مروری نظام‌مند بر مدل‌سازی قابلیت اطمینان نرم‌افزار

سیده‌مهسا هاشمی‌مجد [۱]، عباس رسول‌زادگان [۲،*]، زهرا قویدل یزدی [۳]

اطلاعات مقاله

دریافت مقاله: ۱۳۹۴/۰۲/۱۱
پذیرش مقاله: ۱۳۹۴/۰۹/۲۳

واژگان کلیدی:
قابلیت اطمینان،
مدل‌سازی قابلیت اطمینان،
پیش‌بینی قابلیت اطمینان،
تخمین قابلیت اطمینان،
اندازه‌گیری کمّی قابلیت اطمینان.

چکیده

امروزه سیستم‌های نرم‌افزاری نقش مهمی در بسیاری از کاربردهای حساس و بحرانی ایفا می‌کنند. گاهی رخ دادن خطا در نرم‌افزار می‌تواند باعث ضررهای مالی و حتی در برخی کاربردها باعث ضررهای جانی گردد. به همین دلیل تضمین قابلیت اطمینان به عنوان یک نیازمندی غیرکارکردی (کیفی) از اهمیت بسزایی برخوردار است. یکی از اقدامات کلیدی در تضمین عملکرد عاری از خطای نرم‌افزار، اندازه‌گیری کمّی میزان قابلیت اطمینان مورد نیاز است. تاکنون مدل‌های متعددی برای کمّی‌سازی قابلیت اطمینان نرم‌افزار ارائه شده است. هر کدام از این مدل‌ها دارای مزایا و محدودیت‌هایی هستند، لذا انتخاب مناسب‌ترین مدل در کاربردهای مختلف اهمیت فراوانی دارد. در این مقاله ابتدا به کمک یک روش نظام‌مند تحقیق، ضمن مرور دسته‌بندی‌های پُراستناد، یک مدل جامع و به‌روز برای دسته‌بندی روش‌های مختلف مدل‌سازی قابلیت اطمینان نرم‌افزار ارائه می‌نماییم. هدف از ارائه دسته‌بندی پیشنهادی، تسهیل فرآیند شناسایی و انتخاب مناسب‌ترین مدل قابلیت اطمینان در کاربردهای مختلف می‌باشد. در نهایت بر مبنای تحلیل مزایا و محدودیت‌های مدل‌های موجود، به مقایسه تطبیقی روش‌ها و بیان برخی از چالش‌ها و مسائل باز مرتبط می‌پردازیم.

۱- مقدمه

با توجه به نقش مهمی که امروزه نرم‌افزار در سیستم‌های مختلف دارد، تولید نرم‌افزارهای باکیفیت از اهمیت روزافزونی برخوردار است [۱و۲]. استاندارد «ایزو ۹۱۲۶» [۳] جنبه‌های مختلفی از کیفیت را معرفی کرده است. قابلیت اطمینان یکی از جنبه‌های مهم کیفیت نرم‌افزار است [۴]. بر اساس استاندارد IEEE ۱۶۳۳ که در سال ۲۰۰۸ ارائه شده است، قابلیت اطمینان نرم‌افزار عبارتست از احتمال بدون شکست بودن عملکرد سیستم برای یک بازه‌ی زمانی معین و در یک محیط معین.

امروزه یکی از مقولات مهم و چالش‌برانگیز مهندسی نرم‌افزار، اندازه‌گیری کمّی جنبه‌های مختلف کیفیت است [۴ و ۵]. مهندسی قابلیت اطمینان نرم‌افزار، به طور عام، و اندازه‌گیری کمّی آن، به طور خاص، در بسیاری از کاربردها، به‌ویژه در سیستم‌هایی که با جان انسان سر و کار دارند از اهمیت بسزایی برخوردار است [۵]. یکی از ملزومات

*. پست الکترونیک نویسنده مسئول: rasoolzadegan@um.ac.ir
۱. فارغ‌التحصیل کارشناسی ارشد، مرکز آموزش‌های الکترونیکی، دانشگاه فردوسی مشهد
۲. استادیار دانشکده مهندسی، دانشگاه فردوسی مشهد
۳. دانشجوی کارشناسی ارشد، مرکز آموزش‌های الکترونیکی، دانشگاه فردوسی مشهد

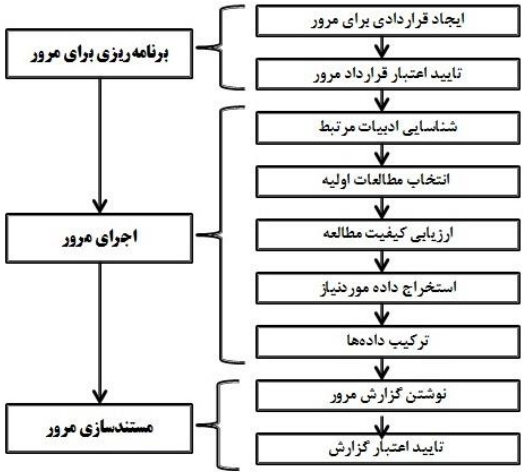
مهندسی قابلیت اطمینان، مدل‌سازی این نیازمندی می‌باشد. بر اساس یک مطالعه‌ی قاعده‌مند که در سال ۲۰۱۴ صورت گرفته است [۶]، ۶۳٪ درصد از کارهای انجام شده در زمینه‌ی مهندسی قابلیت اطمینان نرم‌افزار، به ارائه‌ی یک مدل جدید برای اندازه‌گیری قابلیت اطمینان و یا ارتقای مدل‌های قبلی اختصاص یافته است. در واقع در طی ۳۰ سال گذشته بیش از ۱۰۰ مدل مختلف در این زمینه پیشنهاد شده است [۷ و ۸]، اما مدل واحدی وجود ندارد که در تمام موقعیت‌ها و کاربردها مناسب باشد [۲].

با توجه به تعدد روش‌های مدل‌سازی و اندازه‌گیری موجود و افزایش روزافزون تعداد آن‌ها، نیاز به دسته‌بندی، مقایسه‌ی روش‌ها و ارائه‌ی الگوریتمی جهت انتخاب مناسب‌ترین روش مدل‌سازی و اندازه‌گیری قابلیت اطمینان نرم‌افزار در کاربردهای مختلف، بیش از پیش احساس می‌شود. در این مقاله این هدف دنبال شده است؛ زیرا دسته‌بندی مدل‌های موجود بر اساس معیارهای مختلف، به ما یک دید روشن از روش‌های موجود خواهد داد و در نتیجه می‌توان به ویژگی‌ها، مزایا، محدودیت‌ها و مفروضات هر دسته پی برد تا در عمل نیز بتوان از این مدل‌ها به صورت کارا و بهینه با توجه به ویژگی‌های موردنیاز، اهداف و محدودیت‌های هر پروژه استفاده نمود.

ادامه این مقاله به این صورت سازماندهی شده است: ابتدا در بخش دوم روش تحقیق انجام گرفته در این مقاله تشریح می‌شود. در بخش سوم، در ابتدا با شناسایی مراجع معتبر در زمینه‌ی مدلسازی قابلیت اطمینان نرم‌افزار بر مبنای یک روش نظام‌مند تحقیق که در بخش دوم ارائه شده است، دسته‌بندی‌های مختلفِ موجود معرفی می‌شوند. سپس با انتخاب چند نمونه از دسته‌بندی‌های معتبر و پُراستناد و ترکیب آن‌ها با یکدیگر، یک دسته‌بندی جامع در زمینه قابلیت اطمینان نرم‌افزار ارائه خواهد شد. در بخش چهارم، الگوریتمی برای انتخاب مناسب‌ترین روش مدل‌سازی قابلیت اطمینان از بین روش‌های موجود در دسته‌بندی پیشنهادی، معرفی می‌گردد. در بخش پنجم، با تمرکز ویژه بر دسته‌بندی پیشنهادی، به مقایسه‌ی تطبیقی روش‌های مدل‌سازی و اندازه‌گیری کمّی قابلیت اطمینان نرم‌افزار پرداخته و بر این مبنا برخی از مهم‌ترین چالش‌ها و مسائل باز این حوزه معرفی می‌شود. در نهایت در بخش ششم، نتیجه‌گیری و کارهای آینده ارائه خواهد شد.

۲- روش تحقیق

در این مقاله برای دریافت درک درستی از تعاریف و روش‌های ارائه شده برای اندازه‌گیری کمّی قابلیت اطمینان نرم‌افزار، مروری نظام‌مند بر ادبیات گذشته انجام شده است. یک مرور نظام‌مند برای تولید خروجی معتبر، منصفانه، عادلانه و قابل‌اعتماد از روش‌های شفاف و قابل تکرار کمک می‌گیرد. در این نوع مرور، ساختار حاکم بر فرآیند تحقیق، مانع از اعمال نظرات شخصی و مهندسی نتایج به سمت نتایج دلخواه پژوهشگر می‌گردد [۱۴-۹]. در شکل (۱) ساختار کلی یک مرور نظام‌مند نشان داده شده است. در ادامه متناظر با هر یک از مراحل ساختار کلی مرور نظام‌مند (شکل (۱))، آنچه که در این پژوهش و مقاله به کار گرفته شده است معرفی و به اختصار توضیح داده می‌شود.



شکل ۱: ساختار کلی یک مرور نظام‌مند [۹]

۲-۱- برنامه‌ریزی برای مرور

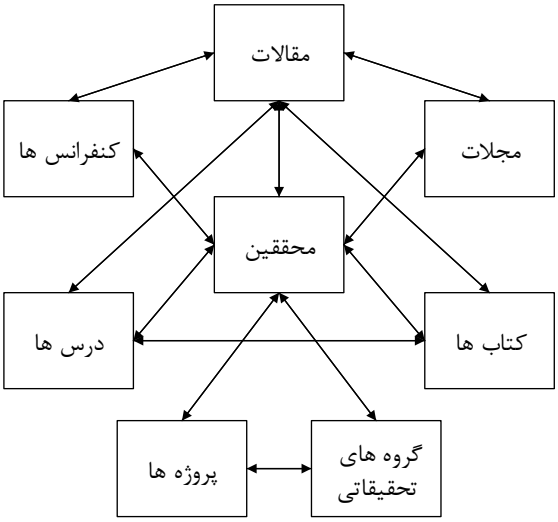
این بخش از دو زیربخش «ایجاد قراردادی برای مرور» و «تایید اعتبار قرارداد مرور» تشکیل شده است.

الف) ایجاد قراردادی برای مرور: در صورتی که موتور جستجوی گوگل را باز کنیم و با کلمات کلیدی خاصی به دنبال منابع مختلف موجود مرتبط با حوزه‌ی پژوهش‌مان

باشیم، مطمئناً این کار مستعد خطا خواهد بود. زیرا این احتمال وجود خواهد داشت که کارهای خوبی در این زمینه از نظر ما پنهان بماند. روشی که ما می‌خواهیم تعریف کنیم و قرارداد کنیم که از آن برای پژوهش خود استفاده کنیم، یک روش نظام‌مند است. به این معنا که اگر نفر دیگری نیز از این مسیر به دنبال منابع مرتبط با حوزه‌ی ما بگردد، بعد از چندین تکرار به منابعی خواهد رسید که ما رسیده‌ایم. قراردادی که در این مقاله برای مرور استفاده شده است، روشی برپایه‌ی تکرار و تکامل است. در راستای پژوهش در حوزه‌ی مربوطه، ما با ۸ موجودیتِ مقالات، مجله‌ها، کنفرانس‌ها، محققین، کتاب‌ها، پروژه‌ها، درس‌ها و گروه‌های تحقیقاتی مواجه هستیم. این‌ها در واقع منابعی هستند که ما می‌توانیم در آن‌ها به دنبال یک کار خوب در حوزه‌ی پژوهشی‌مان بگردیم. البته برای تایید اعتبار هر کدام از این منابع می‌توان معیار ارزیابی تعیین کرد. به عنوان نمونه، به معیارهای ارزیابی مورد استفاده برای مجلات در بخش (۲-۲) اشاره خواهد شد. ارتباطات این ۸ موجودیت در شکل (۲) نشان داده شده است. به عنوان مثال ما می‌توانیم از یک مجله‌ی خوب به تعدادی مقاله‌ی خوب برسیم و بالعکس.

برای شروع فرآیند مرور، ابتدا بهتر است یک کتاب و یا یک مقاله‌ی خوب که حاوی یک مرور خوب و جامع پیرامون موضوع مورد نظر است را انتخاب نماییم. از این راه می‌توانیم به تعاریف و مفاهیم پایه در رابطه با موضوع مورد نظرمان برسیم. از یک کتاب خوب می‌توان به نویسندگان و محققین آن کتاب دسترسی پیدا کرد و با مراجعه به صفحه‌ی آن‌ها در اینترنت، با دروسی که آن‌ها ارائه می‌دهند، سایر مقالاتی که در رابطه با موضوع مورد نظر ارائه داده‌اند، پروژه‌هایی که در رابطه با موضوع مورد نظر در آن شرکت داشته‌اند، گروه‌های تحقیقاتی و ژورنال‌هایی که جزء هیئت تحریریه‌ی آن هستند، آشنا می‌شویم. همچنین می‌توان با مراجعه به بخش مراجع کتاب، با لیستی از مراجع معتبر دیگر آشنا شد. این فرآیند را می‌توان با توجه به شکل (۲) دنبال کرد و در نهایت پس از چندین مرحله تکرار، به اکثر منابع خوب موجود در حوزه‌ی مورد نظر دست خواهیم یافت.

ب) تایید اعتبار قرارداد مرور: قراردادی که در بخش قبلی مطرح شد، بر پایه‌ی تکرار و تکامل است. براساس روش مرور گفته شده، بعد از چندین مرحله تکرار، به منابع مفید و ارزشمندی دست خواهیم یافت که اگرچه نمی‌توان این ادعا را کرد که تمام منابع را در اختیار خواهیم داشت اما می‌توان این ادعا را کرد که به منابع معتبری دست یافته‌ایم؛ زیرا مدام در حال ارزیابی کیفیت منابع مورد نظرمان هستیم. همچنین در این فرآیند، پس از چندین مرحله تکرار، به منابعی می‌رسیم که در مراحل قبل به آن‌ها رسیده بودیم و این خود نشان دهنده‌ی اعتبار منابع انتخابی و در نتیجه اعتبار روش مرور است.



شکل ۲: روش جمع‌آوری منابع

۲-۲- اجرای مرور

این بخش خود از ۶ زیربخش تشکیل شده است که در ادامه متناظر با کاری که در این مقاله صورت گرفته است هریک را شرح خواهیم داد.

الف) شناسایی ادبیات مرتبط: دراین مقاله از کلماتی کلیدی مانند: ارزیابی/اندازه‌گیری کمّی قابلیت اطمینان نرم‌افزار و مدل‌های قابلیت اطمینان برای جستجوی اولیه در منابع استفاده شده است. سپس با بازخوردی که از این منابع گرفته شد، به کلمات کلیدی جدیدی از قبیل انواع مدل‌های اندازه‌گیری قابلیت اطمینان نرم‌افزار (مثلاً مدل‌های رشد قابلیت اطمینان نرم‌افزار) و یا روش‌های انتخاب

مدل مناسب، معیارهای انتخاب مدل مناسب و واژگانی این-چنینی رسیدیم.

ب) انتخاب مطالعات اولیه: در این بخش، با استفاده از کلمات کلیدی که از بخش قبلی بدست آمد، در منابعی نظیر: ««ACM Digital Library»، «Scopus»، «Springer Online Journal Collection»، «Pad- پایگاه اشتراک داده دانشگاه فردوسی مشهد»، « Science Direct»، «Xplore Digital Library IEEE»و نیز « «Google Scholarمرور نظامندی (برپایه‌ی قرارداد مرور که در بخش (۲-۱) معرفی شد) در حوزه‌ی مربوطه انجام شده است و در فرآیند تحقیق، منابع (نظیر مقالات، کتب، پایان‌نامه‌ها و رساله‌ها) به زبان انگلیسی و فارسی، مورد توجه قرار گرفته است. البته در ابتدا تمرکز اصلی ما برروی مرورهای ارزشمند این حوزه بود که یکی از مهم-ترین آن‌ها مرجع [۸] می‌باشد.

