می‌شود. در طول این جلسه، صاحب محصول (Product Owner)، تیم تولید و توسعه نرم‌افزار (development team) را درباره‌ی مواردی از Product Backlog آگاه می‌کند. سپس اعضاء تیم تولید مشخص می‌کنند که چه مقدار از موارد مشخص شده توسط Product Owner را می‌توانند در این sprint انجام دهند و چه میزان از آنرا در sprint های بعدی.

مواردی از Product Backlog که قرار است در یک Sprint انجام شود را اصطلاحا Sprint Backlog می‌نامند. مفاد Sprint Backlog در واقع توافقی است بین اعضاء تیم تولید و Product Owner، که بعد از تصویب‌شدن مفاد یک sprint، دیگر هیچ‌کس نمی‌تواند آنرا در طول sprint تغییر دهد. در طول دوره، نیازمندی‌های لحاظ شده در Sprint Backlog از Product Backlog حذف می‌شوند. اما امکان دارد به دلایلی که در ادامه می‌آید این نیازمندی‌ها دوباره به Product Backlog برگردد.

در ساده‌ترین روش معمولا از نرم‌افزارهای صفحه گسترده (Spread Sheet) همچون Microsoft Excel برای ساختن و نگهداری Product Backlog و Sprint Backlog استفاده می‌شود، اما می‌توان از طیف وسیعی از ابزارهای نرم‌افزاری که برای استفاده در تیم‌های Agile نوشته شده‌اند نیز استفاده کرد.

مانند تمام متدولوژی‌های iterative، توسعه‌ی نرم‌افزار در اسکرام نیز Time Boxed است، به این معنی که sprint بایستی دقیقا سر وقت تمام شود و اگر نیازمندی‌های تعیین شده در Sprint Backlog به هر علتی تکمیل نشده باشند آنها را کنار گذاشته و دوباره وارد Product Backlog می‌کنند.

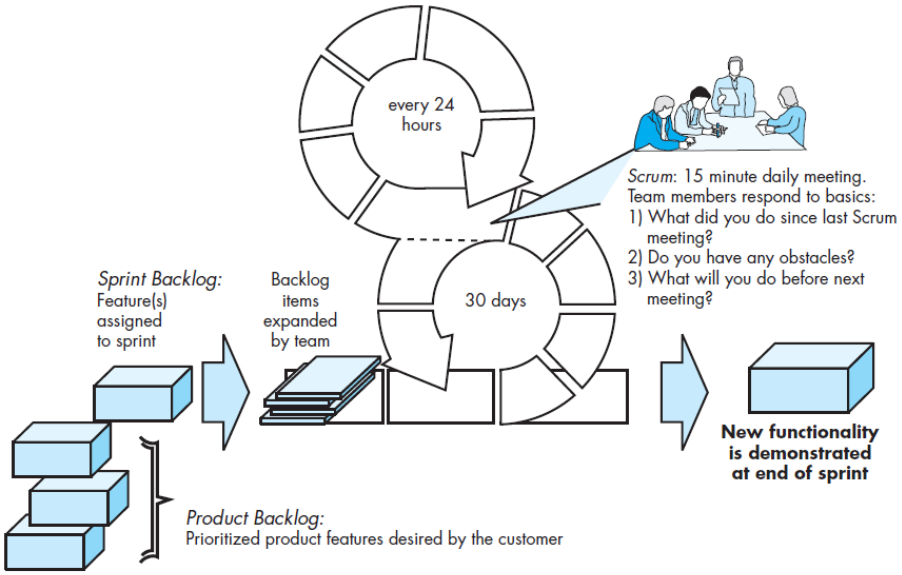
در طول اجرای هر sprint جلساتی روزانه (هر ۲۴ ساعت) و کوتاه حدود ۱۵ دقیقه با هدف کشف زودهنگام خطاها و بررسی پیشرفت پروژه توسط تیم اسکرام و با حضور اعضای تیم (تیم تولید و ذینفعان) برگزار می‌شود که به آن جلسات اسکرام (scrum meeting) گفته می‌شود.

در این جلسات باید به سه پرسش زیر پاسخ داده شود:

- از جلسه قبل تا این جلسه چه کارهایی انجام شده است؟
- از جلسه قبل تا این جلسه با چه موانعی مواجه شده‌اید؟

- از این جلسه تا جلسه بعدی چه کارهایی قرار است انجام شود؟

اگرچه جلسات اسکرام به نسبت کوتاه است، اما تاثیر جلسات بر روند بهبود فرآیند قابل چشم‌پوشی نیست. کشف زودهنگام خطاهای بالقوه از مهم‌ترین مزایای این جلسات است. این موضوع منجر به این می‌شود که تیم پروژه زمانی خطا را مرتفع کند که هزینه و زمان مورد نیاز برای رفع آن هنوز مقداری قابل قبول است. مزیت دیگر این جلسات آن است که دانش عمومی تیم، نسبت به کار یکدیگر و کل کاری که در sprint جاری انجام می‌شود، بالا رود. این موضوع باعث استحکام کار تیمی خصوصا در تیم‌های خودسازمانده (self-organizing) می‌شود. جلسات اسکرام توسط رهبر اسکرام (scrum master) هدایت می‌شود. رهبر اسکرام ضمن هدایت جلسات، مسئولیت ارزیابی پاسخ‌های اعضا در این جلسات را نیز برعهده دارد. شکل زیر گویای روند کارکرد متدولوژی اسکرام است:



