به نام خداوند بخشنده‌ی مهربان

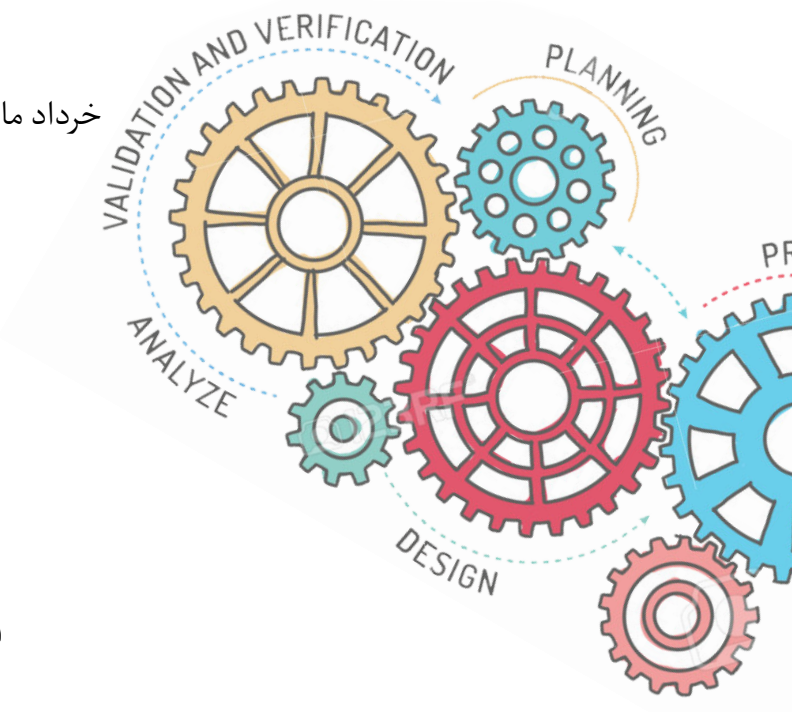
مهندسی نرم‌افزار

تمرین سری چهار

استاد درس: دکتر حبیبی

محمّدمهدی فاریابی
۹۳۱۰۱۹۵۱

خرداد ماه ۱۳۹۷



۱ پرسش اول

با مطالعه در مورد متودولوژی Twelve-factor app به پرسش‌های مطرح شده پاسخ می‌گوییم:

۱.۱ بررسی انطباق هر یک از فاکتورهای دوازده گانه با فاکتورهای کیفی McCall

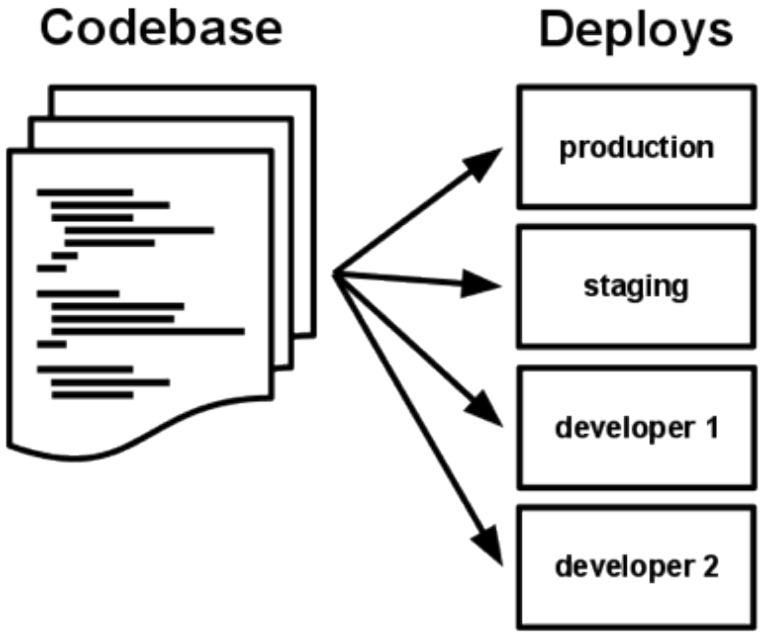
۱.۱.۱ Codebase

یک برنامه‌ی twelve-factor باید دارای یک codebase واحد باشد اما می‌توان گسترش‌های متفاوتی را از همین codebase واحد برای آن متصور بود. به عنوان مثال یک گسترش برای آزمون نرم‌افزار در محیط سرور آزمون، یک گسترش به ازای هر توسعه‌دهنده در محیط داخلی سیستم توسعه‌ی او و یک گسترش نهایی محصول.

این فاکتور بر ویژگی‌های ناظر بر نگهداری و تغییر نرم‌افزار در مجموعه‌ی McCall نظارت دارد. به این صورت که داشتن گسترش‌های متفاوت برای یک codebase می‌تواند باعث سادگی ایجاد تغییرات در نرم‌افزاری شود که هم اکنون در حال خدمت رسانی و سرویس دهی است .(flexibility) به این صورت تغییرات اعمال شده پس از طی چندین گسترش مختلف تست می‌شوند و در نهایت به گسترش نهایی محصول راه می‌یابند.

این فاکتور ناظر بر ویژگی testablility نیز هست. به این صورت که ویژگی جدید یا تغییرات اعمال شده را می‌توان بدون واهمه از مشکل در گسترش محصول در گسترش‌های آزمون به راحتی تست کرد.

همچنین maintainability یک محصول بالاتر خواهد رفت. زیرا کنکاش و اعمال تغییرات برای یافتن مشکلات در codebase در ابتدا در گسترش نهایی انجام نمی‌شود و ریسک ایجاد مشکل برای محصول حین خدمت رسانی وجود ندارد



تصویر ۱: چندین گسترش با یک codebase [۱]

۲.۱.۱ Dependencies

تمام وابستگی‌های یک برنامه باید صراحتا مشخص و ایزوله شوند. یک برنامه‌ی twelve-factor نباید به هیچ عنوان به کتابخانه‌ها و ابزارهای خارجی موجود در محیط اجرایی میزبان وابسته یا امیدوار باشد. بلکه باید نیازمندی‌ها و وابستگی‌های خود به این ابزارها (مانند کتابخانه‌های زبان که معمولا با ابزارهای package management نصب می‌شوند یا برنامه‌های کاربردی (shell را صراحتا اعلام کند و در محیطی ایزوله از محیط بیرونی خود در سیستم این نیازمندی‌ها را برای خود فراهم کند (مثل استفاده از virtualenv در زبان پایتون).

این فاکتور ناظر به برخی از ویژگی‌های کیفی .استMcCall

با بهره‌گیری از این فاکتور برنامه usability بیشتری خواهد داشت. توسعه‌دهندگان جدید و کاربران وقت بسیار کمتری را صرف آماده کردن محیط خارجی کد در سیستمشان و نصب نسخه‌ی مناسب کتابخانه‌ها و ابزارهای مورد نیاز کد و رفع تداخلات خواهند کرد.