ج) ارزیابی کیفیت مطالعه: بخش عمده‌ای از منابع مورد تحقیق را مقالات معتبر تشکیل داده‌اند. برای تایید اعتبار مجلات و مقالات می‌توان از معیارهایی نظیر فاکتور مؤثر ، میزان استناد به آن اثر و رتبه‌بندی مجله‌ی علمی استفاده کرد. همچنین برای تعیین این‌که آیا مقالات بدست آمده حاوی اطلاعات مرتبط هستند، عنوان و چکیده‌ی آن‌ها مورد مطالعه قرار گرفته است. که در نهایت به تعدادی منابع مفید در راستای هدف مقاله رسیده‌ایم که در بخش ۲ در جدول ۱ بخشی از آن‌ها به اختصار آورده شده است.

ه) استخراج داده‌ی مورد نیاز و ترکیب داده‌ها: ما در این قسمت داده‌های موجود و مرتبط با موضعمان را در منابع مربوطه مورد تحلیل قرار داده‌ایم و آن دسته از داده-هایی را که به تکمیل و ارتقاء مرور ما کمک می‌کند، استخراج نموده‌ایم. برخی از منابع را، به دلیل حجم زیادی که دارند، نمی‌توان به طور کامل مورد مطالعه قرار داد و البته نیازی هم به این کار نیست و صرفاً باید بخش‌هایی که مربوط به داده‌های مورد نیاز ما هست شناسایی کرده و داده‌های مورد نظر را استخراج نمود. داده‌هایی که در این مرحله استخراج شد، عبارتند از: مفاهیم اولیه مربوط به قابلیت اطمینان نرم‌افزار و مدلسازی قابلیت‌اطمینان، انواع دسته‌بندی‌های مدل‌های قابلیت اطمینان نرم‌افزار (جدول ۱)، شرح هریک از مدل‌های ارائه شده و روش انتخاب مدل مناسب. ما هریک از این داده‌ها را با بازخوردی که از منابع مختلف گرفته‌ایم، در قسمت‌های بعدی این مقاله مورد تحلیل قرار داده‌ایم. پس از انتخاب داده‌های مورد نیاز، آن‌ها را برحسب نیاز و براساس درجه‌ی اهمیتی که دارند با هم ترکیب کردیم و به یک دسته‌بندی جدید برای روش‌های مدلسازی قابلیت اطمینان نرم‌افزار رسیدیم که در بخش (۱۳-۳) آن را مفصلاً شرح خواهیم داد.

۲-۳- مستندسازی مرور

این بخش شامل نوشتن گزارش مرور و تایید گزارش مرور است. ما ابتدا گزارشی اولیه از مرور را تهیه کردیم و پس از بازبینی آن توسط اعضای تیم تحقیقاتی (در این مرحله ابهامات و نارسایی‌های موجود در گزارش برطرف شد)، آن را براساس استانداردهای گفته شده در «مجله‌ی علمی پژوهشی مدلسازی در مهندسی»، تنظیم کردیم. گزارش حاصل مجدداً توسط اعضای تیم تحقیقاتی مورد بازبینی قرار گرفت و در این مرحله از بازبینی، تمرکز خود را بر روی رفع اشکالات نگارشی قرار دادیم.

۳- مروری نظام‌مند بر دســـته‌بندی‌های مختلف مدل‌های قابلیت اطمینان نرم‌افزار

سؤال‌هایی مانند «چرا سیستم نرم‌افزاری دچار شکست می‌شود؟» یا «چگونه می‌توان قابلیت اطمینان یک سیستم نرم‌افزاری را اندازه‌گیری کرد؟»، مهندسین نرم‌افزار را به سمت مدل‌سازی قابلیت اطمینان نرم‌افزار سوق داده است. مدل‌های قابلیت اطمینان نرم‌افزار ابزارهایی قدرتمند برای پیش‌بینی، کنترل، ارزیابی و اندازه‌گیری کمّی قابلیت اطمینان نرم‌افزار هستند. از دهه‌ی ۱۹۷۰ تا اکنون بیش از ۱۰۰ مدل در این زمینه پیشنهاد شده است و هر ساله این مدل‌ها در حال پیشرفت هستند. ولی هنوز مدلی وجود ندارد که بتواند در همه موارد به کار رود [۱۵]. در جدول ۱ این روش‌ها در قالب دسته‌بندی‌های متنوعی ارائه

شده‌اند.

جدول ۱: برخی از روش‌های دسته‌بندی مدل‌های قابلیت اطمینان نرم‌افزار

ردیف	دسته‌بندی
۱	دسته‌بندی موسا و اکیوموتو [۸]
۲	دسته‌بندی لیو [۸]
۳	دسته‌بندی بر مبنای فاز توسعه نرم‌افزار [۱۶]
۴	دسته‌بندی بر مبنای نیازهای داده‌ای [۱۷، ۱۸]
۵	دسته‌بندی بر مبنای میزان قطعیت [۷]
۶	دسته‌بندی بر مبنای دوره‌ی چرخه‌ی حیات توسعه‌ی نرم‌افزار [۱۹]
۷	دسته‌بندی بر مبنای خرابی‌های نرم‌افزار [۷]
۸	دسته‌بندی بر مبنای دامنه ورودی و دامنه زمان [۲۰]
۹	دسته‌بندی بر مبنای تاریخچه‌ی شکست [۲۱]
۱۰	دسته‌بندی بر مبنای زمان اندازه‌گیری [۲۳، ۲۲]
۱۱	دسته‌بندی بر مبنای چرخه زندگی [۲۴]
۱۲	دسته‌بندی بر مبنای کارکرد [۲۲]
۱۳	دسته‌بندی معرفی شده در مرجع [۶]

همان‌طور که در جدول ۱ مشاهده می‌شود، یکی از این دسته‌بندی‌ها (ردیف ۳)، مدل‌های قابلیت اطمینان را بر مبنای فاز توسعه‌ی نرم‌افزار تقسیم‌بندی مینماید. در رابطه با این دسته‌بندی مطالعات فراوانی انجام شده است [۲۵ و ۲۶]. در ادامه، ابتدا، به اختصار، سایر دسته‌بندی‌های موجود را معرفی می‌نماییم و سپس به تفصیل، به معرفی دسته-بندی پیشنهادی که با محوریت این دسته‌بندی ارائه شده است، می‌پردازیم.

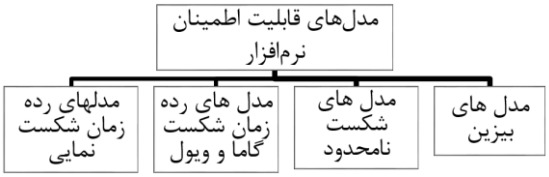
۳-۱- دسته‌بندی موسا و اکیوموتو

یکی از اولین دسته‌بندی‌های ارائه شده برای مدل‌های قابلیت اطمینان نرم‌افزار، توسط موسا و اکیوموتو ارائه شده است [۸]. این رده‌بندی بر اساس ۵ فاکتورِ حوزه‌ی زمان (کل زمان سپری شده و یا زمان اجرا)، دسته (تعداد شکست‌های محدود و یا نامحدود)، نوع توزیع (توزیع پواسن و یا توزیع دو جمله‌ای)، رده و خانواده می‌باشد. اهمیت این دسته‌بندی به خاطر این است که جزء اولین دسته‌بندی-هاست و مطالعات و پژوهش‌هایی که بعداً در این حوزه انجام شده‌است، از آن باز خورد گرفته‌اند.

۳-۲- دسته‌بندی لیو

در مرجع [۸]، که به عنوان یک مرجع پایه و اصلی برای مبحث اندازه‌گیری قابلیت اطمینان محسوب می‌شود و بازبینی مفصلی از مدل‌های تخمین قابلیت اطمینان نرم‌افزار در آن ارائه شده است، دسته بندی جدیدی معرفی گردیده است (شکل (۳)).

بر اساس این مرجع، مدل‌های رده‌ی زمان شکست نمایی، جزء مهم‌ترین مدل‌ها هستند که تا آن زمان بیشترین پژوهش‌ها در مورد آن‌ها بوده است. معروف‌ترین مدل‌های موجود در این دسته مدل جلینیسکی - موراندا، مدل فرآیند پواسن نامتجانس، مدل زمان اجرای پایه موسا، مدل فوق نمایی و مدل اشنیدویند می‌باشد. در این مدل‌ها تابع شدت شکست، نمایی است.



شکل ۳: دسته‌بندی لیو

در مدل‌های رده‌ی زمان شکست گاما و ویبول، برای توصیف تابع شدت شکست، به جای توزیع نمایی از توزیع گاما و ویبول استفاده می‌شود. معروف‌ترین مدل‌های موجود در این دسته عبارتند از: مدل ویبول، مدل رشد قابلیت اطمینان S شکل.

مدل‌های رده‌ی شکست نامحدود نیز مدل‌هایی هستند که تعداد شکست‌ها در آن‌ها محدود نیست و در زمان بینهایت تابع متوسط تعداد شکست‌ها در آن‌ها به سمت بینهایت میل می‌کند ($\lim_{t\to\infty} \mu(t) = \infty$)؛ بدین معنا که نرم‌افزار هیچ وقت بدون اشکال نخواهد بود و دلیل آن هم می‌تواند پیدا شدن اشکالات جدید در حین فرآیند بازیابی و تصحیح خطا باشد. معروف‌ترین مدل‌های موجود در این دسته عبارتند از: مدل دوان ، مدل هندسی، مدل پواسن لگاریتمی موسا و اوکیوموتو.

آخرین دسته از مدل‌ها مربوط می‌شود به معرفی مدل‌های بیزین. در دسته‌های گذشته، قابلیت اطمینان نرم‌افزار تنها در صورتی تغییر می‌کند که یک خطا بروز کند و در اکثر

آن‌ها تاثیر خطاها یکسان در نظر گرفته می‌شود. در مدل-های بیزین در صورتیکه هیچ شکستی گزارش نشود نیز قابلیت اطمینان افزایش می‌یابد. در واقع در این مدل‌ها قابلیت اطمینان تابعی است از تعداد اشکالات کشف شده و تعداد عملیات بدون شکست نرم‌افزار. همچنین در این مدل‌ها تاثیر خطاها یکسان در نظر گرفته نمی‌شود و تاثیر خطاها بسیار مهم‌تر از تعداد آن‌هاست [۸] و [۲۷]. مدل معروف در این دسته، مدل رشد قابلیت اطمینان لیتل وود – ورال است که برای مطالعه‌ی بیشتر شما را به مرجع [۸] ارجاع می‌دهیم.

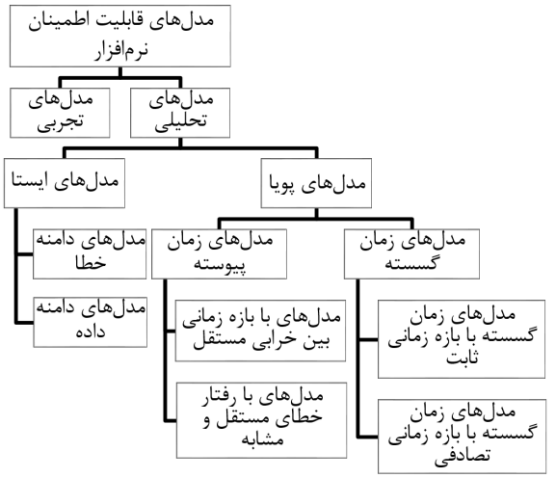
۳-۳- دسته‌بندی بر مبنای نیازهای داده‌ای

همان‌طور که در شکل (۴) مشاهده می‌شود، مدل‌های قابلیت اطمینان نرم‌افزاری، بر مبنای نیازهای داده‌ای به دو دسته تقسیم می‌شوند: مدل‌های تجربی و مدل‌های تحلیلی [۱۷، ۲۰، ۱۸].

برای اندازه‌گیری قابلیت اطمینان نرم‌افزار می‌توان از مدل‌های تجربی استفاده کرد. در این مدل‌ها، N نسخه از نرم‌افزار بکار گرفته می‌شود، و در لحظه‌ی t بررسی می‌گردد که چند نسخه از این تعداد، سالم هستند.

مدل‌های تحلیلی نیاز به مقداری داده جمع‌آوری شده در زمان خرابی نرم‌افزار دارند. این مدل‌ها با در نظر گرفتن تعدادی فرض و توزیع مناسب داده‌ها، قابلیت اطمینان نرم‌افزار را اندازه‌گیری می‌نمایند. مدل‌های تحلیلی به دو دسته تقسیم می‌شوند: مدل‌های ایستا و پویا.

مدل‌های ایستا، رفتار وابسته به زمان خرابی‌های نرم‌افزار را در نظر نمی‌گیرند. این مدل‌ها بر اساس نوع داده‌ای که در توسعه‌ی مدل مورد استفاده قرار می‌دهند، به دو دسته تقسیم می‌گردند: مدل‌های دامنه خطا و مدل‌های دامنه داده. مدل‌های دامنه خطا از مجموعه‌های مختلف خطاها استفاده می‌کنند. مدل‌های دامنه داده از مجموعه‌های مختلف داده‌های ورودی و خرابی‌های مشاهده شده‌ی نرم‌افزار استفاده می‌کنند.



شکل ۴: دسته‌بندی بر مبنای نیازهای داده‌ای

مدل‌های پویا وابستگی زمانی خرابی‌های نرم‌افزار را در نظر می‌گیرند. داده‌های خرابی نرم‌افزار در دوره‌های زمانی مختلف جمع‌آوری می‌شوند. بسته به بازه‌های زمانی مورد استفاده، مدل‌های پویا به دو دسته‌ی مدل‌های زمان پیوسته و مدل‌های زمان گسسته تقسیم می‌شوند.

در مدل‌های زمان گسسته، تعداد خرابی‌های نرم‌افزار در طول بازه‌های زمانی و یا در مراحل مختلف تست، ثبت می-گردند. بازه‌های زمانی می‌توانند ثابت یا تصادفی باشند. بر این مبنا می‌توان این دسته از مدل‌ها را به دو زیر شاخه تقسیم کرد: مدل‌های زمان گسسته با بازه زمانی تصادفی و مدل‌های زمان گسسته با بازه زمانی ثابت.

در مدل‌های زمان گسسته با بازه زمانی تصادفی، هر بازه یک مرحله است که زنجیره‌ای از تست‌ها در حال اجرا هستند و تعداد خرابی‌ها ثبت می‌گردد. داده‌ای که جمع‌آوری می‌گردد عبارتند از تعداد اجراهای تست، تعداد خرابی‌ها و طول هر بازه. در مدل‌های زمان گسسته با بازه زمانی ثابت، طول بازه‌های زمانی ثابت می‌باشد. علاوه بر این در این مدل‌ها فرض می‌شود که تعداد خطاهایی که در هر بازه زمانی رخ می‌دهد، مستقل از بازه‌های زمانی دیگر است و توزیع پواسن دارند. داده‌هایی که در مدل‌های زمان پیوسته مورد استفاده قرار می‌گیرند، زمان‌های واقعی خرابی‌های نرم‌افزار هستند و از این رو به صورت پیوسته ارائه می‌شوند. بسته به توزیع زمان‌های خرابی، این دسته از مدل‌ها به دو زیرشاخه تقسیم می‌گردند: مدل‌های «با بازه

زمانی بین خرابی مستقل» و مدل‌های «با رفتار خطای مستقل و مشابه».