بعد از خاتمه یک sprint، اعضاء تیم طی جلسه‌ای به Product Owner و سایر ذینفعان پروژه نشان می‌دهند که چکار کرده‌اند و چطور از نسخه‌ی جاری نرم‌افزار می‌شود استفاده کرد. در واقع پس از اتمام هر sprint کارکردهای جدید که در طول این sprint ساخته‌شده‌اند، در قالب یک افزایش جدید از نرم‌افزار به صاحب محصول و ذینفعان ارائه می‌شود. همانطور که پیش‌تر نیز گفتیم، کارکردهای جدید هر افزایش، دقیقا برابر تمام کارکردهایی که در ابتدای sprint برنامه‌ریزی شده بود، نیست. در واقع گاهی محدودیت زمانی sprint باعث می‌شود در مواقعی که sprint از زمان‌بندی عقب است، برخی از کارکردها در آن افزایش پیاده‌سازی نشوند. در این شرایط پیاده‌سازی این کارکردها به افزایش‌های بعدی سپرده می‌شود. به مرحله‌ی ارائه‌ی کارکردهای جدید نرم‌افزار در انتهای هر sprint یک demo گفته می‌شود. پس از تحویل هر افزایش یا sprint، افزایش

یا sprint بعـدی برنامـه‌ریزی می‌شـود. همچنـین در هنگـام برنامـه‌ریزی sprint بعـدی، بـازخورد برنامه‌ریزی sprint قبلی مورد استفاده قرار می‌گیرد.

نتیجه اینکه از مجموع جلسات اسکرام یک Sprint بوجود می‌آید و از مجمـوع ایـن sprintهـا هم کل فرآیند توسعه‌ی نرم‌افزار ایجاد می‌گردد. متدولوژی اسکرام برای انجام پروژه‌های پراهمیـت که تحت فشار زمان‌بندی بوده و همواره درحال تغییر هستند، مناسب و کارآمد است.

تست‌های فصل یازدهم

۱- کدام مورد به عنوان یک متدولوژی شناخته می‌شود؟ **(مهندسی IT – دولتی ۸۸)**

۱) ORACLE ۲) 4GT ۳) RAD ۴) X-Programming

۲- کدام‌یک از گزینه‌های زیر از اصول رسیدن به چابکی نیست؟ **(مهندسی IT – دولتی ۹۳)**

۱) شرط چابکی ادغام طراحی و ساخت است.

۲) تولیدکنندگان و مشتریان باید روزانه و پیوسته با یکدیگر همکاری کنند.

۳) بهترین طراحی‌ها، معماری‌ها و نیازها از تیم‌های خود سازمانده منتج می‌شود.

۴) رضایت مشتری از طریق تحویل نسخ محصول بطور پیوسته و سریع بالاترین اولویت را دارد.

۳- کدام عبارت در مورد بازآرایی کد (Refactoring) در XP، هیچگاه درست نیست؟

(مهندسی IT – دولتی ۹۶)

۱) بازآرایی، کد را خواناتر می‌کند.

۲) بازآرایی، کد را تغییرپذیرتر می‌کند.

۳) بازآرایی رفتار بیرونی کد را اصلاح می‌کند.

۴) بازآرایی ساختار داخلی کد را بهبود می‌دهد.

پاسخ تست‌های فصل یازدهم

۱- گزینه (۴) صحیح است.

ORACLE یک DBMS است، 4GL زبان‌های نسل چهارم را گویند و RAD یک مدل توسعه سریع می‌باشد. X-Programming یک متدولوژی چابک است.

۲- گزینه (۱) صحیح است.

طراحی در متدولوژی XP بر پایه‌ی سادگی بنا شده است، این اصل که سادگی را حفظ‌کن (keep it simple). در متدولوژی XP همواره یک طراحی ساده بر یک طراحی پیچیده برتری دارد. اما این سادگی در طراحی به این معنی نیست که شرط چابکی حذف طراحی و یا ادغام طراحی و ساخت است.

۳- گزینه (۳) صحیح است.

طراحی در متدولوژی XP برپایه‌ی سادگی بنا شده است، بنابراین این دغدغه وجود دارد که گاهی طراحی‌های XP، از کیفیت الگوریتمی (غیروظیفه‌مندی یا مزه) مناسبی برخوردار نباشد. متدولوژی XP برای مرتفع نمودن این دغدغه، روش فاکتورگیری مجدد یا بازآرایی (refactoring) را پیشنهاد می‌کند. بنابراین با وجود اینکه روش فاکتورگیری مجدد در مرحله کدنویسی انجام می‌شود اما ماموریت آن ارتقاء کیفیت الگوریتمی و خوشمزه‌سازی طراحی است. فاکتورگیری مجدد یا بازآرایی، فرآیند تغییر، بهبود و ارتقاء کیفیت کد است، بازآرایی، کد را خواناتر می‌کند، بازآرایی، کد را تغییرپذیرتر می‌کند، بازآرایی، ساختار داخلی کد را بهبود می‌دهد اما بدون آنکه بازآرایی، رفتار بیرونی کد را اصلاح کند، به عبارت دیگر بازآرایی، رفتار بیرونی کد را اصلاح نمی‌کند یعنی ورودی‌ها و خروجی‌های کد را تغییر نمی‌دهد.