[۱] منبع تصویر: `https://12factor.net/codebase`

این فاکتور باعث portability بیشتر کد و برنامه برای اجرا در محیط‌های متفاوت خواهد شد. به این صورت که تفاوت‌های معماری و ساختاری محیط‌های متفاوت تاثیر کمتری بر کد خواهد داشت. زیرا کد برای رفع نیازمندی‌های خود به آنچه در سیستم هدف وجود دارد وابسته نیست.

این فاکتور حتی باعث افزایش interoperabality هم خواهد شد. زیرا چندین برنامه می‌توانند در یک سیستم واحد در کنار هم کار کنند و هیچیک از یک مجموعه‌ی مشترک برای رفع نیازمندی‌های خود استفاده نکنند. به این صورت جلوی تداخلات احتمالی نیازمندی‌هایشان و کتابخانه‌ها گرفته خواهد شد.

Config ۳.۱.۱

این فاکتور بیان می‌کند که تنظیمات و config مربوط به گسترش بایستی در متغیرهای محیطی environment varaibles ذخیره شود. این تنظیمات شامل دسترسی‌ها به سرویس‌هایی مانند پایگاه داده و صف‌ها، اطلاعات محرمانه مانند token های امنیتی دسترسی به سرویس‌های خارجی و تنظیمات مربوط به گسترش خاص هستند.

این اطلاعات نباید به صورت فایل config در repository ذخیره شوند زیرا codebase نباید شامل اطلاعاتی باشد که در گسترش‌های متفاوت فرق خواهد کرد. همچنین نباید این اطلاعات درون فایل‌هایی با اسامی مختلف ذخیره شود و در زمان گسترش بین انتخاب از آنها تصمیم گرفته شود زیرا با بزرگ شدن پروژه مدیریت این فایل‌ها نیز دشوار خواهد شد.

این فاکتور ناظر بر چندین ویژگی McCall است.

استفاده از آن باعث افزایش integrity خواهد شد. به این ترتیب می‌توان با تنظیم متغیرهای محیطی مختلف در محیط‌های گسترش مختلف دسترسی‌های مختلفی را تنظیم کرد.

باعث افزایش portability خواهد شد زیرا codebase حاوی اطلاعات گسترش نیست و می‌تواند به سادگی و بدون تغییرات در گسترشی دیگر مورد استفاده قرار بگیرد.

راه را برای ارائه‌ی flexibility بیشتر هموار می‌کند. با استفاده‌ی درست از متغیرهای محیطی می‌توان به سادگی و سرعت رفتار یک سیستم در حال خدمت را تغییر داد.

Backing Services ۴.۱.۱

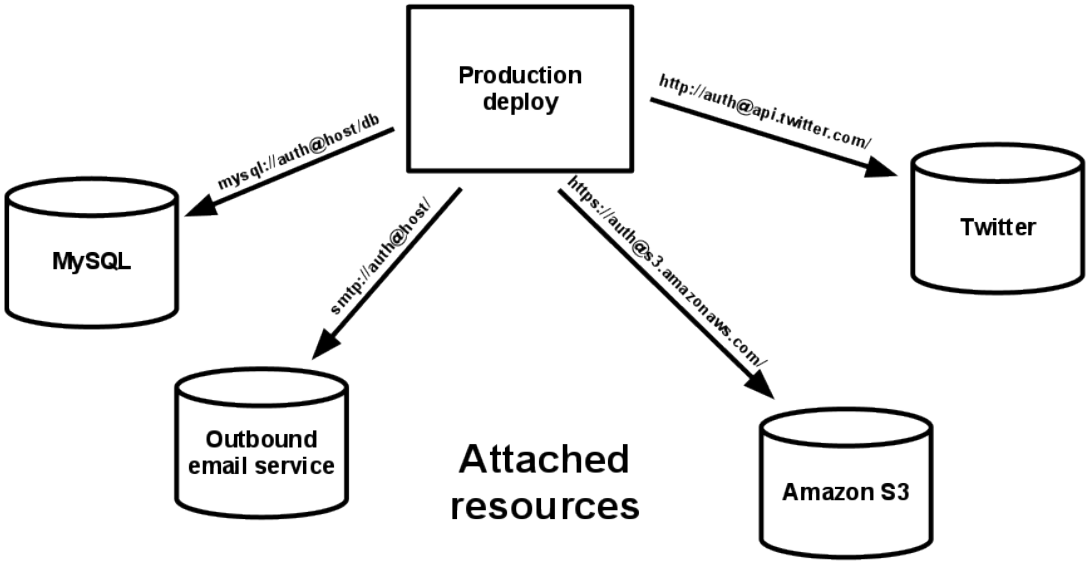
این فاکتور بیان می‌کند یک نرم‌افزار twelve-factor بایستی با سرویس‌های backing مانند پایگاه‌های داده، ابزارهای مدیریت صف، ابزارهای مدیریت حافظه، ابزارهای تبادل و مدیریت ایمیل و سایر ابزارهای کمکی داخلی و یا خارجی مانند منابع متصل به سیستم برخورد کند. به این صورت که شناسه و handle این ابزارها در یک پیکربندی ذخیره شده و برنامه از طریق این پیکربندی بتواند به منبع مورد نظر دسترسی داشته باشد. این اطلاعات و تنظیمات نباید در کد برنامه ذخیره شوند.

نباید از دید برنامه تفاوتی میان یک پایگاه داده‌ی داخلی و یک پایگاه داده‌ی راه دور تهیه شده از شرکتی مثل amazon وجود داشته باشد. بایستی در هرلحظه امکان تغییر استفاده از یک منبع به منبع دیگر با تغییر دادن handle آن منبع در پیکربندی موجود باشد.

به این صورت coupling بین مولفه‌های نرم‌افزار به شدت پایین می‌آید.

استفاده از این فاکتور ناظر بر چندین ویژگی McCall است.

با coupling بسیار کم بین مولفه‌های یک سیستم امکان تغییر در یک سیستم در حال اجرا (flexibility) مهیا می‌شود. امکان نگهداری سیستم بالاتر می‌رود زیرا می‌توان به سادگی مولفه‌های دارای ایراد را با مولفه‌های سالم جایگزین کرد .(maintainability) می‌توان به سادگی مولفه‌های آزمون را بری آزمون سیستم به آن متصل و از آن جدا کرد .(testability) می‌توان به سادگی از یک مولفه در سیستمی دیگر استفاده کرد(reusablility). می‌توان چندین مولفه را به سادگی و در تعامل با یکدیگر در یک سیستم مورد استفاده قرار داد .(interoperability) می‌توان بدون تغییر از codebase در محیط‌های متفاوت و با منابع متصل متفاوت استفاده کرد .(portability)



تصویر ۲: تعامل با سرویس‌های backing مانند منابع متصل به سیستم [۲]

۵.۱.۱ Build, release, run

این فاکتور بیان می‌کند که در یک نرم‌افزار twelve-factor گام‌های build و release و run بایستی جدا از هم دیده شوند.