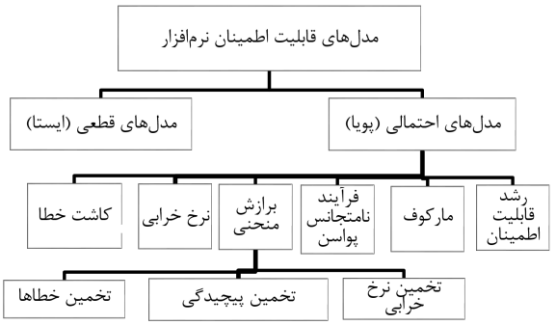
در مدل‌های «با بازه زمانی بین خرابی مستقل»، بازه‌های زمانی خرابی‌ها، به صورت غیروابسته توزیع شده‌اند. این مدل‌ها، توابع توزیع مشابه با پارامترهای مختلف دارند.مدل‌های «با رفتار خطای مستقل و مشابه»، بازه‌های زمانی خرابی‌ها برای هر مدل، مشابه و غیر وابسته‌ی احتمالی فرض می‌گردند [۱۷].

۳-۴- دسته‌بندی بر مبنای میزان قطعیت

همان‌طور که در شکل (۵) مشاهده می‌شود، مدل‌های قابلیت اطمینان نرم‌افزار را می‌توان به دو دسته تقسیم کرد: مدل‌های قطعی و مدل‌های احتمالی [۷ و ۲۷]. مدل‌های قطعی (ایستا) علاوه بر داده‌های خرابی، از برخی صفات برنامه یا پروژه برای تخمین تعداد خطاهای باقیمانده در نرم‌افزار استفاده می‌کنند [۷].

در مدل‌های احتمالی (پویا) داده‌های مربوط به مدل‌های احتمالی برخلاف مدل‌های قطعی، از یک محصول دلخواه جمع‌آوری می‌شوند. بنابراین نتیجه‌ی این دسته از مدل‌ها تنها به آن محصول خاص مرتبط است. علاوه‌براین این دسته از مدل‌های قابلیت اطمینان، داده‌های خرابی تولید شده توسط سیستم نرم‌افزاری را دنبال می‌کنند تا بتوانند پروفایل عملیاتی سیستم را در بازه‌ی زمانی خاصی تشکیل دهند. از آن‌جایی که این مدل‌ها بر مبنای توزیع احتمالی می‌باشند، مدل‌های پویا نیز نامیده می‌شوند. ویژگی معمول مدل‌های پویا این است که به صورت تابعی از زمان معرفی می‌گردند. مدل‌های احتمالی را می‌توان به شش دسته مدل تقسیم کرد: مدل‌های رشد قابلیت اطمینان، نرخ خرابی، برازش منحنی، کاشت خطا، فرآیند نامتجانس پواسن و مارکوف. هر کدام از این دسته مدل‌ها دارای روشی برای تخمین قابلیت اطمینان نرم‌افزار و سایر اندازه‌های کارآیی مانند پیچیدگی نرم‌افزار، امنیت نرم‌افزار و تعداد خطاهای باقی‌مانده هستند.

مدل‌های برازش منحنی از آنالیز رگرسیون آماری برای بررسی رابطه‌ی بین پیچیدگی برنامه و تعداد شکست‌ها و نرخ خرابی استفاده می‌کنند. این مدل‌ها همچنین از رگرسیون خطی، غیر خطی یا تحلیل‌های سری زمانی برای پیدا کردن رابطه‌ی بین متغیرهای وابسته و مستقل استفاده می‌نمایند. برای مثال تعداد خطاهای نرم‌افزار می‌تواند یک متغیر وابسته باشد. از طرف دیگر زمان بین شکست‌ها، توانایی برنامه‌نویسان و اندازه‌ی برنامه می‌توانند متغیرهای مستقل باشند. این مدل‌ها به سه دسته‌ی مدل‌های تخمین خطا، تخمین پیچیدگی و تخمین نرخ خرابی تقسیم می‌شوند. مدل‌های تخمین خطا با استفاده از مدل‌های رگرسیون، تعداد خطاهای برنامه را تخمین می‌زنند. مدل‌های تخمین پیچیدگی به عنوان مدلی برای تخمین پیچیدگی نرم‌افزار استفاده می-شوند. این دسته از مدل‌ها از روش سری زمانی برای تخمین پیچیدگی نرم‌افزار استفاده می‌کنند. از این مدل‌ها می‌توان برای نرم‌افزارهایی که نسخه‌های مختلفی را در بازه‌های زمانی مختلف منتشر می‌کنند، استفاده کرد. مدل‌های تخمین نرخ خرابی تلاش می‌کنند که نرخ خرابی را با استفاده از زمان‌های خرابی گذشته، تخمین بزنند. این مدل‌ها فرض می‌کنند که نرخ خرابی به کمک روش حداقل مربعات[۱۶] قابل دستیابی است و علاوه بر آن یکنواخت و غیرافزایشی می‌باشد [۷].

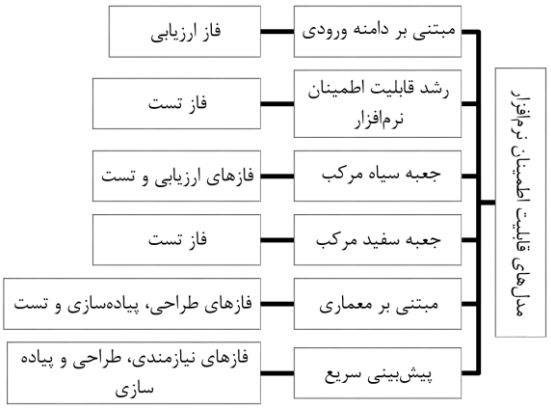


شکل ۵: دسته‌بندی بر مبنای قطعیت

۳-۵- دســـته‌بندی بر مبنای دوره‌ی چرخه‌ی حیات توسعه‌ی نرم‌افزار

مقاله‌ی [۱۹]، مدل‌های قابلیت اطمینان نرم‌افزار را بر مبنای چرخه‌ی حیات توسعه نرم‌افزار به زیردسته‌های مدل‌های پیش‌بینی سریع، مدل‌های رشد قابلیت اطمینان نرم‌افزار، مدل‌های مبتنی بر دامنه ورودی، مدل‌های مبتنی

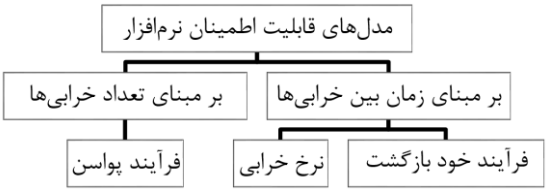
بر معماری، مدل‌های جعبه سیاه مرکب و مدل‌های جعبه سفید مرکب دسته‌بندی می‌کند. شکل (۶) بیان می‌کند که هر کدام از این زیردسته‌ها، در کدامیک از فازهای توسعه نرم‌افزار قابل استفاده هستند.



شکل ۶: دسته‌بندی بر مبنای دوره چرخه حیات توسعه نرم‌افزار

۳-۶- دسته‌بندی بر مبنای خرابی‌های نرم‌افزار

همان‌طور که در شکل (۷) مشاهده می‌شود، بر مبنای خرابی‌های نرم‌افزار، مدل‌های قابلیت اطمینان نرم‌افزار را می‌توان به دو دسته تقسیم کرد [۷].

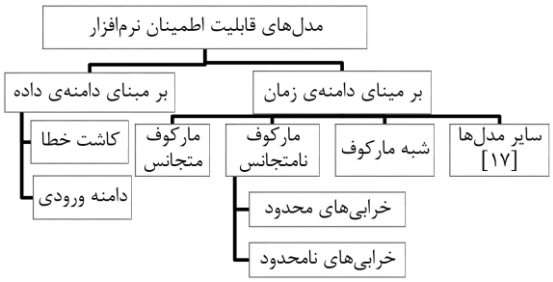


شکل ۷: دسته‌بندی بر مبنای خرابی‌های نرم‌افزار

اولین دسته، بر مبنای زمان بین خرابی‌های نرم‌افزار است که به دو زیر دسته تقسیم می‌شود: آن دسته از مدل‌ها که از نرخ خرابی به عنوان ابزار مدل‌سازی استفاده می‌کنند (مدل‌های نرخ خرابی)، و آن‌هایی که زمان بین خرابی‌ها را به عنوان تابعی از زمان‌های بین خرابی گذشته مدل می‌کنند (فرآیند خود بازگشت[۱۷]). دسته‌ی دوم، بر مبنای تعداد خرابی‌های نرم‌افزار تا زمانی مشخص، می‌باشد. در این دسته اغلب از فرآیند پواسن برای پیش‌بینی قابلیت اطمینان نرم‌افزار استفاده می‌شود [۷].

۳-۷- دســـته‌بندی بر مبنای دامنه‌ی داده و دامنه‌ی زمان

شکل (۸) مدل‌های قابلیت اطمینان نرم‌افزار را به دو دسته (بر مبنای دامنه داده و بر مبنای دامنه زمان) تقسیم می‌کند [۲۰]. هر یک از دسته مدل‌های ارائه شده در این شکل، در بخش‌های دیگر توضیح داده شده‌اند.



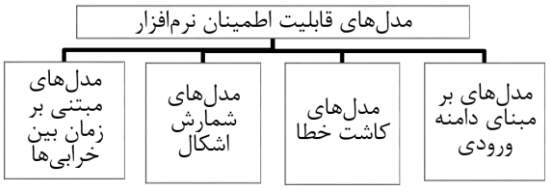
شکل ۸: دسته‌بندی بر مبنای دامنه‌ی ورودی و دامنه‌ی زمان

۳-۸- دسته‌بندی بر مبنای تاریخچه‌ی شکست[۱۸]

همان‌طور که در شکل (۹) مشاهده می‌شود، بر اساس تاریخچه‌ی شکست، می‌توان مدل‌های قابلیت اطمینان نرم‌افزار را به چهار دسته تقسیم کرد [۲۱].

در مدل‌های «مبتنی بر زمان بین خرابی‌ها»[۱۹]، فرآیندی که مورد بررسی قرار می‌گیرد، زمان بین شکست‌ها است. فرض می‌شود که زمان بین $(i-1)$ و i امین شکست، مقداری تصادفی می‌باشد و در ضمن، این زمان از توزیعی پیروی می‌کند که پارامترهای آن به شکست‌های باقی‌مانده درطی این بازه بستگی دارد. تخمین این پارامترها به کمک مدل‌های «مبتنی بر زمان بین خرابی‌ها» انجام می‌گیرد.

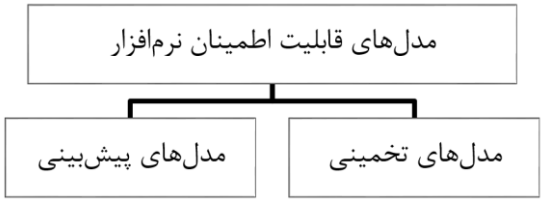
در مدل‌های «شمارش اشکال»[۲۰]، تعداد خطاهایی که در یک بازه‌ی زمانی معلوم رخ می‌دهد، نامعلوم و تصادفی می‌باشد. در این مدل‌ها فرض می‌شود که تعداد خطاها از یک فرآیند نامعلوم تبعیت می‌کند. زمان می‌تواند زمان تقویمی و یا زمان پردازنده[۲۱] باشد. پارامترهای نرخ شکست نیز بر اساس تعداد شکست‌های مشاهده شده، تخمین زده می‌شوند. دو دسته مدل دیگر در بخش‌های دیگر معرفی شده‌اند.



شکل ۹: دسته‌بندی بر مبنای تاریخچه‌ی شکست

۳-۹- دسته‌بندی بر مبنای زمان اندازه‌گیری

همان‌طور که در شکل (۱۰) مشاهده می‌شود، از منظر زمان اندازه‌گیری قابلیت اطمینان، مدل‌های قابلیت اطمینان نرم‌افزار را می‌توان به دو دسته تقسیم کرد [۲۲، ۲۳].



شکل ۱۰: دسته‌بندی بر مبنای زمان اندازه‌گیری

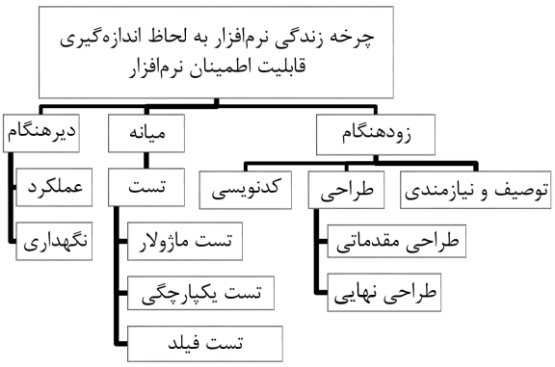
تخمین قابلیت اطمینان به معنی این است که قابلیت اطمینان فعلی را با اعمال تکنیک‌های استنتاج آماری بر روی داده‌های دارای اشکال در طی مراحل تست یا عملکرد سیستم، بدست آوریم. این نوع اندازه‌گیری، قابلیت اطمینان را از ابتدا تا نقطه‌ی فعلی مد نظر قرار می‌دهد. از تخمین برای ارزیابی قابلیت اطمینان فعلی استفاده می‌شود تا بتوان تعیین کرد که آیا این مدل قابلیت اطمینان در این برهه از عملیات موفق بوده است یا خیر.

مدل‌های پیش‌بینی، قابلیت اطمینان را با توجه به متریک‌ها و اندازه‌گیری‌های نرم‌افزار موجود، پیش‌بینی می‌نماید. در طی مراحل مختلف توسعه‌ی نرم‌افزار، پیش‌بینی از تکنیک‌های مختلفی استفاده می‌نماید. بیشتر مدل‌های قابلیت اطمینان نرم‌افزاری در دسته‌ی تخمین قرار می‌گیرند تا پیش‌بینی. اما هنوز مدل‌هایی برای پیش‌بینی این نیازمندی پیشنهاد می‌گردند [۲۲].

۳-۱۰- دسته‌بندی بر مبنای چرخه‌ی زندگی

مقاله‌ی [۲۴]، چرخه‌ی زندگی نرم‌افزار را به لحاظ اندازه‌گیری قابلیت اطمینان نرم‌افزار به زیردسته‌های معرفی شده در شکل (۱۱) تقسیم‌بندی می‌کند.

در فازهای اولیه (زودهنگام) چرخه‌ی حیات نرم‌افزار، از مدل‌های پیش‌بینی قابلیت اطمینان نرم‌افزار استفاده می‌شود. زیرا در این فازها، داده‌های شکست موجود نیستند. این دسته از مدل‌ها، تعداد خطاهای برنامه را قبل از تست نرم‌افزار، پیش‌بینی می‌کنند.



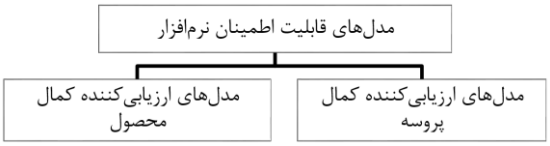
شکل ۱۱: دسته‌بندی بر مبنای چرخه‌ی زندگی

در فاز تست (میانه)، در جریان اشکال‌زدایی برنامه، قابلیت اطمینان نرم‌افزار افزایش پیدا می‌کند. در این مرحله مدل‌های رشد قابلیت اطمینان نرم‌افزار، قابلیت اطمینان فعلی نرم‌افزار، زمان و منابع مورد نیاز برای رسیدن به قابلیت اطمینان مورد نظر را تخمین می‌زنند. در طول این فاز، تخمین قابلیت اطمینان نرم‌افزار بر مبنای داده‌های شکست، صورت می‌گیرد.

پس از انتشار نرم‌افزار (دیرهنگام)، اضافه کردن ماژول جدید، حذف ماژول‌های قدیمی، حذف خطاهای مشخص شده، ترکیب کدهای جدید با کدهای نوشته شده در گذشته، تغییر سخت‌افزار و موارد اینچنینی باید در ارزیابی قابلیت اطمینان نرم‌افزار مد نظر قرار گیرند. در طول این فاز، سیر تکاملی مدل مورد نیاز می‌باشد.