در گام build ، codebase برنامه در commit مورد نظر توسعه‌دهنده از repository دریافت شده، و با ادغام با کتابخانه‌های مورد استفاده build شده و تبدیل به فایل‌های binary قابل اجرا می‌شود.

در گام release تنظیمات اجرایی محیط به نتیجه‌ی گام قبل اضافه شده و بسته‌ای اجرایی تولید می‌شود که شامل فایل‌های باینری اجرایی و تنظیمات گسترش است.

در نهایت گام run گامی است که بسته‌ی اجرایی نهایی را در محیط گسترش اجرا می‌کند.

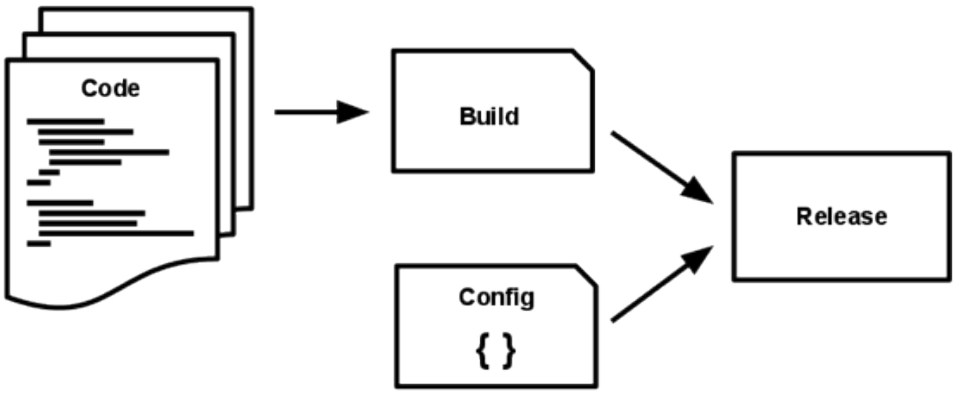
گام run بایستی حتی‌الامکان گامی کوچک و متشکل از چند قدم ساده باشد. زیرا این گام ممکن است هر لحظه به صورت خودکار رخ دهد (راه اندازی مجدد سرور گسترش پس از یک خطا در نیمه‌ی شب را در نظر بگیرید). از این رو این گام نباید پیچیده و خطاخیز باشد چون ممکن است توسعه‌دهنده‌ای برای رسیدگی به این خطا در دسترس نباشد.

هر release هم بایستی برچسبی منحصر به فرد متناسب با آن release دریافت کند که در صورت لزوم بتوانیم به سادگی بین release های مختلف جابه‌جا شویم.

استفاده از این فاکتور ناظر بر چندین ویژگی McCall است.

استفاده از آن باعث افزایش maintainability خواهد شد زیرا در صورت بروز خطا می‌توان به سادگی به یک حالت امن قبل بازگشت.

باعث افزایش reliability خواهد شد زیرا در صورت بروز خطا سیستم می‌توان به recovery خودکار به حالتی امن بپردازد و down-time ای را تجربه نکند.



تصویر ۳: جدا نگه‌داشتن فازهای build و release و run [۳]

[۲] منبع تصویر: https://12factor.net/backing-services
[۳] منبع تصویر: https://12factor.net/build-release-run

Processes ٦.١.١

این فاکتور بیان می‌کند که یک برنامه‌ی twelve-factor بایستی در قالب یک یا چند پردازه‌ی بدون حالت (stateless) اجرا شود. حافظه‌ی پردازه یا فایل‌سیستم به هیچ عنوان نباید به عنوان محل ذخیره‌سازی دائمی اطلاعات مورد استفاده قرار گیرند.

در محیطی که تعداد زیادی پردازه‌ی مشابه به سرویس‌دهی مشغول هستند ذخیره‌ی اطلاعات در حافظه‌ی یک پردازه کار درستی نیست چون درخواست‌های بعدی به آن اطلاع ممکن است به پردازه‌ی دیگری ارسال شوند.

همچنین در محیط‌های تک پردازه‌ای هم ممکن است طی اتفاقاتی اطلاعات حافظه‌ی پردازه پاک شود یا در دسترس نباشد.

هر آنچه که باید به صورت پایدار ذخیره شود بایستی در یک پایگاه داده قرار بگیرد.

از دیدگاه ویژگی‌های استفادهMcCall از این فاکتور باعث افزایش reliability می‌شود زیرا احتمال در دسترس نبودن اطلاعات در یک دسترسی در محیط‌های parallel یا از دست رفتن آنها به دلیل اتفاقی غیر منتظره بسیار کم می‌شود.

Port binding ٧.١.١

این فاکتور بیان می‌کند که یک برنامه‌ی twelve-factor به صورتی کاملا مستقل فعالیت می‌کند و نیازی به ورود یک webserver در زمان اجرا برای ارائه‌ی خدمت خود ندارد.

به این صورت که چنین برنامه‌ای با reveal کردن یکی از port های سیستم یا container میزبان به بیرون و گوش دادن روی آن پورت برای درخواست‌های ورودی به ارائه‌ی خدمت می‌پردازد.

به این صورت یک برنامه می‌تواند به عنوان یک backing service برای یک برنامه‌ی دیگر مورد استفاده قرار بگیرد و برنامه‌ی استفاده کننده با در اختیار داشتن URI آن برنامه از آن سرویس بگیرد.

از دیدگاه ویژگی‌های استفادهMcCall از این فاکتور باعث افزایش interoperability می‌شود زیرا چند سیستم می‌توانند به عنوان سرویس دهنده و سرویس گیرنده در کنار یکدیگر به غعالیت بپردازند.

باعث افزایش reusability خواهد شد زیرا می‌توان از این برنامه‌ی self-contained در جاهای دیگر استفاده کرد.

باعث افزایش portability می‌شود زیرا یک برنامه‌ی self-contained وابستگی کمی به محیط اجرا دارد و می‌توان آن را به سادگی به محیط‌های دیگری منتقل کرد.