۳-۱۱- دسته‌بندی بر مبنای کارکرد

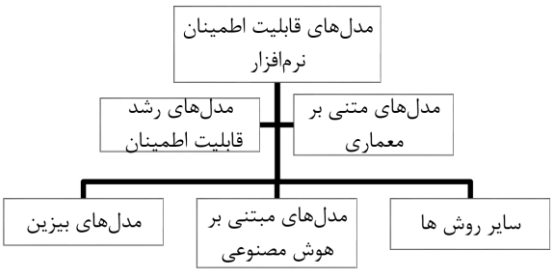
شکل (۱۲) اندازه‌گیری‌ها را بر مبنای دو هدف مهم اندازه‌گیری قابلیت اطمینان، دسته‌بندی می‌کند: ارزیابی کمال[۲۲] محصول و ارزیابی کمال فرآیند. اولین هدف، گواهی آمادگی محصول برای عملیات است. دومین هدف، ارزیابی مداوم قابلیت تولید محصولاتی با کیفیت مطلوب برای نیازهای مشتری، توسط سازنده‌ی نرم‌افزار می‌باشد. درواقع ارزیابی کمال فرآیند شامل ارزیابی کمال محصول است به همراه اصلاح فرآیند هنگامی که لازم باشد [۲۲].



شکل ۱۲: دسته‌بندی بر مبنای کارکرد

۳-۱۲- دسته‌بندی ارائه شده در مرجع [۶]

مرجع [۶] یکی از جدیدترین مطالعات انجام شده (در سال۲۰۱۴) بر اساس یک چارچوب سیستماتیک در مقوله¬ی مدلسازی قابلیت اطمینان نرم‌افزار است. در این مقاله اطلاعات جالبی در زمینه‌ی مدل‌سازی قابلیت اطمینان و همچنین مقالات و پژوهش‌های موجود در این زمینه گردآوری شده است. این مقاله پس از بررسی مدل‌ها و دسته‌بندی‌های موجود، به معرفی یک دسته‌بندی جدید می‌پردازد. در شکل (۱۳) این دسته‌بندی آورده شده است.



شکل ۱۳: دسته‌بندی مرجع [۶]

در مورد مدل‌های رشد قابلیت اطمینان نرم‌افزار، مدل‌های مبتنی بر تست و نیز مدل‌های مبتنی بر معماری در بخش (۳-۱۳) توضیحات لازم داده شده‌است.

روش‌های بیزین بر اساس آمار موجود در مرجع [۶]، بعد از مدل‌های رشد قابلیت اطمینان، ۶٪ از کارها و پژوهش‌های موجود در زمینه‌ی مدلسازی قابلیت اطمینان را به خود اختصاص داده‌اند. برای مطالعه‌ی بیشتر در این زمینه شما را به مرجع [۲۸] ارجاع می‌دهیم.

بر اساس مرجع [۶]، مدل‌هایی که در آن‌ها از تکنیک‌های هوش مصنوعی مانند یادگیری ماشین، یادگیری ایستا، الگوریتم‌های حریصانه، الگوریتم ژنتیک، منطق فازی و مفاهیم اینچنینی استفاده می‌شود، در دسته‌بندی مدل‌های مبتنی بر هوش مصنوعی قرار می‌گیرند. البته معمولاً مدل-های موجود در این دسته از مفهوم و تکنیک شبکه‌های عصبی استفاده می‌کنند.

در قسمت (۳-۱۳) در مورد مدل‌های مبتنی بر معماری نرم‌افزار توضیح داده خواهد شد. بر اساس مرجع [۱۰]، مدل‌های مبتنی بر ویژگی‌های ایستا و معماری نرم‌افزار بر مبنای ملاحظات معماری و ویژگی‌های ایستای نرم‌افزار شکل گرفته‌اند. در واقع این مدل‌ها بین معیارهای مهندسی نرم‌افزار و قابلیت اطمینان نرم‌افزار رابطه بر قرار می‌کنند. این دسته از مدل‌ها در فازهای اولیه‌ی چرخه‌ی حیات نرم-افزار به کار گرفته می‌شوند. از این رو همانطور که قبلا نیز در مورد مدل‌های پیشبینی قابلیت اطمینان نرم‌افزار گفته شد، باعث صرفه جویی در هزینه می‌شوند.

در دسته‌بندی ارائه شده در مرجع [۶] منظور از سایر روش-ها، مدل‌هایی است که از تکنیک‌های اقتضایی[۲۳] استفاده می‌کنند که این تکنیک‌ها بر مبنای تئوری‌های خاص هستند (مانند بلاک دیاگرام‌های قابلیت اطمینان، تحلیل ریسک، تحلیل مد شکست، تئوری قطعی هرج و مرج و تئوری مدل ابری). یا روش‌هایی که برای موردهای خاصی شکل گرفته‌اند (مثل مدل اقتضایی برای یک سیستم خاص مبتنی بر برنامه نویسی چند نسخه‌ای) یا روش‌های مبتنی بر متریک‌های مهندسی و بنچمارک گرفتن یا تخمین قابلیت اطمینان در فازهای اولیه که بر اساس نیازمندی‌های رفتاری سیستم است.

۳-۱۳- دسته‌بندی پیشنهادی

در این بخش با ترکیب دسته‌بندی‌های ردیف‌های ۳، ۶، ۸ و ۱۰ و ۱۳ از جدول ۱ و اعمال یکسری اصلاحات و تغییرات، طبقه‌بندی جامع‌تر و دقیق‌تری ارائه خواهیم کرد. دسته‌بندی پیشنهادی در شکل (۱۴) نشان داده شده است. این دسته‌بندی کمک می‌کند تا مدل مورد نظر راحت‌تر و هوشمندانه‌تر انتخاب گردد. در ادامه به معرفی هر یک از زیر دسته‌های این دسته‌بندی می‌پردازیم.

دسته‌بندی پیشنهادی به این موضوع می‌پردازد که «مدل‌های قابلیت اطمینان نرم‌افزار در چه زمانی اعمال می‌گردند». چرخه‌ی توسعه‌ی نرم‌افزار را می‌توان به فازهای کلی اولیه، میانی و نهایی تقسیم نمود [۲۲]. فازهای مذکور را در حالت جزئی‌تر می‌توان به زیرفازهای نیازمندی، طراحی، پیاده‌سازی، آزمون، ارزیابی، عملیاتی، نگهداری و توسعه تقسیم کرد. در ادامه به معرفی هر یک از

زیردسته‌های شکل (۱۴) می‌پردازیم.

الف) مدل‌های پیش‌بینی سریع: مدل‌های قابلیت اطمینان نرم‌افزار می‌توانند در فازهای نهایی توسعه‌ی نرم‌افزار پیش‌بینی‌های دقیقی داشته باشند، اما انجام این پیش‌بینی‌ها در فازهای اولیه‌ی توسعه‌ی نرم‌افزار می‌توانند از نظر برآورد هزینه مفید باشند. یکی از دسته مدل‌هایی که می‌توانند در این فاز مورد استفاده قرار گیرند، مدل‌های پیش‌بینی سریع هستند [۲۹]. از آن‌جایی که در فازهای اولیه‌ی توسعه‌ی نرم‌افزار اطلاعات خرابی در دسترس نیستند، برخی از مدل‌های پیش‌بینی سریع از معیارهای طراحی برای پیش‌بینی قابلیت اطمینان استفاده می‌کنند [۳۰]. مدل‌های ارائه شده در [۳۱ و ۳۲] و از جمله مدل‌های پیش‌بینی سریع هستند.

ب) مدل‌های مبتنی بر معماری نرم‌افزار: این دسته از مدل‌ها، برای پیش‌بینی قابلیت اطمینان نرم‌افزار، ساختار درونی نرم‌افزار را در نظر می‌گیرند. بر اساس مرجع [۳۳]، مدل‌های مبتنی بر معماری در حالت کلی به دو دسته‌ی اصلی مدل‌های کمّی و مدل‌های کیفی تقسیم می‌شوند. مدل‌های کمی خود به سه دسته‌ی مبتنی بر حالت، مبتنی بر مسیر و افزایشی تقسیم می‌شوند. در رویکرد مبتنی بر حالت (زیردسته‌ی شماره ۲۵)، معماری نرم‌افزار به صورت زنجیر مارکوف زمان گسسته/پیوسته، یا فرآیند نیمه‌مارکوف مدل شده و قابلیت اطمینان نرم‌افزار به صورت شبیه‌سازی (سلسله مراتبی) و یا تحلیلی (توسط ترکیب معماری با رفتار شکست) تخمین زده می‌شود (زیردسته‌های شماره ۳۰ و ۳۱) [۳۴]. از این دسته مدل‌ها می‌توان به مدل‌های ارائه شده در [۳۵، ۳۶ و ۳۷] اشاره کرد. مدل‌های تحلیلی (ترکیبی) خصیصه‌های معماری را با رفتار شکست ترکیب کرده و یک مدل واحد ایجاد می‌کنند. برای اطلاعات بیشتر در مورد این دسته از مدل‌های مبتنی بر حالت، به منابع [۳۵، ۳۸ و ۳۹]. مراجعه نمایید. مدل‌های سلسله مراتبی فرآیند تحلیل قابلیت اطمینان را به دو مرحله‌ی مجزا تقسیم می‌کنند (برای مطالعه‌ی بیشتر به منابع [۳۷، ۴۰ و ۴۱] مراجعه شود).

در رویکرد مبتنی بر مسیر (زیردسته‌ی شماره ۲۶)، برای محاسبه‌ی قابلیت اطمینان نرم‌افزار، مسیرهای ممکن اجرا، به کمک شبیه‌سازی، اجرای برنامه و یا به صورت الگوریتمی (زیردسته‌های شماره ۳۲، ۳۳ و ۳۴)، در نظر گرفته می‌شود. قابلیت اطمینان هر مسیر از طریق ضرب قابلیت‌های اطمینان اجزاء آن مسیر بدست می‌آید. در نهایت قابلیت اطمینان برنامه کاربردی برابر میانگین قابلیت‌های اطمینان مسیرهای مختلف خواهد بود [۳۴]. مدل‌های ارائه شده در منابع [۳۶، ۴۲، ۴۳ و ۴۴]، از جمله مدل‌های مبتنی بر مسیر هستند. در رویکرد افزوده (زیردسته‌ی شماره ۲۷)، معماری به صورت صریح در نظر گرفته نمی‌شود بلکه تمرکز این روش بر تخمین چگالی شکست وابسته به زمان برنامه کاربردی، با استفاده از داده‌های شکست مؤلفه‌ها می‌باشد.

دلیل اینکه به این دسته از مدل‌ها، افزوده گفته می‌شود این است که چگالی خرابی سیستم (از آنجایی‌که قابلیت اطمینان اجزاء به کمک فرآیند نامتجانس پواسن مدل می‌شوند) به صورت حاصل‌جمع چگالی‌های شکست اجزاء بیان می‌گردد [۴۵]. از این دسته مدل‌ها می‌توان به مدل‌های ارائه شده در [۴۶ و ۴۷] و اشاره کرد.

بر اساس مرجع [۳۳]، مدل‌های کیفی (زیر دسته‌ی شماره ۱۳) بر خلاف مدل‌های کمّی که با اعداد و ارقام سر و کار دارند، از طریق دانش به تحلیل قابلیت اطمینان می‌پردازند. این دانش معمولاً خاصِ سیستم تحت بررسی می‌باشد و می‌تواند عینی و بر گرفته از مستندات باشد و یا ضمنی و بر گرفته از فکر و نظرات کارشناسان باشد (که در این صورت به دلیل وابستگی بالا به انسان، مستعد خطاست). همچنین این دانش می‌تواند انتزاعی و کلی باشد یا مربوط به دامنه و کاربرد خاصی باشد. البته اخیراً تمایل به سمت مستند کردن دانش‌های کلی است؛ مثلاً از طریق شناسایی و استفاده از الگوهای طراحی. البته در مورد سبک‌ها و الگوهایی که تمرکز بر روی ویژگی قابلیت اطمینان دارند، با کمبود قابل ملاحظه‌ای مواجه هستیم. در حال حاضر فقط یک سبک معماری وجود دارد که بر روی قابلیت اطمینان

تمرکز می‌کند [۴۸].

انواع مختلفی از روش‌های مبتنی بر دانش وجود دارد؛ تحلیل‌های مبتنی بر درخت اشکال و مد شکست و روش-های ابتکاری. روش‌های ابتکاری خود به دو دسته تقسیم می‌شوند. دسته‌ی اول روش‌هایی هستند که از تجربیات مهندسین نرم‌افزار، در راستای توسعه‌ی نرم‌افزار با درجه‌ی بالایی از کیفیت حاصل می‌شود. به عنوان مثال چارچوب نیازمندی‌های غیرکارکردی معرفی شده در مرجع [۴۹] و مفاهیم اساسی طراحی، که به طور بالقوه در رابطه با بکارگیری الگوهای طراحی می‌باشند و می‌توانند در راستای نگاشت نیازمندی‌ها به تصمیمات طراحی و در نتیجه مستندسازی و تعدیل این تصمیمات استفاده شوند. روش-های موجود در این زمینه، سبک‌ها و روش‌های طراحی هستند و باید با اضافه کردن یک بخش تحلیل استادانه‌تر و شفاف‌تر، ارتقا داده شوند. دسته‌ی دیگر از روش‌های مبتکرانه، شامل روش‌هایی است که به طور خاص بر روی تحلیل معماری نرم‌افزار تمرکز می‌کنند. مانند روش‌های تحلیل معماری مبتنی بر سناریو [۵۰] ، روش‌های تحلیل معماری مبتنی بر سناریو که بر روی سناریوهای پیچیده پایه‌ریزی شده‌اند [۵۱]، روش‌های تحلیل معماری مبتنی بر سناریو تعمیم‌یافته به وسیله‌ی ادغام دامنه [۵۲] و روش تحلیل مبادله‌ی معماری [۵۳].