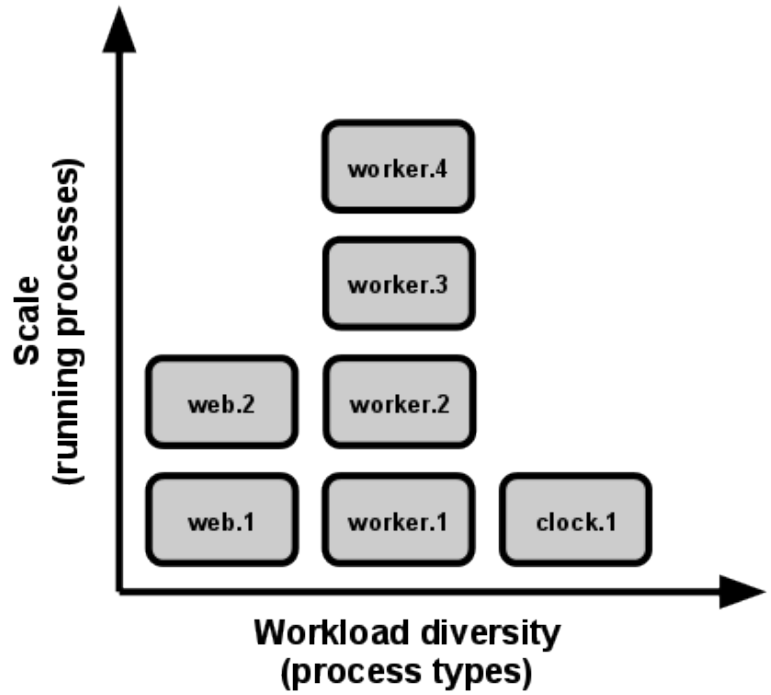
Concurrency ٨.١.١

در یک برنامه‌ی twelve-factor از انواع مختلف پردازه‌ها برای رسیدگی به کارهای مختلف استفاده می‌شود. بسته به میزان workload هر کار ممکن است تعداد کم یا زیادی از پردازه‌ی رسیدگی‌کننده به آن کار تولید شود.

مدل پردازه‌ای برای افزایش ابعاد محصول بسیار مفید و پرکاربرد است. کافی است تعداد پردازه‌های رسیدگی کننده به کار متناسب با افزایش حجم آن کار افزایش یابد.

در یک برنامه‌ی twelve-factor هیچگاه از پردازه‌های daemon استفاده نمی‌شود و مدیریت پردازه‌ها هم به صورت خودکار توسط سیستم عامل صورت می‌پذیرد.

از دیدگاه ویژگی‌های استفادهMcCall از این فاکتور باعث افزایش efficiency و reliablility می‌شود. زیرا با شکستن کار و موازی کردن آن نیاز به منابع مورد نیاز (من جمله زمان) نسبت به حالت تک عاملی کمتر می‌شود و همینطور با افزایش فشار کاری بر سیستم احتمال شکست سیستم و خطا در آن به صورت چشمگیر افزایش نمی‌یابد.



تصویر ۴: بالابردن ابعاد با استفاده از مدل پردازه‌ای [۴]

۹.۱.۱ Disposability

در یک برنامه‌ی twelve-factor پردازه‌ها بایستی به سرعت از لحظه‌ی دستور شروع به کار به حال آماده به کار در آیند. در چنین سناریویی مدیر پردازه می‌تواند بسته به میزان فشار کاری، به سرعت نسبت به افزایش منابع اقدام کند (کمینه کردن زمان startup هر پردازه).

همچنین باید پردازه‌ها به صورت graceful از بین بروند به این صورت که پردازه با دریافت سیگنال SIGTERM ابتدا وظیفه‌ی محول شده را انجام داده یا عدم امکان انجام را اطلاع رسانی کند و سپس از بین برود.

به بیانی دیگر پردازه‌ها باید در مقابل مرگ ناگهانی مقاوم باشند.

از دیدگاه ویژگی‌های استفادهMcCall از این فاکتور باعث افزایش reliability و eficiency می‌شود زیرا افزایش سرعت startup پردازه‌ها معادل افزایش سرعت scaleup در فشارهای کاری بالا و افزایش efficiency است. همچنین robust بودن در مقابل مرگ ناگهانی و خطا معادل reliability برنامه در ارائه‌ی خدمات است.

۱۰.۱.۱ Dev/prod parity

این فاکتور بیان می‌کند که در فرایند توسعه‌ی یک نرم‌افزار twelve-factor ، محیط‌های development ، staging و production باید تا جای ممکن شبیه به هم نگهداری شوند.

این شباهت باید در نزدیکی زمانی و محتوایی کد، نزدیکی پرسنل توسعه دهنده و deploy کننده‌ی کد و همچنین نزدیکی ابزارهای استفاده شده در فرایند development و production باشد.

با استفاده از فرایند continuous delivery سعی در کاهش فاصله‌ی زمانی کد و پرسنل توسعه و deploy می‌شود.

یک توسعه‌دهنده‌ی twelve-factor سعی می‌کند technology-stack مورد استفاده در فرایند توسعه دقیقا همان technology-stack مورد استفاده در production باشد.

از دیدگاه ویژگی‌های استفادهMcCall از این فاکتور باعث افزایش testability ، reliability و correctness می‌شود.

شباهت کد توسعه و کد محصول نهایی و همچنین ابزارها راه تست کردن و پیدا کردن ایرادات را هموار تر می‌کند.

تست راحت‌تر و عدم وجود ایرادات ناشی از تفاوت‌های technology-stack باعث افزایش correctness و همچنین reliability محصول خواهد بود.

۱۱.۱.۱ Logs

این فاکتور بیان می‌کند که یک برنامه‌ی twelve-factor نباید خود را درگیر نوشتن لاگ‌ها در فایل‌های مربوط و مدیریت آنها بکند.

[۴] منبع تصویر: https://12factor.net/concurrency

هر بخش از برنامه بایستی لاگ خود را در stdout بنویسد. توسعه دهنده در زمان توسعه‌ی این بخش لاگ را مشاهده کرده و بر اساس آن به ایراد زدایی و توسعه می‌پردازد. در production لاگ هر ماژول توسط محیط اجرایی آن capture شده و در قالب مشخص و در محل تعیین شده برای بررسی و نگهداری طولانی مدت ذخیره می‌کند.

این لاگ‌ها را می‌توان بعدها توسط ابزارهای مدیریت لاگ برای تحلیل‌های آماری و بررسی‌های سطح بالا مورد استفاده قرار داد.

از دیدگاه ویژگی‌های استفادهMcCall از این فاکتور باعث افزایش flexibility خواهد شد زیرا توان مورد نیاز برای اصلاح یک ایراد در برنامه‌ی در حال اجرا با در نظر گرفتن و مشاهده‌ی لاگ‌های آن کاهش می‌یابد.