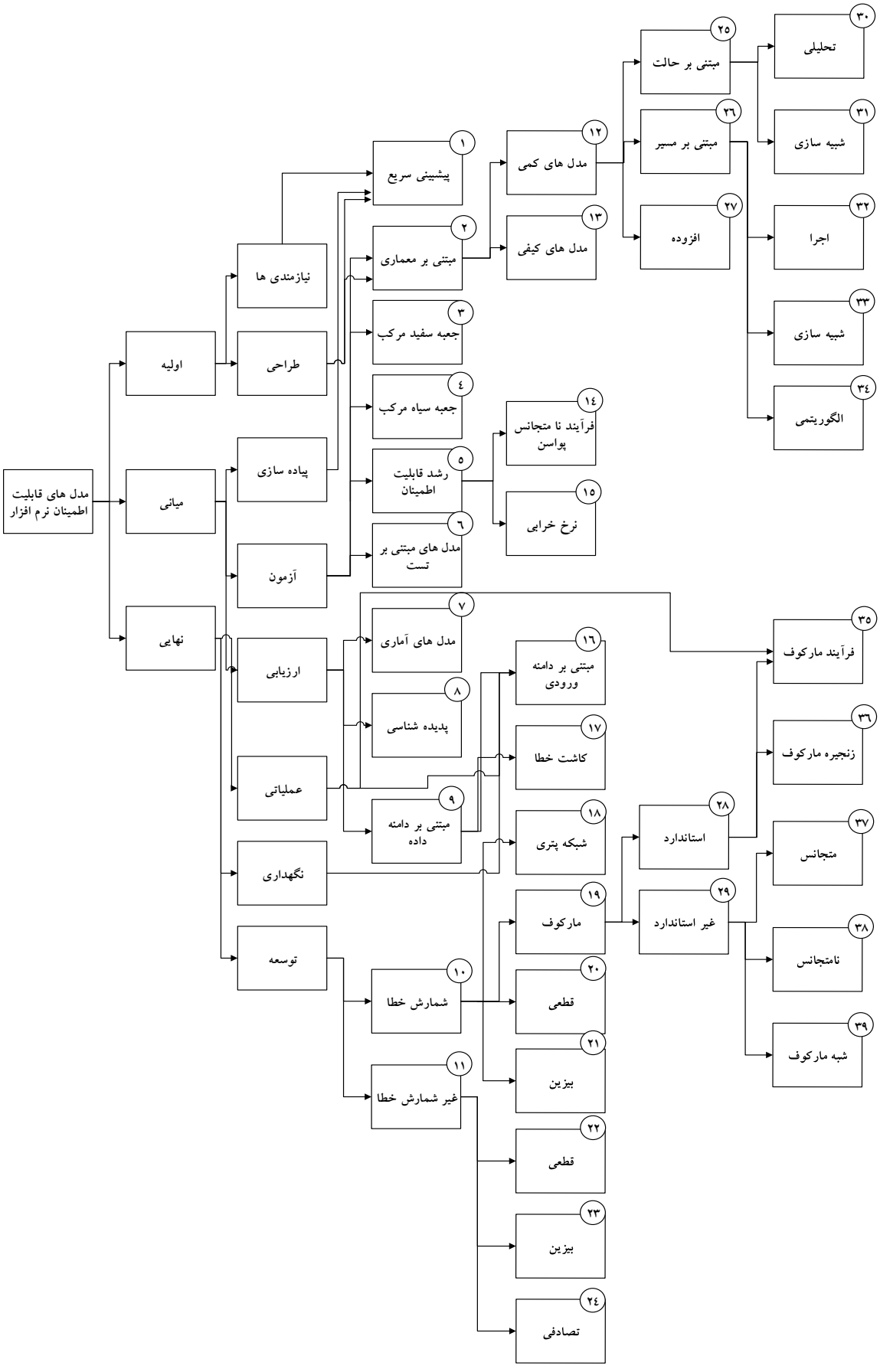
ج) مدل‌های جعبه سفید مرکب: این دسته از مدل‌ها از برخی ویژگی‌های مدل‌های جعبه سفید و جعبه سیاه به صورت ترکیبی استفاده می‌کنند. از آن‌جایی که این دسته از مدل‌ها برای پیش‌بینی قابلیت اطمینان، معماری نرم‌افزار را لحاظ می‌کنند، این مدل‌ها را با عنوان مدل‌های جعبه سفید مرکب مطرح می‌کنند [۱۹]. در [۵۴] نمونه‌ای از این دسته از مدل‌ها معرفی شده است.

د) مدل‌های جعبه سیاه مرکب: این مدل‌ها، ویژگی‌های مدل‌های مبتنی بر دامنه ورودی و مدل‌های رشد قابلیت اطمینان را با یکدیگر ترکیب می‌نمایند [۱۹]. در این زمینه (و همچنین در زمینه مدل‌های جعبه سفید مرکب) مطالعات محدودی انجام شده است. نمونه‌ای از این مدل‌ها در [۵۵] معرفی شده‌است.

ه) مدل‌های رشد قابلیت اطمینان نرم‌افزار: این دسته از مدل‌ها نشان می‌دهند که چگونه قابلیت اطمینان نرم‌افزار در طی فرآیند آزمون به مرور زمان افزایش پیدا می‌کند. هنگامی که خرابی‌های نرم‌افزار کشف شوند، شکست‌هایی که منجر به بروز این خرابی‌ها شده‌اند شناسایی و تعمیر می‌گردند و در نتیجه قابلیت اطمینان در طی آزمون و اشکال‌زدایی نرم‌افزار بهبود پیدا می‌کند. برای پیش‌بینی قابلیت اطمینان، مدل مفهومی ذکر شده، باید به مدلی ریاضی تبدیل گردد. مدل‌های رشد قابلیت اطمینان به دو دسته‌ی نرخ خرابی و فرآیند نامتجانس پواسن تقسیم می‌شوند.

مدل‌های دسته‌ی فرآیند نامتجانس پواسن (زیردسته‌ی شماره ۱۴) بر مبنای مدل آماری فرآیند پواسن غیرمتجانس می‌باشند (فرآیند پواسن غیر متجانس یک فرآیند پواسن با نرخ $\lambda(t)$ است که وابسته به زمان می‌باشد). مدل‌های ارائه شده در [۹، ۵۶، ۵۷ و ۵۸]، از جمله مدل‌های دسته‌ی فرآیند نامتجانس پواسن هستند. مدل‌های دسته‌ی نرخ خرابی (زیردسته‌ی شماره ۱۵) بررسی می‌کنند که نرخ خرابی چگونه در زمان خرابی و در طول بازه‌های خرابی تغییر می‌کند. اگر تعداد شکست‌های باقیمانده در نرم‌افزار تغییر کنند، نرخ خرابی برنامه نیز تغییر خواهد کرد [۷]. در [۵۹ و ۶۰] و نمونه‌ای از این دسته مدل‌ها ارائه شده‌اند.

بر اساس منابع [۶، ۵۶،۲۷ و ۶۱] باید این نکته را در نظر گرفت که این دسته از مدل‌ها به نرم‌افزار به صورت کلی و به اصطلاح جعبه سیاه نگاه می‌کنند و ساختار داخلی و معماری اجزاء نرم‌افزار را در نظر نمی‌گیرند. همچنین بر اساس مرجع [۲۷] نمودار تعداد اشکالات کشف شده بر حسب زمان تست برای این مدل‌ها می‌تواند مقعری شکل و یا به شکل حرف S باشد.



شکل ۱۴: دسته‌بندی پیشنهادی

بر اساس آمارهایی که در مورد کارهای موجود در مدلسازی قابلیت اطمینان نرم‌افزار در مرجع [۶] گردآوری شده است، مدل‌های رشد قابلیت اطمینان نرم‌افزار از سایر مدل‌های دیگر پر استنادتر هستند و اکثر مقالات این حوزه را به خود اختصاص داده‌اند. برای مثال از بین مدل‌هایی که در این دسته‌بندی ارائه شده، ۴۷٪ از مقالات و کارهای انجام شده به این مدل‌ها اختصاص دارد. لذا این دسته از مدل‌ها از اهمیت ویژه‌ای برخوردار هستند و برای مطالعه‌ی بیشتر در این زمینه شما را به منابع [۵۶ و ۶۱] ارجاع می‌دهیم.

و) مدل‌های مبتنی بر تست: این مدل‌ها تنها فاکتورهایی نظیر تعداد شکست‌های اتفاق افتاده در یک بازه‌ی زمانی مشخص و یا متوسط زمان بین شکست‌ها را در نظر نمی‌گیرند؛ بلکه فاکتور دیگری نظیر میزان پوشش تست را نیز به فاکتورهای گفته شده اضافه می‌کنند [۶۲]. در واقع مدل‌های مبتنی بر تست چگونگی استفاده از سیستم توسط کاربر را نیز در نظر می‌گیرند و موردهای تست را بر اساس پروفایل عملیاتی سیستم- که منعکس کننده‌ی فضای ورودی نرم‌افزار است- انتخاب می‌کنند.

ز) مدل‌های آماری: این مدل‌ها با استفاده از داده‌های تجربی شکست نرم‌افزار که در یک فرآیند آزمایشی قابل جمع‌آوری و ثبت می‌باشد، قابلیت اطمینان نرم‌افزار را با بکارگیری روش‌های ریاضی و توزیع‌های آماری مدل‌سازی می‌کنند [۶۳]. مدل‌های ارائه شده در [۶۴ - ۶۶] از جمله مدل‌های آماری هستند.

ح) مدل‌های پدیده‌شناسی: این دسته از مدل‌ها بر اساس تعداد عملیات و عملوندها، تخمینی از میزان خطای موجود در برنامه ارائه می‌کنند. اگر میزان خطا زیاد باشد، به موارد تست بیش‌تری برای شناسایی این خطاها نیاز است [۱۶]. از این دسته از مدل‌ها می‌توان به مدل ارائه شده در [۶۷] اشاره کرد.

ط) مدل‌های مبتنی بر دامنه‌ی داده: این دسته از مدل‌ها، به این صورت عمل می‌نمایند که مجموعه‌ی داده‌های نمونه‌ای را که بیانگر استفاده‌ی عملیاتی (با هدف تخمین شکست‌های باقی‌مانده در محصول نرم‌افزاری) می‌باشند، به برنامه‌ی کامپیوتری داده شده و آزمایش می‌گردند. خروجی برنامه بدست می‌آید و از این طریق قابلیت اطمینان نرم‌افزار تخمین زده می‌شود. این دسته از مدل‌ها را می‌توان به دو زیر دسته تقسیم نمود: مدل‌های مبتنی بر دامنه ورودی و مدل‌های کاشت خطا [۲۰].

در مدل‌های مبتنی بر دامنه ورودی (زیردسته‌ی شماره ۱۶)، تعدادی موردِ تست از روی ورودی تولید می‌شود که پروفایل عملکردی ورودی را پوشش داده و هر کدام مرتبط با یک مسیر برنامه هستند. قابلیت اطمینان یک برنامه بوسیله‌ی خرابی‌هایی که در طی اجرای هر کدام از این موارد تست بدست می‌آید، تخمین زده می‌شود [۲۱]. مدل‌های ارائه شده در [۶۸ و ۶۹] از جمله مدل‌های مبتنی بر دامنه ورودی هستند.

مدل‌های کاشت خطا (زیردسته‌ی شماره ۱۷) بر این فرض استوار هستند که اگر تعدادی خطای معلوم و کنترل‌شده را به عنوان هسته به نرم‌افزار اعمال کنیم، به کمک نسبت خطاهای کشف شده توسط روش‌های تست به خطاهای کشف نشده، می‌توان تعداد خطاهای باقی‌مانده در نرم‌افزار را تخمین زد. تخمین تعداد واقعی خطاهای ذاتی از طریق رابطه‌ی $N = \frac{nS}{s}$ بدست می‌آید. در این رابطه،S تعداد خطاهای انتخاب شده به عنوان هسته، sتعداد خطاهای هسته شناسایی شده، N تعداد خطاهای اصلیِ موجود و پنهان در کد برنامه (این مقدار تخمین زده می‌شود) و n تعداد خطاهای اصلی برنامه که کشف شده‌اند، می‌باشند [۷۰]. از این دسته از مدل‌ها می‌توان به مدل‌های ارائه شده در [۷۱- ۷۳] اشاره کرد.

ی) مدل‌های «شمارش خطا»: این مدل‌ها علاوه بر اندازه‌گیری قابلیت اطمینان فعلی نرم‌افزار، تعداد خطاهای باقی‌مانده در برنامه را نیز تخمین می‌زنند. اما مدل‌های «غیر شمارش خطا» تنها قابلیت اطمینان نرم‌افزار را اندازه‌گیری می‌کنند. مدل‌های شمارش خطا به چهار دسته‌ی شبکه پتری، مارکوف، قطعی و بیزین تقسیم می‌شوند. شبکه‌های پتری (زیردسته‌ی شماره ۱۸) در سال

۱۹۶۲ توسط کارل آدام پتری معرفی شدند. یک شبکه پتری از چهار عنصر پایه تشکیل شده است: مکان، گذار، کمان و نشانه. مکان‌ها نمایانگر وضعیت‌های ممکن سیستم هستند. هر مکان می‌توان شامل تعدادی نشانه باشد. نشانه‌ها بیان می‌کنند که سیستم یا بخشی از آن در زمان جاری در چه وضعیتی ممکن است قرار داشته باشند. گذارها فعالیت‌های سیستم را نشان می‌دهند. یک گذار، زمانی توانا می‌شود که تمام مکان‌های ورودی آن نشانه را داشته باشند. یک گذارِ توانا می‌تواند آتش کند، که در این صورت یک نشانه از هر مکان ورودی آن کم شده و یک نشانه به هر مکان خروجی آن اضافه می‌گردد. مقدار قابلیت اطمینان به صورت یک تجمیع از مضارب بر اساس احتمالات آتش محاسبه می‌شود [۷۴].

مارکوف (زیردسته‌ی شماره ۱۹) یک سیستم ریاضی است که شامل تعدادی حالت $S_1, \dots, S_n$ (محدود و قابل شمارش) است و در زمانهای مختلف، حالت سیستم با توجه به مجموعه‌ای از احتمالات تغییر می‌کند. مدل مارکوف ابزار قدرتمندی برای حل مسائل دارای فرآیند تصادفی (احتمالی) است. مدل‌های مارکوف را می‌توان به دو دسته‌ی مدل‌های استاندارد (زیردسته‌ی شماره ۲۸) و غیراستاندارد (زیردسته‌ی شماره ۲۹) تقسیم‌بندی کرد. مدل‌های استاندارد نیز به دو دسته‌ی زنجیره‌ی مارکوف و فرآیند مارکوف تقسیم می‌شوند. اگر فضای حالت در مدل مارکوف پیوسته باشد، به آن مدل‌ها فرآیند مارکوف (زیردسته‌ی شماره ۳۵)، و اگر گسسته باشد، زنجیره‌ی مارکوف (زیردسته‌ی شماره ۳۶) گفته می‌شود [۷۵]. مدل‌های مارکوف غیر استاندارد به سه دسته‌ی متجانس، نامتجانس و شبه‌مارکوف[۴۸] تقسیم می‌شوند. مدل‌های مارکوف متجانس (زیردسته‌ی شماره ۳۷) فرض می‌کنند که تعداد شکست‌های اولیه‌ی نرم‌افزارِ تحتِ بررسی، نامشخص اما ثابت است (تعداد شکست‌های سیستم در هر زمان از فضای حالت زنجیره‌ی متجانس مارکوف) و میزان شکست نرم‌افزار و یا نرخ انتقال در زنجیره‌ی مارکوف به تعداد شکست‌های باقیمانده در نرم‌افزار بستگی دارد. مدل‌های ارائه شده در [۲۱ و ۵۹]، مدل‌های معروفی هستند که در این دسته قرار می‌گیرند. در مدل‌های مارکوف نامتجانس (زیردسته‌ی شماره ۳۸) فرض می‌شود که تعداد شکست‌های ظاهر شده در محصول نرم‌افزاری به صورت تصادفی بوده و رفتاری مشابه فرآیند نامتجانس پواسن دارند [۲۰]. از این دسته مدل‌ها می‌توان به مدل‌های ارائه شده در [۲۱ و ۷۶] اشاره کرد. مدل‌های شبه‌مارکوف (زیردسته‌ی شماره ۳۹)، مدل‌های احتمالی هستند که در تحلیل سیستم‌های پیچیده‌ی پویا مناسب هستند [۷۷]. این دسته از مدل‌ها فرض می‌کنند که تعداد شکست‌های اولیه‌ی نرم‌افزار، نامشخص اما ثابت است. علاوه بر این، چگالی شکست نرم‌افزار و نرخ انتقال از یک حالت، نه تنها به تعداد شکست‌های باقیمانده در نرم‌افزار، بلکه به زمان سپری شده در آن حالت نیز بستگی دارد [۲۰]. در [۷۸] نمونه‌ای از این مدل‌ها ارائه شده است.

مدل‌های قطعی (زیردسته‌ی شماره ۲۰) فرض می‌کنند که اگر پارامترهای مدل شناخته شده باشند، آن‌گاه تصحیح یک خطا موجب افزایش مشخصی در قابلیت اطمینان می‌شود و نرخ شکست (λ) به میزان مشخصی کاهش پیدا می‌کند. اما چون می‌دانیم که خطاهای مختلف نرخ شکست‌های مختلف دارند، مدلِ بهتر آن است که مقدار نرخ شکست را مقداری تصادفی فرض کنیم. مدل‌های ارائه شده در [۵۹، ۷۹، ۸۰ و ۸۱] از جمله مدل‌های قطعی هستند. در مدل‌های بیزین (زیردسته‌ی شماره ۲۱)، بر خلاف مدل‌های قطعی و تصادفی، فرض می‌شود که بین هر دو تصحیح خطای متوالی (حتی اگر هیچ تغییری در نرم‌افزار صورت نگیرد)، نرخ شکست کاهش می‌یابد [۱۶].