همچنین باعث افزایش testability و maintainability خواهد شد زیرا آزمون یک برنامه‌ی درحال اجرا و نگهداری از آن و برطرف کردن ایرادات آن با در اختیار داشتن خروجی‌ها و بازخوردهای آن به تعاملات بسیار ساده تر خواهد بود.

۱۲.۱.۱ Admin processes

این فاکتور بیان می‌کند که در یک برنامه‌ی twelve-factor فعالیت‌ها و پردازه‌های کوتاه مدیریتی بایستی مانند پردازه‌های طولانی و اصلی خدمت و در همان محیط و با همان پیکربندی اجرا شوند.

این پردازه‌ها و وظایف مانند دستورات کوتاه مدیریتی و نظارت و نگهداری و پشتیبان گیری غالبا به صورت فایل‌های script هستند.

فعالیت‌های مشابه dependency-isolation ای که برای کد اصلی خدمت انجام می‌شود باید برای این کدها نیز انجام شود.

از دیدگاه ویژگی‌های استفادهMcCall از این فاکتور باعث افزایش portability و maintainability می‌شود زیرا ایزوله بودن این کدها از وابستگی‌های خارجی باعث کاهش وابستگی برنامه به محیط اجرا می‌شود و همچنین یکسانی محیط اجرای این دستورات مدیریتی و دستورات طولانی خدمت باعث عدم نیاز به در نظر گرفتن تفاوت محیط‌ها در نگهداری و سادگی فرایند آن خواهد بود.

۲.۱ تضمین کیفی توسط متودولوژی؟

این متودولوژی تعداد خوبی از مشکلات با نرخ رخداد بالا در توسعه‌ی نرم‌افزارهای SaaS را آدرس‌دهی کرده و برای حل آنها راه‌حل‌هایی را پیشنهاد می‌کند. استفاده از این متودولوژی قطعا در کیفیت نهایی نرم‌افزار تولید شده تاثیر خواهد داشت. اما می‌دانیم کیفیت یک معیار نسبی است و عوامل بسیار زیادی در آن موثرند. این متودولوژی به بسیاری از این عوامل می‌پردازد و سعی در بهبود آنها دارد ولی در نهایت تضمینی برای کیفیت بی نقص وجود ندارد.

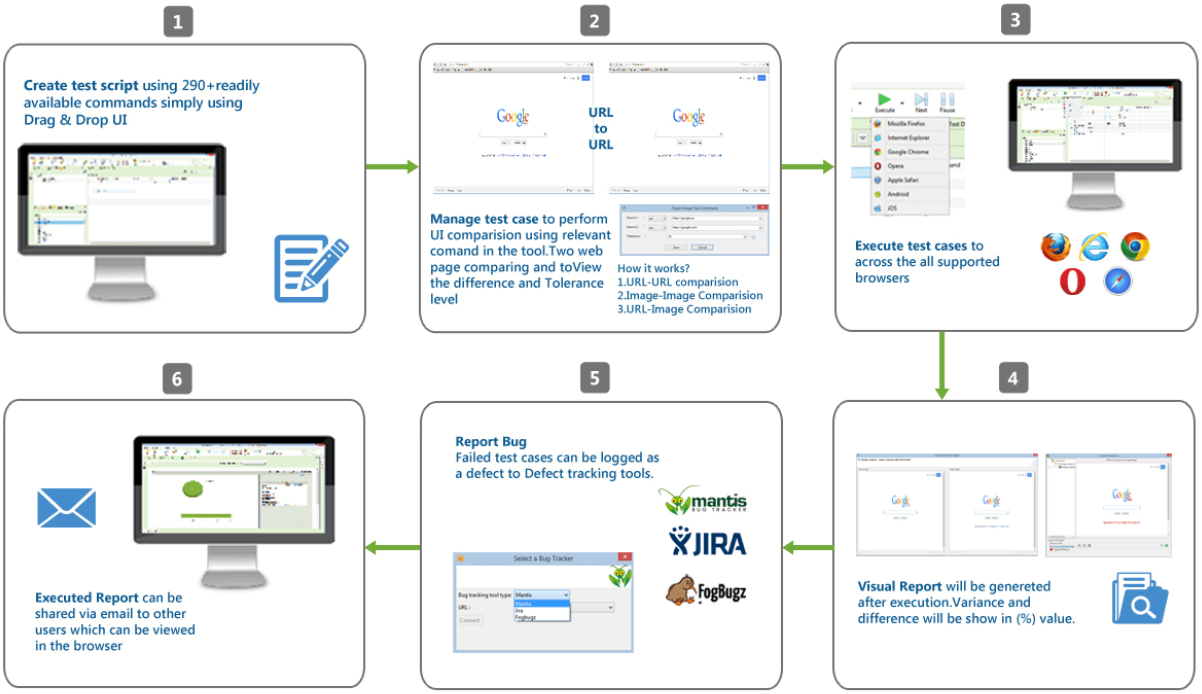
۲ پرسش دوم

هر یک از جفت‌های تست و ابزار پیشنهادی را بررسی کرده و در مورد امکانات و نحوه‌ی کارکرد و نحوه‌ی تست و فراهم کردن نیازهای موضوع توسط ابزار توضیح می‌دهیم:

۱.۲ تست رابط کاربری وب، TestingWhiz

ابزار آزمون خودکار رابط کاربری وب TestingWhiz امکان صحت سنجی رابط کاربری و front-end نرم‌افزارهای تحت وب ، بررسی درستی عملکرد آن در نمایش UI و ارائه‌ی UX مناسب در واسط‌های کاربری متنوع و browser های متفاوت را به ما می‌دهد.
با استفاده از این ابزار می‌توان دو صفحه‌ی وب را capture کرده و در ادامه صحفه‌ی مورد تست را با صفحه‌ی هدف مقایسه کرد. این قیاس به صورت پیکسل به پیکسل و با دقت و حساسیت قابل تنظیم انجام می‌شود.
مقایسه‌ی انجام شده می‌تواند بین دو صفحه، یک صفحه و یک تصویر یا دو تصویر انجام شود.
به وسیله‌ی این ابزار می‌توان قابلیت جابه‌جایی بین صفحات [۵] ، چینش صفحات [۶] و قالب [۷] بخش‌های مختلف وبسایت را مورد بررسی قرار داد و از درستی و هم فرمی آنها اطمینان حاصل کرد.
همچنین می‌توان با استفاده از مکانیزم‌های ارائه شده، یکسانی و هم فرمی طراحی و look & feel المان‌های مورد استفاده و برندینگ را مورد بررسی قرار داد.
مزایای استفاده از این ابزار تست عبارتند از:

- افزایش سرعت آزمون رابط کاربری وب با نتایجی قابل اطمینان و درک.
- داشتن عملکرد و رابط کاربری یکسان و یک فرم در محیط‌ها و بخش‌های مختلف.
- فرایند user-friendly آزمون رابط کاربری.
- میزان دقت و حساسیت قابل تنظیم در تست‌ها برای انجام تست‌هایی مطابق با نیازها و دقت مورد انتظار.
- افزایش کیفیت محصولات ارایه شده و تجربه‌ی کاربری.



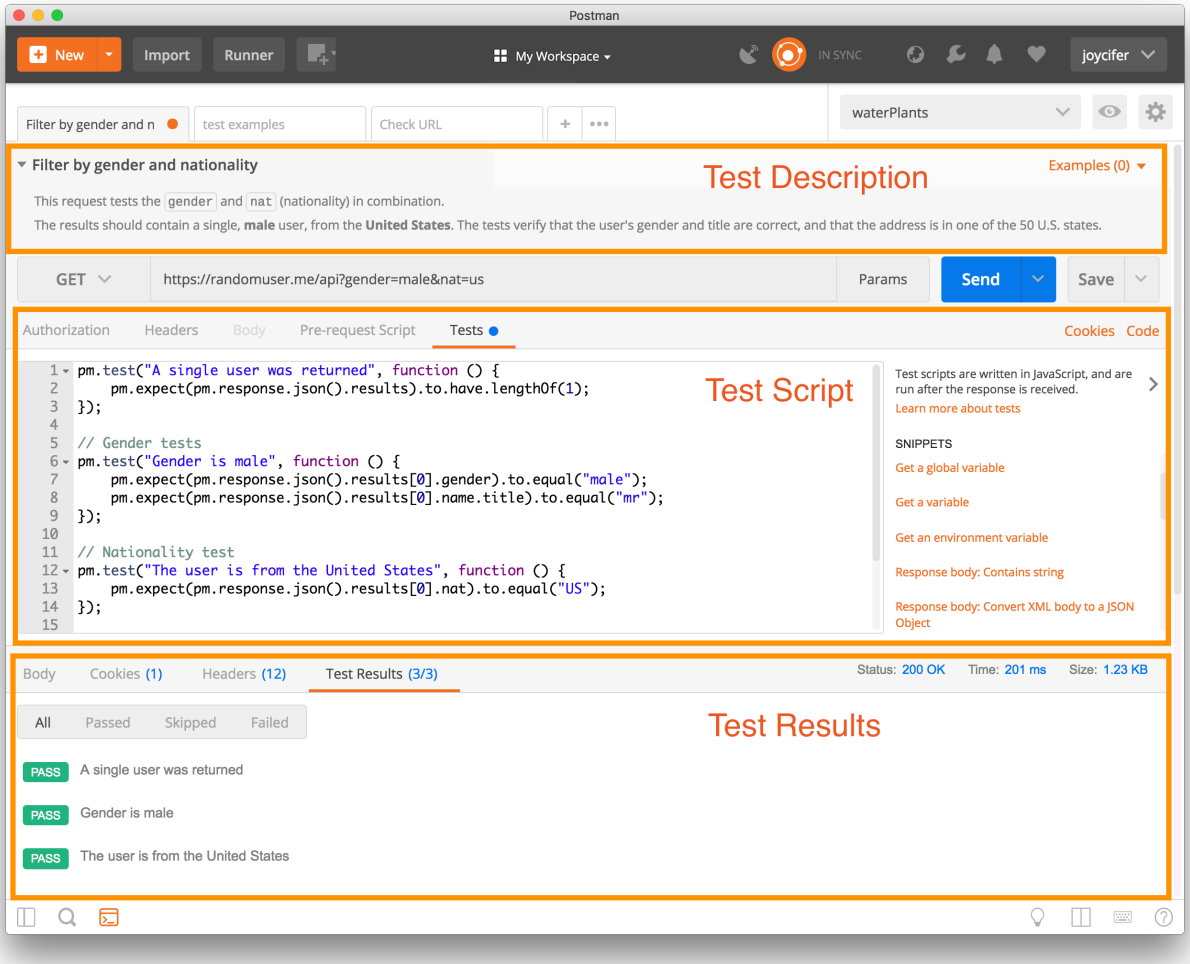
تصویر ۵: فرایند آزمون رابط کاربری وب با استفاده از ابزار TestingWhiz [۸]

[5] navigation
[6] screen layout
[7] theme
[۸] منبع تصویر: https://www.testing-whiz.com/web-ui-comparison-and-functional-test-automation

۲.۲ تست API ، Postman

ابزار Postman یک محیط توسعه‌ی API یا به عبارتی دیگر یک ADE است [۹] .
این ابزار امکانات مناسبی در کنار محیط توسعه‌ی API برای تست خودکار آن ارائه می‌دهد.
ابزار Postman مجموعه‌ی کوچکی از خدمات API ها را یک collection و مجموعه‌ای از collection ها را یک folder می‌نامد.
همچنین این ابزار امکانی ارائه می‌کند که به وسیله‌ی آن می‌توان یک تست را به یک collection یا folder تخصیص داد و به این صورت با اجرای هریک از API های آن مجموعه، تست مذکور اجرا خواهد شد. به این ترتیب امکان reuse برای تست‌های نوشته شده وجود خواهد داشت.
می‌توان با استفاده از ابزار newman تست‌های نوشته شده خودکار را در فرایند CI/CD وارد کرد.

- تست‌هادر ابزار postman با زبان JavaScript نوشته می‌شوند.
- تست‌ها در محیطی sandbox و غیر از محیط اجرای اصلی API هااجرا می‌شوند و از این رو انجام تست‌ها هیچ‌گونه مشکلی برای محیط اصلی ایجاد نخواهد کرد.
- ین ابزار امکانی ارائه می‌کند که به وسیله‌ی آن می‌توان یک تست را به یک collection یا folder تخصیص داد و به این صورت با اجرای هریک از API های آن مجموعه، تست مذکور اجرا خواهد شد.
- تست‌ها بعد از هر فراخوانی API به صورت خودکار اجرا خواهند شد و نتایج آنها برای تحلیل و آمارگیری ثبت می‌شود.



تصویر ٦: تصویری از پنل تست در ابزار postman [۱۰]

[9] API Development Environment

[۱۰] منبع تصویر: https://www.getpostman.com/docs/v6/postman/scripts/test_scripts

۳.۲ تست مقاومت در برابر اشکال در محیط ابر، سرویس‌های Netflix Simian Army

Netflix Simian Army شامل سرویس‌های Chaos Monkey ، Janitor Monkey و Conformity Monkey است. در ادامه به توضیح هر یک می‌پردازیم: [۴]

۱.۳.۲ Chaos Monkey

در سیستم‌ها و محیط‌های ابری معمولا چندین مولفه و زیر سیستم در تعامل با یکدیگر مغول به کار هستند یا برای توزیع بار از یک مولفه چندین نمونه در حال فعالیتند.