ک) مدل‌های «غیر شمارش خطا»: این مدل‌ها، به سه دسته (زیردسته‌های شماره ۲۲، ۲۳ و ۲۴) تقسیم می‌شوند: مدل‌های قطعی (مدل معرفی شده در [۸۲])، تصادفی (مانند مدل ارائه شده در [۸۳]) و بیزین (مانند مدل‌های معرفی شده در [۸۳ و ۸۴]). مدل‌های تصادفی فرض می‌کنند که خطاهای مختلف تأثیرات مختلفی روی نرخ شکست برنامه دارند و تأثیر یک خطا موجب افزایش میزان

قابلیت اطمینان به میزان نامشخص و تصادفی می‌شود. علاوه بر این فرض می‌شود که نرخ شکست بین دو تصحیح خطای متوالی ثابت باقی می‌ماند [۱۶].

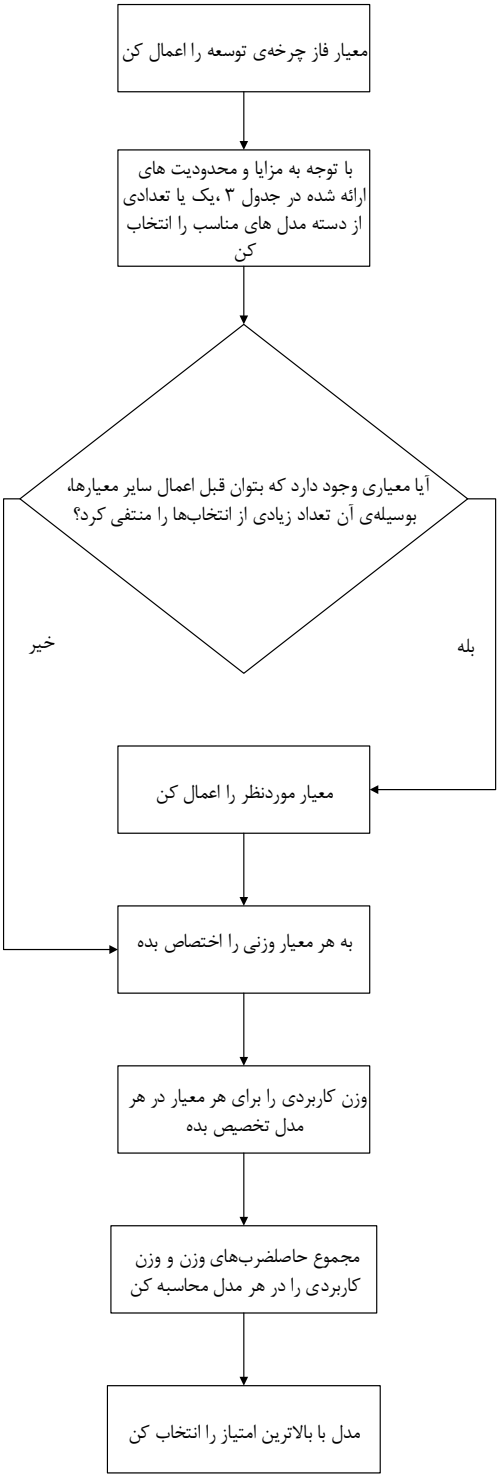
۴- الگوریتم انتخاب مناسب‌ترین مدل

در این بخش الگوریتمی برای انتخاب مناسب‌ترین مدل از بین زیر دسته‌های معرفی شده از دسته‌بندی پیشنهادی (که در بخش۳ معرفی گردید) ارائه شده است. این الگوریتم براساس الگوریتم انتخاب مدل در مرجع [۱۹]، تنظیم شده است. ما گام شماره ۲ را برای کارایی بیشتر و محدود کردن تعداد انتخاب‌های ممکن، به این الگوریتم اضافه کرده‌ایم. همچنین در جدول ۲ معیارهای انتخاب مناسب‌ترین مدل ارائه شده است. تعدادی از این معیارها در مرجع [۱۹] معرفی شده‌اند و تعدادی دیگر نیز پس از مطالعه بر روی مدل‌ها و روش‌های مختلف اندازه‌گیری قابلیت اطمینان نرم‌افزار، بدست آمده است. در شکل (۱۵) روندنمای مربوط به این الگوریتم را مشاهده می‌کنید. در ادامه به تشریح گام‌های الگوریتم مذکور می‌پردازیم. هر مدلی که در نهایت در این الگوریتم بیشترین امتیاز را بدست آورد، مدل منتخب خواهد بود.

گام ۱: در این گام معیار انتخاب فاز چرخه تولید نرم‌افزار اعمال می‌گردد. بعد از اعمال این معیار، تنها مدل‌هایی باقی می‌مانند که در آن فاز قابل استفاده باشند.

گام۲: در این گام سعی می‌شود با توجه به مزایا و محدودیت‌های مطرح شده در جدول ۳، یک یا چند دسته مدل انتخاب شود. بنابراین پس از این گام، تعداد زیادی از مدل‌ها حذف شده و مدل‌های تحت بررسی محدود می‌شوند.

گام ۳: در این مرحله به دنبال معیاری می‌گردیم که بتواند در فازی که در آن قرار داریم، قبل از بررسی سایر معیارها، تعداد زیادی از انتخاب‌های ممکن را حذف کند. مثلاً اگر ما در فاز طراحی قرار داریم، مدل‌های قابل استفاده در این فاز دو دسته‌ی مدل‌های پیشبینی سریع (مدل‌های جعبه سیاه) و مدل‌های مبتنی بر معماری (مدل‌های جعبه سفید) هستند. حال معیار تعیین کننده می‌تواند «ساختار پروژه» باشد؛ بر اساس این معیار، در صورتی که پروژه‌ی ما مبتنی بر جزء و مولفه باشد، استفاده از مدل‌های مبتنی بر معماری پیشنهاد می‌شود.



شکل ۱۵: روندنمای الگوریتم انتخاب مدل مناسب

جدول ۲: معیارهای انتخاب مناسب‌ترین مدل قابلیت اطمینان نرم‌افزار

معیار انتخاب مدل	توضیح
فاز چرخه‌ی توسعه‌ی نرم‌افزار	بر این اساس مدل بسته به این که در چه فازی قرار است مورد استفاده قرار بگیرد انتخاب می‌شود.
ورودی مورد نیاز توسط مدل	باید داده‌های مورد نیاز توسط مدل قابل دسترسی باشند. بعضی از موارد (فاز طراحی)، این معیار اهمیت پیدا می‌کند؛ زیرا تلاش بیشتری برای فراهم نمودن ورودی‌های مورد نیاز به وسیله مدل جعبه سفید نیاز است.
خروجی مطلوب توسط کاربر	تنها مدلی می‌تواند انتخاب شود که خروجی مطلوب را فراهم نماید.
روند ارائه شده توسط داده	اگر منحنی داده‌ای جمع‌آوری شده مربوط به شکست نرم‌افزار با منحنی مدل مشابه باشد، شانس بیشتری برای بدست آوردن نتیجه دقیق توسط این مدل وجود دارد.
درستی فرض‌ها بنا بر داده‌ها	تنها مدل‌هایی باید مورد بررسی قرار گیرند که درصد اعتبار فرض‌هایشان (طبق داده قابل دسترس) بالاتر باشد.
ماهیت پروژه	ماهیت یک پروژه شامل قابلیت اتمام نرم‌افزار (چه نرم‌افزار پایان یابد یا پایان نیابد)، اندازه‌ی آن و غیره می‌باشد. همچنین روش‌های معماری متفاوتی مانند قدرت تحمل نقص سیستم‌ها، فراخوانی و بازگشت و همچنین روش فیلتر موازی/لوله بررسی می‌شوند.
ساختار پروژه	اطلاعات ساختاری، چگونگی کنترل جابجایی‌های بین واحدها را نشان می‌دهد. این معیار برای برنامه‌های مبتنی بر مؤلفه بکار برده می‌شود.
فرآیند آزمون	مدل‌های قابلیت اطمینان نرم‌افزار مبتنی بر معماری، روش‌های آزمون استفاده شده در طول گام آزمون را در نظر می‌گیرند.
فرآیند توسعه	فرآیند توسعه شامل گزارش خطاها، رویکرد مورد استفاده و غیره می‌باشد.
هدف از اندازه‌گیری قابلیت اطمینان	آیا هدف ما از اندازه‌گیری قابلیت اطمینان مدیریت منابع تست هست یا می‌خواهیم برای تصمیم‌گیری در مورد اینکه چه زمانی می‌توانیم نرم‌افزار را انتشار دهیم از مدل اندازه‌گیری قابلیت اطمینان استفاده کنیم.
دقت در مقابل آسانی	دقت مدل بیشتر برایمان اهمیت دارد یا آسان بودن و قابل تفسیر بودن نتایج حاصل از آن. مدل‌هایی که از فاکتور زمان اجرا[۴۹] استفاده می‌کنند دقیق‌تر هستند؛ این در حالی است که مدل‌های مبتنی بر کل زمان سپری شده[۵۰]، به کارگیریشان آسان‌تر است.

گام ۴: بعد از گام ۳ با تعداد محدودی مدل مواجه می‌شویم. که برای انتخاب مناسب‌ترین مدل از بین آن‌ها، از معیارهای معرفی شده در جدول ۲ استفاده می‌کنیم. به این صورت که به هر کدام از آن معیارها، وزنی را که نمایانگر درجه‌ی اهمیت آن معیار است. نسبت وزن‌های نسبت داده شده، برای تمام مدل‌های درون یک دسته مساوی و متناسب با تعداد معیارهای قابل استفاده در آن دسته هستند. به عنوان مثال اگر از شش معیار برای یک دسته استفاده می‌شود، مقیاس وزنی از یک تا شش خواهد بود. در این حالت اگر وزن نسبت داده شده به دو معیار برابر باشد، مقیاس وزنی از ۱ تا ۵ می‌باشد. برای تبدیل مقیاس وزن‌ها، دو حالت ممکن وجود دارد:

الف) اگر تمام معیارهای ممکن برای یک دسته، برای تمامی مدل‌های موجود در آن دسته قابل استفاده باشد، برای تمامی مدل‌های درون آن دسته از وزن‌های مساوی استفاده خواهیم کرد.

ب) اگر برای یک یا چند مدل درون یک دسته، معیارهای قابل استفاده کمتر باشد، از فرمول تبدیل مقیاس برای آن مدل‌ها استفاده خواهیم کرد.

برای تبدیل مقیاس، ابتدا ضرب‌کننده را بدست خواهیم آورد. این ضرب کننده در وزن نسبت داده شده‌ی مدل‌های دارای معیار کمتر، ضرب خواهد شد. برای بدست آوردن ضرب‌کننده، تعداد معیارهای قابل استفاده در آن دسته را تقسیم بر تعداد معیارهای قابل استفاده در مدل مورد نظر

(مدلی با تعداد معیار کمتر) می‌کنیم. سپس مقدار بدست آمده را با تمامی مقادیر مربوط به مقیاس کمتر ضرب می‌کنیم.

گام ۵: پس از اینکه به هر کدام از معیارها یک وزن تخصیص داده شد، باید بررسی شود که هر کدام از این معیارها به چه میزان توسط هر مدل برآورده می‌شود. این میزان را به صورت یک وزن به هر یک از معیار مدل نسبت می‌دهیم. وزن تخصیص داده شده مقداری بین ۰ تا ۱ دارد. اگر یک معیار به طور کامل توسط مدل برآورده می‌شود، به آن مقدار یک و در غیر این صورت مقدار صفر را نسبت می‌دهیم. مقادیر میانی نیز (بین ۰ تا ۱) به نسبت میزان برآورده شدن آن نیاز تخصیص داده می‌شود.

گام ۶: در این گام دو وزن نسبت داده شده به هر معیار را در هم ضرب کرده و مقادیر بدست آمده را برای محاسبه‌ی امتیاز هر مدل با یکدیگر جمع می‌کنیم.

گام ۷: در انتها مدلی که بیش‌ترین امتیاز را بدست آورده است را به عنوان مدلی برای اندازه‌گیری قابلیت اطمینان نرم‌افزار استفاده می‌کنیم.

۵- تحلیل دسته‌بندی پیشنهادی

در این بخش برخی از مزایا و محدویت‌های زیر دسته‌های دسته‌بندی پیشنهادی (بر مبنای شماره‌گذاری‌های انجام شده در شکل (۱۴)) بیان می‌شوند. سایر مزایا و محدودیت‌های این دسته مدل‌ها به صورت خلاصه و دسته‌بندی شده در بخش ضمایم در قالب جدول ۳ ارائه می‌گردند. در نهایت بر مبنای تحلیل نتایج، به معرفی چالش‌ها و مسائل باز مرتبط پرداخته می‌شود.

مدل‌های پدیده‌شناسی (زیردسته شماره ۸): همانطور که در بخش ۳ بیان شد، مدل‌های پدیده‌شناسی از جمله دسته مدل‌های قابلیت اطمینان نرم‌افزار هستند. در صورتی که تعداد خطاها زیاد باشد، این مدل‌ها نیازمند موارد تست بیش‌تر برای شناسایی خطاها هستند [۱۶].

مدل‌های مبتنی بر حالت (زیردسته شماره ۲۵): یکی از انواع مدل‌های مبتنی بر معماری، مدل‌های مبتنی بر حالت است. هیچ کدام از مدل‌های مبتنی بر حالت، واسط بین اجزاء را در نظر نمی‌گیرند، این در حالی است که امکان دارد اجزاء به کمک تکنولوژی‌های پیشرفته مانند «احضار مِتُد از راه دور»، «فراخوانی رویه از راه دور» توزیع شده باشند. مدل معماری اجزاء مختلف از یک برنامه کاربردی ممکن است متفاوت باشد. اما در این دسته مدل‌ها، یک روش مدل‌سازی قابلیت اطمینان بر روی تمام اجزاء اعمال می‌گردد. در سطح سیستم برای استفاده از مدل‌های مبتنی بر حالت، از تحلیل زنجیره مارکوف زمان گسسته استفاده می‌شود (با پیچیدگی $O(n^3)$ که n تعداد حالت‌ها می‌باشد). سیستم‌های نرم‌افزاری بزرگ و پیچیده دارای هزاران حالت هستند، و به همین دلیل حل مدل مبتنی بر زنجیره مارکوف زمان گسسته برای آن‌ها بسیار هزینه‌بر خواهد بود. بنابراین مهم‌ترین ایراد این مدل‌ها این است که برای سیستم‌های بزرگ که می‌توانند صدها حالت داشته باشند، استفاده‌ی مکرر از این نوع آنالیز باعث اتلاف زمان زیادی می‌شود و کار طاقت فرسایی است [۳۳]. در برخی موارد نیز از مدل‌های مارکوف پنهان استفاده می‌شود (هنگامی که در مورد احتمال انتقال بعدی اطمینان وجود ندارد). اما در این حالت ماتریس گذار و ماتریس احتمال مشاهده (که نشان‌دهنده‌ی احتمال مشاهده‌ی یک رخداد در یک حالت خاص می‌باشد) به صورت تصادفی مقداردهی اولیه می‌شوند، که این امر دقت مدل‌سازی را پایین می‌برد [۸۵].