در چنین سیستم‌هایی مولفه‌ها باید به نحوی طراحی شوند که در صورتی که یک مولفه از کار افتاد کل سیستم فلج نشود و باقی مولفه‌ها در حد امکان به فعالیت و خدمت رسانی ادامه دهند.

Chaos Monkey در ساعات کاری روزهای غیر تعطیل به صورت تصادفی یکی از مولفه‌های سیستم را انتخاب کرده و آن را از کار می‌اندازد تا رفتار باقی مولفه‌ها در قبال آن مورد بررسی قرار بگیرد.

علت این زمان بندی‌هم آن است که پرسنل نگهداری در حال کار باشند تا در صورت بروز مشکل بتوانند آن را رفع کنند. [۲]

۲.۳.۲ Janitor Monkey

در سیستم‌های ابری منابع تقریبا غیرمحدودی در اختیار نرم‌افزارها قرار دارد از این رو به سادگی می‌توان در مدیریت این منابع سردرگم شد و نسخه‌های قدیمی نرم‌افزار در حال اجرا، داده‌های غیر کاربردی و نسخه‌های پشتیبانی که دیگر مورد نیاز نیستند را کماکان نگهداری کرد. این فضای اشغال شده‌ی اضافی تنها باعث اعمال هزینه‌های بیشتر به تیم توسعه‌ی نرم‌افزار خواهد شد.

Janitor Monkey به صورت پیش‌فرض ساعت ۱۱ صبح روزهای کاری در بین منابع سیستم شروع به گردش می‌کند و هر منبع را در مقابل تعدادی از شرایط می‌سنجد. اگر هر یک از شرایط این موضوع که این منبع دیگر مورد نیاز نیست را به دست بدهند، آن منبع را برای پاک کردن نشانه گذاری می‌کند و پیامی به مالک منبع (در قالب ایمیل) می‌فرستد. طی این پیام به مالک منبع تاکید می‌شود که منبع بنا به شرایط محقق شده، اضافی‌است و ظرف مدت مشخصی که به صورت پیش‌فرض ۳ روز کاری است پاک خواهد شد.

مالک منبع می‌تواند از طریق یک درخواست REST جلوی این پاک شدن را بگیرد یا آن را پیش از موعد انجام دهد.

در صورتی که Janitor Monkey به منبعی برخورد کند که برای پاک شدن نشانه‌گذاری شده و زمان پاک شدن آن گذشته است، در صورتی که state آن منبع عوض نشده باشد آن را پاک می‌کند.

شرایط پاک شدن یک منبع و بازه‌های زمانی و زمان شروع به کار Janitor Monkey مطابق نیازهای کسب و کاری قابل تغییر است. [۳]

۳.۳.۲ Conformity Monkey

در محیط‌های ابری و self-scaling ممکن است بنا به دلایلی مانند عدم دانش کافی انسانی یا فراموشی نمونه‌هایی از محصول شروع به فعالیت کنند که مطابق الگوهای به روز طراحی نشده‌اند یا مشکلاتی دارند که با قواعد conformity که قابل تنظیم هستند، هم‌خوانی ندارند.

Conformity Monkey در روزهای کاری، هر ساعت یک بار به دنبال چنین نمونه‌هایی می‌گردد و در صورتی که نمونه‌ای را پیدا کرد که در شرایط آن صدق می‌کند، اطلاعات شرایط و نمونه‌ی پیدا شده و سیستم آن نمونه را به مالک سیستم و منبع ارسال می‌کند. به این ترتیب فرد مسئول می‌تواند اقدامات لازم را برای حل موضوع در دست اقدام قرار دهد.

شرایط conformity منبع و زمان کار Conformity Monkey مطابق نیازهای کسب و کاری قابل تغییر است. [۱]

۳ پرسش سوم

ابزارهای مدیریت پیکربندی معرفی شده را با یکدیگر مقایسه می‌کنیم: [۵]

	Ansible	Puppet	Chef
زبان	Python	Clojure C++،	Ernalng، Ruby
گواهی استفاده(license)	GPLv3+	Apache	Apache 2.0
احراز هویت دوطرفه	بله	بله	بله
رمزگذاری	بله	بله	بله
تایید بدون تغییر	بله	بله	بله
بدون agent	بله	خیر	خیر
رابط کاربری گرافیکی	بله	بله	بله
زبان توصیف سیستم	Yaml	a custom language	Ruby
تاریخ انتشار	۲۰۱۲	۲۰۰۵	۲۰۰۹
آخرین به روزرسانی	ژانویه ۲۰۱۸	مه ۲۰۱۷	سپتامبر ۲۰۱۷

- احراز هویت دو طرفه به معنی آن است که server ، client خود را تایید هویت کند و به صورت برعکس client ، server خود را هویت سنجی و تایید کند.
- agent داشتن به معنی این است که سرویس برای ارائه‌ی خدمات خود به یک سری پردازه‌ی daemon تحت عنوان agent در سیستم وابسته است.
- تایید بدون تغییر به این معنی است که سیستم بتواند اینکه یک المان یا منبع مطابق قوانین سیتم یا محیط رفتار می‌کند یا پیکربندی شده است را بدون تغییر دادن آن المان و صرفا با خواندن اطلاعات آن تشخیص دهد.
- هرسه‌ی این سرویس‌ها روی سیستم‌های عامل و پلتفرم‌های رایج امروزی تست شده و کارایی مناسبی دارند.

۱.۳ توضیحات تکمیلی

۱.۱.۳ Ansible

سیستم مدیریت پیکربندی Ansible فعالیت‌های استقرار چند گره‌ای [۱۱] اجرای تک کاره (adhoc) وظایف و مدیریت پیکربندی را با هم ترکیب کرده‌است. گره‌ها را از طریق SSH مدیریت می‌کند و برای این کار نیاز دارد که روی تک‌تک آنها Python نصب شده باشد.

۲.۱.۳ Puppet

برای مدیریت گره‌ها از فراخوانی REST و پارادایم کارفرما-کارگزار استفاده می‌کند. از مفهومی به نام resource abstraction layer استفاده می‌کند و به آن این امکان را می‌دهد که پیکربندی سیستم را به زبانی سطح بالا انجام دهد.

۳.۱.۳ Chef

می‌توان از Chef در حالت‌های کارفرما-کارگزار یا تنها (solo) استفاده کرد

[11] Multi-node deployment

۴ پرسش چهارم

SCM برای موفقیت یک پروژه‌ی نرم‌افزاری در تمامی متودولوژی‌های توسعه‌ی نرم‌افزار نقش کلیدی بازی می‌کند. اصول اصلی SCM یعنی شناسایی، کنترل، مشاهده و گزارش ضامن یکپارچگی و کیفیت محصول در دست تولید هستند.