مدل‌های مبتنی بر مسیر و رویکرد افزوده (زیردسته‌های شماره ۲۶ و ۲۷): این دو مدل، زیر دسته‌های دیگر مدل‌های مبتنی بر معماری هستند. این مدل‌ها، در سطح سبک‌های معماری نرم‌افزار، به دلیل عدم وجود سیستم نرم‌افزاری برای بررسی مسیرهای ممکن اجراء و تخمین چگالی شکست وابسته به زمان سیستم، قابل استفاده نمی‌باشند. اما امکان استفاده از روش دیاگرام بلوکی قابلیت اطمینان، که یک روش مبتنی بر ساختار است، برای سنجش قابلیت اطمینان سبک‌های معماری نرم‌افزار وجود دارد [۸۶].

به طور کلی مدل‌های مبتنی بر معماری، که برای اندازه-گیری قابلیت اطمینان نرم‌افزار، معماری اجزاء نرم‌افزار را نیز در نظر می‌گیرند، برای مدل کردن قابلیت اطمینان سیستم‌های مبتنی بر جزء مناسب هستند. به مدل‌های مبتنی بر معماری در اصطلاح مدل‌های جعبه سفید نیز می‌گویند. معروف‌ترین مدل‌های موجود در این دسته را می‌توانید در مرجع [۳۳] پیدا کنید. در این مرجع انواع مدل‌های مبتنی بر معماری با هم مقایسه می‌شوند و در پایان نیز این نتایج حاصل می‌شود که هیچ کدام از مدل-هایی که تا کنون معرفی شده‌اند به تنهایی برای پیشبینی قابلیت اطمینان کافی نیستند. همچنین به دلیل نقایصی که این مدل‌ها دارند، در صنعت به ندرت از آن‌ها استفاده می‌شود. بر اساس این مقاله مهم‌ترین نقایص مدل‌های بررسی شده شامل کمبود ابزارهای پشتیبانی، عدم داشتن خاصیت تغییرپذیری و تحلیل ضعیف قابلیت اطمینان مولفه‌ها (به تنهایی) می‌باشد. همچنین هیچ استنباطی مبنی بر این که این روش‌ها به اندازه‌ی کافی کامل و مناسب هستند وجود ندارد (به دلیل عدم اعتبارسنجی و استفاده در صنعت). روش‌ها و مدل‌هایی که در این زمینه وجود دارند بر روی تحلیل و آنالیز سیستم از طریق محاسبات (مثلاً متوسط زمان بین شکست‌ها یا تعداد شکست‌ها در یک بازه‌ی زمانی) تمرکز دارند. حال آن که هدف از به کارگیری این مدل‌ها باید پر کردن شکاف بین معماری و نیازمندی‌ها باشد. یعنی باید بتوان از طریق تحلیلی که آن‌ها به ما ارائه می‌دهند رابطه‌ی بین معماری مناسب یا نا مناسب را با نیازمندی مربوطه پیدا و ردیابی کرد.

مدل‌های کاشت خطا (زیردسته شماره ۱۷): یکی از انواع مدل‌های مبتنی بر دامنه داده، مدل‌های کاشت خطا هستند. ایده‌ی اصلی در این مدل‌ها این است که مطمئن باشیم خطاهای معرفی شده، معرف و نمونه‌ای از تمام خطاهای محتمل در کُد برنامه باشند، در غیر این صورت نتایج بدست آمده دقیق نمی‌باشد. اما متأسفانه تاکنون مدل معتبر و قابل قبولی برای تضمین اینکه خطاهای انتخاب شده به عنوان هسته، این ویژگی را دارند یا خیر، معرفی نشده است. بنابراین بجای یافتن هسته‌هایی که معرف خطاها و اشکالات واقعی برنامه هستند، خطاها را در طول برنامه توزیع کرده با شناخت طبیعت خطاها و شرایط بروز آن‌ها، با انجام آزمایشاتی، خطاها و شکست‌های واقعی را تخمین می‌زنند. یکی دیگر از چالش‌های مطرح در زمینه‌ی کاشت خطا، تعیین تعداد خطاهایی است که باید به عنوان هسته انتخاب شوند. این موضوع به عوامل زیادی نظیر زبان برنامه‌نویسی مورد استفاده، تجربه‌ی برنامه‌نویسان، اندازه‌ی سیستم، دامنه‌ی نرم‌افزار بستگی دارد. به دلیل تنوع نتایج، بدست آوردن مدل توزیع خطای قطعی، غیر ممکن است. برخی از تحقیقات نشان داده‌اند که یکی از ایده‌های خوب در زمینه‌ی انتخاب تعداد خطاهای هسته، تجربه‌ی سازمان مورد نظر در رابطه با خطاها می‌باشد. در بسیاری از مواقع اعمال نتایج تجربیات یک سازمان برای سازمانی دیگر، بسیار دشوار می‌باشد [۷۰].

مدل‌های مبتنی بر شبکه پتری (زیردسته شماره ۱۸): یکی از زیردسته‌های مدل‌های شمارش خطا، مدل‌های مبتنی بر شبکه پتری هستند. این مدل‌ها علاوه بر اینکه دارای ساختار و رفتار صوری هستند، قابلیت نمایش گرافیکی نیز دارند که همین علت، مدل‌سازی توسط آن‌ها را آسان می‌کند. یکی از دلایل موفقیت شبکه‌های پتری سادگی آنهاست که البته این سادگی گاه مدل کردن سیستم‌های پیچیده را دشوار می‌سازد. استفاده از این مدل‌ها به عنوان ابزار اصلی یا مکمل در تخمین قابلیت اطمینان، می‌تواند مفید باشد [۷۴].

مدل‌های مارکوف (زیردسته شماره ۱۹): همان‌طور که در بخش ۳ بیان شد، یکی دیگر از انواع مدل‌های شمارش خطا، مدل‌های مارکوف هستند. این مدل‌ها تنها بر روی نرم‌افزارهایی قابل اعمال هستند که خصیصه‌ی مارکوف را ارضا کنند. مسئله‌ای که در این زمینه مطرح می‌شود این است که تعمیرات سیستم در عمل، با نرخ ثابتی انجام نمی‌شوند. تعمیرات و اصلاحات معمولاً یا با فاصله‌ی زمانی ثابتی از رخداد خرابی انجام می‌شوند، و یا به صورت دوره‌ای (بازرسی/تعمیر) صورت می‌گیرند. بنابراین این دسته از

عملیات به طور دقیق رفتار مارکوفی ندارند. این موضوع یکی از موانع استفاده‌ی گسترده از مدل‌های مارکوف در صنعت و نرم‌افزار است [۸۷ و ۸۸]. مدل‌های مارکوف فرض می‌کنند که انتقال کنترل بین اجزاء نرم‌افزاری به صورت فرآیند مارکوف است. این موضوع باعث می‌شود که این دسته از مدل‌ها برای مدل‌سازی نرم‌افزارهای احتمالی «مستقل از گذشته» بسیار مناسب باشند. همین موضوع باعث می‌شود که مدل‌سازی فرآیندهای قطعی به کمک این مدل‌ها دشوار باشد. در نرم‌افزارهای با مقیاس بزرگ، وجود پردازش‌هایی بی‌انتها (به عبارت دیگر با انتهای نامعلوم) بسیار محتمل است. مدل‌سازی این پردازش‌ها (که ممکن است قطعی و مستقل از گذشته باشند) به کمک مدل‌های مارکوف بسیار دشوار است. علاوه بر این مدل‌سازی نرم‌افزارهایی با فرآیندهای ترکیبی بی‌انتها بسیار دشوار است. این فرآیندها دارای تعداد نامحدودی مسیر اجرا هستند، اما هر کدام از این مسیرها پایان‌پذیر است [۸۹].

مدل‌های مبتنی بر تست (زیر دسته‌ی شماره ۶): مدل‌های قابلیت اطمینان نرم‌افزار که از معیار پوشش تست نیز استفاده می‌کنند، در سال‌ها اخیر به طور گسترده‌ای مورد مطالعه قرار گرفته‌اند [۶ و۶۲]. یکی از چالش‌هایی موجود این است که این مدل‌ها بنا به نوع نرم‌افزار و بستر مورد استفاده، از متریک‌های مختلفی- نظیر تعداد بلاک‌ها و انشعاب‌ها- برای اندازه‌گیری میزان پوشش تست استفاده می‌کنند. حال آنکه نبود یک روش واحد که از همه‌ی معیارهای گفته شده استفاده کند، احساس می‌شود. در واقع نیاز است تا این معیارها به طور یکپارچه و با هم برای مدل-های قابلیت اطمینان مبتنی بر میزان پوشش تست به کار گرفته شوند. در مرجع [۶۲] که در سال ۲۰۱۰ منتشر شده است، روشی بر این مبنا ارائه شده است که برای مطالعه‌ی بیشتر در این زمینه شما را به آن ارجاع می‌دهیم.

مدل‌های رشد قابلیت اطمینان (زیر دسته‌ی شماره ۵): به طور کلی مدل‌های رشد قابلیت اطمینان (زیر دسته-ی شماره ۵) در اصطلاح مدل‌های جعبه سیاه نامیده می-شوند؛ زیرا به نرم‌افزار به حالت کلی نگاه می‌کنند و سیستم را به حالت جزء به جزء در نظر نمی‌گیرند. در واقع مشکل اینگونه از سیستم‌ها همین مطلب است. زیرا تنها رفتار سیستم را نسبت به محیط خارجی مدل می‌کنند و رفتار داخلی سیستم را در نظر نمی‌گیرند. در واقع این مدل‌ها برای اندازه‌گیری قابلیت اطمینان یک سیستم بزرگ مبتنی بر جزء[۵۷] مناسب نیستند [۲۷].

اکثر مدل‌های رشد قابلیت اطمینان گذشته، با فرض محیط اشکال‌زدایی ایده‌آل در نظر گرفته شده‌اند. در این حالت فرض بر این است که هر اشکالی که در طی فرآیند تست و فاز عملیاتی مشاهده شد، به طور ایده‌آل حذف می‌شود. این در حالی است که اشکال‌زدایی در یک محیط عملیاتی و فرآیند تستِ واقعی، همیشه به طور ایده‌آل انجام نمی‌شود. از این رو مدل‌های رشد قابلیت اطمینان در محیط اشکال-زدایی غیر ایده‌آل مطرح می‌شوند. این دسته از مدل‌های رشد قابلیت اطمینان، قابلیت اطمینان را با دقت بالاتری تخمین می‌زنند. برای مطالعه‌ی بیشتر در این زمینه به مرجع [۶۱] مراجعه نمایید.

در مرجع [۹۰]، ۸ تا از مدل‌های رشد قابلیت اطمینان روی یک تعداد از پروژه‌های نرم‌افزاری بزرگ که در حوزه‌ی نرم-افزارهای نهفته قرار داشتند و مربوط به سه شرکت با دامنه-ی کاری متفاوت هستند، مورد ارزیابی قرار گرفته است. براساس نتایج حاصل از این ارزیابی، در صورتی که بخواهیم از قابلیت اطمینان به عنوان ابزاری در جهت تصمیم‌گیری در مورد اختصاص بهینه‌ی منابع تست استفاده کنیم، مدل گومپرتز[۵۸]، در صورتیکه در فرآیندهای توسعه‌ی نرم‌افزاری مبتنی بر مدلِ V و یا فرآیندهای توسعه‌ی کوچک و سریع بکار گرفته شود، بهترین انتخاب است. همچنین در این حالت در فرآیند توسعه آبشاری، بهتر است از مدل‌های منطقی[۵۹] استفاده شود. در راستای تصمیم‌گیری در مورد اختصاص بهینه‌ی منابع تست، با پیشبینی شکل پروفایل جریان نقص، مدل‌های منطقی، گومپرتز و مدل تاخیری S شکل برای پروفایل‌های جریانِ نقص S شکل، مقعر و محدب، بهترین انتخاب می‌باشند.

به منظور تعیین زمان انتشار نرم‌افزار، به طور کلی مدل‌های

روند[۶۰] (منطقی و گومپرتز) بهترین عملکرد را از لحاظ دقت مجانبی دارند (دقت مجانبی یک خصیصه‌ی مهم برای تعیین زمان انتشار است). هچنین مدل منطقی از بین مدل-های رشد قابلیت اطمینان، به منظور تعیین زمان انتشار، بهترین گزینه بوده است. البته در صورتیکه فرآیند توسعه آبشاری باشد، مدل گومپرتز پیشنهاد می‌شود. زمانیکه نمودار جریان نقص پیشینی شد، مدل منطقی بهترین انتخاب برای نمودار جریان نقص محدب و S شکل هست. این در حالی است که مدل گومپرتز بهترین انتخاب برای پروفایل نقص مقعر شکل است.

یکی دیگر از نتایج بدست آمده در مرجع [۹۰] برای مدل-های رشد قابلیت اطمینان نرم‌افزار، بهبود تعیین زمان انتشار با استفاده از نرخ رشد حاصل از پروژه-های گذشته می‌باشد. استفاده از اطلاعاتی نظیر نرخ رشد از پروژه‌های کامل شده‌ی قبل، این را نشان می‌دهد که دقت مجانبی در میان مدل‌ها در فرآیند توسعه‌ی مبتنی بر مدل V به طور قابل ملاحظه‌ای بهبود یافته‌است. در فرآیند توسعه‌ی آبشاری نیز این امر صادق است. اما در فرآیندهای توسعه‌ی نرم‌افزاری کوچک و سریع، استفاده از نرخ رشد، دقت مجانبی را بهبود نبخشیده است.

نتایج تحلیل: در مجموع با توجه به پیچیدگی نرم‌افزارها و همچنین مزایا و محدودیت‌های مطرح شده در جدول ۳، می‌توان به این نتیجه رسید که انتخاب یک مدل واحد به عنوان مناسب‌ترین مدل در راستای کاربردی خاص، تقریباً غیر ممکن است. به همین دلیل بهتر است چند مدل به کمک الگوریتم معرفی شده در مقاله انتخاب شده و به کمک آن‌ها قابلیت اطمینان نرم‌افزار اندازه‌گیری و توصیف گردد. البته مدل‌های دسته‌ی فرآیند نامتجانس پواسن از جمله مدل‌هایی هستند که در بسیاری از کاربردها مورد استفاده قرار می‌گیرند. این دسته از مدل‌ها برای مدل‌سازی قابلیت اطمینان سخت‌افزاری نیز قابل استفاده هستند. به همین دلیل می‌توان از این مدل‌ها برای پیش‌بینی قابلیت اطمینان سیستم (سخت‌افزار - نرم‌افزار) استفاده کرد. علاوه بر این، مدل‌های دسته‌ی فرآیند نامتجانس پواسن برای توصیف فرآیند خرابی که دارای رشد یا کاهش قابلیت اطمینان می‌باشد، بسیار مناسب هستند. این عامل باعث می‌شود که این مدل‌ها برای پیش‌بینی قابلیت اطمینان نرم‌افزار به سادگی قابل استفاده باشند. به همین دلیل مدل‌های دسته‌ی فرآیند نامتجانس پواسن یکی از دسته مدل‌های معروف و پرکاربرد در زمینه مدل‌سازی قابلیت اطمینان هستند. البته یکی از چالش‌های مطرح در رابطه با این مدل‌ها این است که برای مدل‌سازی قابلیت اطمینان نرم‌افزار به کمک مدل‌های فرآیند نامتجانس پواسن، معمولاً از فرضیاتی استفاده می‌شوند که ممکن است در عمل از واقعیت بسیار فاصله داشته باشند [۹۱].