می‌توان SCM را متناسب با نیاز فرایند و متودولوژی تولید نرم‌افزار به شیوه‌های متفاوتی پیاده سازی کرد (مادامی که اصول آن پابرجا بمانند).

در متودولوژی‌های اجایل بایستی مکانیزم‌های اعمال SCM دستخوش تغییراتی شوند که آن را با سرعت تغییرات و گردش‌های متودولوژی هماهنگ کنند.

به این صورت باید تمرکز بر پیاده‌سازی SCM به نحوی باشد که از release های کوتاه‌تر و سریعتر پشتیبانی کند، با ساخت‌های پیوسته [۱۲] هماهنگ باشد، از فضاهای کاری اشتراکی بین افراد و بخش‌های مختلف پروژه و تیم پشتیبانی کند، با چند شاخه‌شدن و ادغام‌های مکرر کد در روش‌های اجایل هماهنگ باشد، از خودکارسازی و اتوماسیون بالایی برخوردار باشد تا بتواند با سرعت بالای متودولوژی همراه شود، نقش‌های CM بایستی توسط پرسنل deployment انجام شوند و معیارهایی قابل اندازه‌گیری CM برای محصولات، فعالیت‌ها و وظایف تعریف و مدیریت و گزارش‌گیری شوند.

۱.۴ گام‌های تغییر SCM برای متودولوژی‌های agile

۱.۱.۴ پشتیبانی از گام‌های کوتاه‌تر

SCM باید از گام‌ها و افزایش‌ها و گردش‌های کوتاه‌تر پشتیبانی کند. به این صورت فرایند SCM باید بتواند به سرعت به شناسایی و کنترل تغییرات و المانها در سیستم نرم‌افزاری بپردازد و گلوگاهی برای سرعت متودولوژی اجایل نباشد در عین حال باید توان مورد نیاز برای مدیریت تغییرات را کمینه کند.

۲.۱.۴ پشتیبانی از گام‌های کوتاه‌تر

برای ارائه‌ی ساخت‌های پیاپی می‌توان از ابزارهایی که به همین منظور طراحی شده‌اند استفاده کرد و به وسیله‌ی این ابزارها سرعت بالای تغییرات و ساخت‌ها را مدیریت کرد.

همچنین نیاز به درک درستی از فعالیت‌های فعلی ساخت و انطباق آنها با انواع مختلف و مورد نیاز ساخت‌های موجود و سطوح مختلف آنها محسوس است.

این فعالیت به SCM در شناسایی محل هدررفت منابع و توان کمک شایانی می‌کند.

۳.۱.۴ محیط‌های کاری، چند شاخه شدن و ادغام کد

سرعت بالای توسعه در متودولوژی‌های اجایل و ابعاد کار نیاز به وجود محیط‌های جدای کار برای افراد و گروه‌های مختلف و محیط‌های اشتراکی لازم بین افراد و گروه ها و همچنین مکانیزم‌هایی برای ادغام این محیط‌ها را محسوس می‌کند.

ابزارهای SCM امروزه با امکان CM-coop environments عرضه می‌شوند که به تیم‌ها این امکان را می‌دهند که ساخت محیط جدید و ادغام محیط‌ها را به سادگی و بدون نیاز به تحمل سربار نصب ابزارهای مدیریتی به ازای هر محیط انجام دهند.

۴.۱.۴ خودکارسازی

سرعت بالای متودولوژی‌های چابک نیاز به سرعت بالای تمامی فعالیت‌های جانبی انجام شده در کنار آنها را ایجاب می‌کند. فعالیت‌های SCM هم از این قاعده مستثنی نیستند. از این رو این فعالیت‌ها باید حتی الامکان تماما خودکار انجام شوند و دستی نباشند.

۵.۱.۴ انقال مدیریت تغییر به مرحله‌ی برنامه‌ریزی برای گردش

تغییرات اعمال شده در پروژه نباید تا قبل از iteration بعدی رسیدگی شوند. جلسات رسیدگی به تغییرات تبدیل به جلسات روزانه با سایر اعضای تیم می‌شود و مدیریت تغییرات به مرحله‌ی برنامه‌ریزی برای گردش بعدی انتقال می‌یابد.

۶.۱.۴ نقش‌های SCM

این نقش‌ها در یک تیم چابک بایستی توسط افرادی انجام شود که بیشتر با تغییرات نهایی یک سیستم در حال توسعه و سرویس در ارتباط هستند. این افراد پرسنل deployment و نگهداری از سیستم هستند. این افراد در جریان تغییرات کلی و migration های قبلی سیستم هستند و تغییرات بعدی نیز باید به دست آنها انجام و اعمال شود.

[12] Continuous Builds

۷.۱.۴ تعیین و تغییر معیارهای SCM برای تشخیص هدررفت‌ها

برای تشخیص هدررفت‌ها در توان و منابع تیم توسعه‌ی چابک باید معیارهای SCM متناسب با شرایط چابک جدید تغییر کند. در اینجا نمونه‌ای از این معیارهای تغییریافته آورده شده است:

- مقایسه‌ی user story یا requirment هایی که برای توسعه و ارایه برنامه‌ریزی و اولویت بندی شده بودند با آنچه واقعا توسعه یافته و ارایه شده.
- شناسایی تغییرات اضافه‌ای که برنامه ریزی نشده بودند ولی انجام شدند.
- اندازه گیری زمان صرف شده برای بازیابی از ساخت‌هایی که با شکست مواجه شده‌اند.
- ...

[٦]

مراجع

[1] "**What is Conformity Monkey?**", Netflix, retrieved 12 khordad 1397

https://github.com/Netflix/SimianArmy/wiki/Conformity-Home

[2] "**What is Chaos Monkey?**", Netflix, retrieved 12 khordad 1397

https://github.com/Netflix/SimianArmy/wiki/Chaos-Monkey

[3] "**What is Janitor Monkey?**", Netflix, retrieved 12 khordad 1397

https://medium.com/netflix-techblog/the-netflix-simian-army-16e57fbab116

[4] "**The Netflix Simian Army**", Medium.com, retrieved 12 khordad 1397

https://github.com/Netflix/SimianArmy/wiki/Janitor-Home

[5] "**Comparison of open-source configuration management software**", Wikipedia.org, retrieved 12 khordad 1397

https://en.wikipedia.org/wiki/Comparison_of_open-source_configuration_management_software

[6] "**Applying Configuration Management to Agile Teams**", Mario Moreira, retrieved 12 khordad 1397

https://www.cmcrossroads.com/article/applying-configuration-management-agile-teams