۶- نتیجه‌گیری

در این مقاله یک مرور کلی بر روی مدل‌های اندازه‌گیری قابلیت اطمینان نرم‌افزار ارائه شده‌است. اکثر مقالاتی که در زمینه‌ی اندازه‌گیری و مدلسازی قابلیت اطمینان وجود دارند، به ارائه‌ی یک روش جدید و یا کامل کردن روش‌های قبلی می‌پردازند که این به نوبه‌ی خود بر پیچیدگی کار در این حوزه افزوده است. در واقع در این حوزه حجم بالایی از مدل‌ها ارائه شده است، لذا نیاز است کارهایی ارائه شود که به وسیله‌ی آن‌ها فهم و درک مدل‌های موجود آسان‌تر شود، تا کم‌کم این مدل‌ها از محیط آکادمیک و آزمایشگاهی وارد محیط کاربردی و صنعت شوند. این مقاله با معرفی روش-های موجود، تحلیل مزایا و محدودیت‌های آن‌ها و نیز ارائه-ی یک دسته‌بندی جامع و جدید، به سمت هرچه آسان‌تر کردن این مقوله و انتخاب مدل مناسب پیش رفته است. به علاوه، مزایا و محدودیت‌های هر زیر دسته از دسته‌بندی پیشنهادی به طور خلاصه معرفی و تحلیل شد. همچنین یک روش برای انتخاب مناسب‌ترین مدل قابلیت اطمینان ارائه گردید.

همان‌طور که در بخش ۵ بیان شد، مدل‌های فرآیند نامتجانس پواسنی، از دسته مدل‌های معروف و پرکاربرد در زمینه مدل‌سازی قابلیت اطمینان نرم‌افزار هستند. این مدل‌ها دارای محدودیت‌هایی نیز هستند که در جدول ۳ به

آن‌ها اشاره شده است. بهبود این مدل‌ها و برطرف کردن نقاط ضعف آن‌ها از طریق ترکیب با مدل‌های دیگر و نزدیک‌تر کردن فرضیات مدل به واقعیات، می‌تواند کاری ارزشمند برای آینده باشد. برخی دیگر از کارهای پیشنهادی برای آینده عبارتند از: ارائه‌ی یک تحلیل آماری از کارهای انجام شده در هر کدام از زیردسته‌های دسته‌بندی پیشنهادی، انجام مطالعاتی در زمینه‌ی انتخاب مدل و نیز انجام مطالعات اینچنینی در مورد زیر گروه خاصی از مدل-ها، انجام مطالعات موردی روی روش‌های مدل‌سازی ارائه شده و بررسی صحت مزایا و محدودیت‌های آن‌ها. انجام چنین کارهایی می‌تواند نقش بسزایی در به کارگیری این مدل‌ها در محیط‌های عملی ایفا نماید.

۷- ضمایم

جدول ۳: مزایا و محدودیت‌های برخی از دسته مدل‌های ارائه شده در شکل (۱۱)

پیش‌بینی سریع		
مزایا		۱- کاهش هزینه‌ها [۹۲] ۲- کمک به تصمیم‌گیری‌های به موقع [۲۹] ۳- قابل استفاده به کمک اطلاعات نسخه‌های قبلی نرم‌افزار یا محصولات مشابه [۲۹] ۴- قابل استفاده با استفاده از معیارهای طراحی [۳۰]
محدودیت ها		۱- محدودیت استفاده در بسیاری از کاربردها [۹۲] ۲- عدم اندازه‌گیری قابلیت اطمینان به صورت بسیار دقیق [۹۲] ۳- در دسترس نبودن اشکال و خرابی جهت اندازه‌گیری قابلیت اطمینان [۹۲]
مبتنی بر معماری		
افزوده	مزایا	۱- مناسب برای سیستم‌های نرم‌افزاری نامتجانس [۴۵] ۲- امکان مدل‌سازی قابلیت اطمینان هر جزء به کمک فرآیند نامتجانس پواسن [۹۳]
	محدودیت‌ها	۱- در نظر نگرفتن معماری نرم‌افزار به صورت صریح [۹۳] ۲- داشتن فرض مشخص بودن قابلیت اطمینان اجزاء [۴۵] ۳- در نظر نگرفتن وابستگی خرابی بین اجزاء و رابطهای بین آن‌ها [۴۵] ۴- دشواری و یا غیرممکن بودن بدست آوردن معماری نرم‌افزار در برخی موارد [۴۵] ۵- غیر قابل استفاده در سطح سبک‌های معماری نرم‌افزار [۸۶] ۶- داشتن فرض رشد قابلیت اطمینان [۴۵]
مبتنی بر حالت	مزایا	۱- مناسب برای سیستم‌های نرم‌افزاری نامتجانس [۴۵] ۲- در نظر گرفتن معماری نرم‌افزار به صورت صریح [۹۳] ۳- قابلیت استفاده حتی در صورت وجود مسیرهای بی‌نهایت در فضای حالت ساختار نرم‌افزار [۹۴]
مبتنی بر حالت	محدودیت‌ها	۱- زمان‌بر بودن تخمین پارامترها در سیستم‌های نرم‌افزاری بزرگ [۸۵] ۲- کم‌توجهی به اعتبارسنجی نتیجه‌ی پیش‌بینی قابلیت اطمینان [۸۵] ۳- امکان بهینه‌سازی پیش‌بینی قابلیت اطمینان به صورت محدود (تنها زمانی که معماری نرم‌افزار نیز قابل بهینه‌سازی باشد) [۸۵] ۴- استفاده از مدل مارکوف پنهان (در صورت عدم اطمینان از احتمال انتقال بعدی) [۸۵] ۵- در نظر نگرفتن طبیعت روابط بین اجزاء [۸۵] ۶- در نظر نگرفتن تغییر پویای معماری در هنگام عملکرد نرم‌افزار [۸۵]

ادامه جدول ۳

مبتنی بر حالت	محدودیت‌ها	۷- فرض داشتن خصیصه مارکوفی در انتقال کنترل بین ماژول‌ها [۹۳] ۸- داشتن فرض مشخص بودن قابلیت اطمینان اجزاء [۴۵] ۹- در نظر نگرفتن وابستگی خرابی بین اجزاء و رابطهای بین آن‌ها [۴۵] ۱۰- دشواری و یا غیرممکن بودن بدست آوردن معماری نرم‌افزار در برخی موارد [۴۵] ۱۱- در مورد مدل‌های سلسله مراتبی نقطه ضعف اصلی است که فقط یک تخمین ارائه می‌کنند و بنابراین به اندازه‌ی مدل‌های ترکیبی دقیق نیستند [۳۳].
مبتنی بر مسیر	مزایا	۱- مناسب برای سیستم‌های نرم‌افزاری نامتجانس [۴۵] ۲- در نظر گرفتن معماری نرم‌افزار به صورت صریح [۹۳]
	محدودیت‌ها	۱- غیر قابل استفاده در سطح سبک‌های معماری نرم‌افزار [۸۶] ۲- داشتن فرض استقلال شکست اجزاء [۹۳] ۳- معماری نرم‌افزار را به صوزت ترتیبی از اجزاء درنظر می‌گیرند [۹۳] ۴- داشتن فرض مشخص بودن قابلیت اطمینان اجزاء [۴۵] ۵- در نظر نگرفتن وابستگی خرابی بین اجزاء و رابطهای بین آن‌ها [۴۵] ۶- در نظر نگرفتن طبیعت واسط بین اجزاء [۸۵] ۷- عدم داشتن قابلیت برای محاسبه‌ی قابلیت اطمینان مسیرهای بی‌نهایت (حلقه‌ها) [۹۴] ۸- ناتوانی در محاسبه‌ی قابلیت اطمینان مسیرهای بی‌نهایت (حلقه‌ها) [۹۴]
رشد قابلیت اطمینان		
فرآیند نامتجانس پواسن	مزایا	۱- بیش‌ترین مدل استفاده شده در زمینه مدل‌سازی رشد قابلیت اطمینان [۹۵] ۲- سادگی تعیین معیارهای مهم نظیر پریود زمانی، تعداد شکست‌های باقیمانده، میانگین زمان بین شکست‌ها[۶۱] و میانگین زمان تا شکست[۶۲] [۹۶] ۳- قابلیت استفاده در زمانی که تمام خطاهای ذاتی برنامه شناسایی و اصلاح شوند [۹۷]
فرآیند نامتجانس پواسن	محدودیت‌ها	۱- غیر قابل استفاده بودن زمانی که تقریباً تمام خطاها شناسایی شده باشند [۹۵] ۲- بهترین قابلیت استفاده در مراحل اولیه و میانی تولید نرم‌افزار (به نسبت مرحله‌ی نهایی) [۹۵] ۳- داشتن فرض استقلال تعداد خطاهای مشاهده شده در هر فاصله زمانی، برای هر مجموعه محدود زمانی [۹۸] ۴- قابلیت تمایز خطاهای محدود و نامحدود [۹۸] ۵- حساس بودن تعداد کل شکست‌های ذاتی محاسبه شده توسط این مدل‌ها برای برنامه به داده‌ی زمان-مانده-تا-خرابی [۹۶] ۶- نیاز به ثابت بودن (تقریباً) تعداد کل شکست‌های ذاتی نرم‌افزار در حین افزایش تعداد خرابی‌ها (برای مدل‌سازی موفق بر روی سیستم‌های بحرانی-ایمن) [۹۷] ۷- نیاز به داشتن حداقل ۲۰ مجموعه داده خرابی [۹۷] ۸- استفاده از فرضیاتی که در عمل رعایت نمی‌شوند [۹۹]
نرخ خرابی	مزایا	۱- سادگی تعیین معیارهای مهم (مانند پریود زمانی، تعداد شکست‌های باقیمانده، میانگین زمان بین شکست‌ها و میانگین زمان تا شکست[۶۳] [۹۶] ۲- قابلیت استفاده در زمانی که تمام خطاهای ذاتی برنامه شناسایی و اصلاح شوند [۹۷]
	محدودیت‌ها	۱- حساس بودن تعداد کل شکست‌های ذاتی محاسبه شده توسط این مدل‌ها برای برنامه به داده‌ی زمان-مانده-تا-خرابی [۹۶] ۲- نیاز به ثابت بودن (تقریباً) تعداد کل شکست‌های ذاتی نرم‌افزار در حین افزایش تعداد خرابی‌ها (برای مدل‌سازی موفق بر روی سیستم‌های بحرانی-ایمن) [۹۷] ۳- نیاز به داشتن حداقل ۲۰ مجموعه داده خرابی (مفید بودن تنها در زمانی که به تعداد کافی داده خرابی موجود باشد) [۹۷] ۴- استفاده از فرضیاتی که در عمل محقق نمی‌شوند [۹۹]

ادامه جدول ۳

مدل‌های آماری		
مزایا		۱- دقت بالا در تخمین قابلیت اطمینان [۱۰۰] ۲- قابل استفاده به کمک اطلاعات آزمون‌های گذشته اجزاء نرم‌افزار [۱۰۱]
محدودیت‌ها		۱- نیاز به داده‌های فراوان برای بدست آوردن مدل آماری [۱۰۲] ۲- داده‌ها بعد از انجام تعدادی تست باید رشد قابلیت اطمینان را نشان دهند (در غیر اینصورت تحلیل مفید نخواهد بود) [۱۰۲]
مبتنی بر دامنه داده		
مبتنی بر دامنه ورودی	مزایا	۱- تخمین مستقیم احتمال درستی برنامه [۱۶]
	محدودیت‌ها	۱- داشتن فرض استقلال در انتخاب ورودی‌ها [۱۰۳] ۲- مدل‌سازی با فرض نداشتن اطلاعات اولیه از محصول تحت تست [۱۰۳]
کاشت خطا	مزایا	۱- وجود ابزارهایی که به کمک آن‌ها می‌توان خطاها و شکست‌های مورد نظر را به سیستم اعمال کرد [۸۹] ۲-کمک به تشخیص میزان توزیع شکست‌ها در محصول نرم‌افزاری [۸۹] ۳- قابل استفاده بودن در ارزیابی تعداد خطاهای باقی‌مانده در نرم‌افزار بعد از فاز تست سیستم [۷] ۴- تصمیم‌گیری در مورد انتشار محصول نرم‌افزاری، بر اساس نتیجه‌ی تکنیک کاشت خطا [۷]
	محدودیت‌ها	۱- زمان‌بر بودن اعمال این مدل بر روی نرم‌افزار [۸۹] ۲- پایین بودن دقت مدل‌سازی و اندازه‌گیری (به دلیل دخالت تصمیمات انسانی در انتخاب هسته‌ها) [۸۹] ۳- دشوار بودن تعیین تعداد خطاهایی که باید به عنوان هسته انتخاب شوند [۸۹] ۴- خطاهای معرفی شده باید معرف و نمونه‌ای از تمام خطاهای محتمل در کُد برنامه باشند [۸۹] ۵- سرراست و مستقیم نبودن اضافه کردن خطاها [۸۹] ۶- دشوار بودن اعمال نتایج تجربیات یک سازمان برای سازمانی دیگر [۸۹]
غیر شمارش خطا		
بیزین	مزایا	۱- اجازه دادن ترکیب اجراهای مختلف مشاهده شده در محیط عملکردی (حتی هنگامی که خرابی وجود ندارد) [۱۰۴]
	محدودیت‌ها	۱- عدم اندازه‌گیری تعداد خطاهای باقیمانده در نرم‌افزار (علاوه بر قابلیت اطمینان) [۱۶] ۲- کمبود ابزارهای مناسب در این زمینه [۲۸] ۳- دشواری یافتن اطلاعات مورد نیاز برای این مدل [۲۹]
قطعی	مزایا	۱- ساده‌تر بودن پیاده‌سازی و تفسیر این مدل‌ها، نسبت به مدل‌های تصادفی [۱۰۵]
	محدودیت‌ها	۱- افزایش مشخص قابلیت اطمینان و کاهش مشخص نرخ شکست (λ) به میزان مشخصی در اثر تصحیح یک خطا (اگر پارامترهای مدل شناخته شده باشند) [۱۶] ۲- عدم اندازه‌گیری تعداد خطاهای باقیمانده در نرم‌افزار (علاوه بر قابلیت اطمینان) [۱۶]
تصادفی	مزایا	۱- نزدیک‌تر بودن این مدل‌ها به واقعیت، نسبت به مدل‌های قطعی (به دلیل تصادفی بودن نرخ شکست خطاهای مختلف) [۱۶]
	محدودیت‌ها	۱- عدم اندازه‌گیری تعداد خطاهای باقیمانده در نرم‌افزار (علاوه بر قابلیت اطمینان) [۱۶] ۲- دشوار بودن پیاده‌سازی و تفسیر این مدل‌ها نسبت به مدل‌های قطعی[۱۰۵]

ادامه جدول ۳

شمارش خطا		
شبکه پتری	مزایا	۱- مدل کردن فرآیند تست و اشکال‌زدایی به کمک فرآیندهای «شمارش خطا» (در بیشتر مواقع) [۹۵] ۲- ساده بودن مدل [۷۴] ۳- قابل مدیریت بودن اندازه‌ی مدل [۷۴] ۴- مناسب برای مدل‌سازی وقایع پویا [۷۴] ۵- شهودی بودن مدل [۷۴] ۶- قابل تغییر و بهبود [۷۴] ۷- امکان تحلیل سیستم تحت شرایط مختلف [۷۴] ۸- امکان شناسایی تعارضات و اشکالات سیستم به صورت شهودی [۷۴]
	محدودیت‌ها	۱- نبود بسته نرم‌افزاری برای تحلیل قابلیت اطمینان به کمک این مدل [۷۴]
مارکوف	مزایا	۱- مدل کردن فرآیند تست و اشکال‌زدایی به کمک فرآیندهای «شمارش خطا» (در بیشتر مواقع) [۹۵] ۲- اندازه‌گیری تعداد خطاهای باقیمانده در نرم‌افزار [۱۶] ۳- امکان مدل‌سازی بسیاری از ابزارها و سیستم‌های دنیای واقعی (دارای نرخ خرابی ثابت) [۱۰۶] ۴- مدل‌سازی اصلاحات و تعمیر شکست‌ها (علاوه بر مدل کردن خرابی‌ها) [۱۰۶] ۵- امکان مدل سازی و تحلیل سیستم‌هایی با تمامیت بالا[۶۴] و مقاوم در برابر ا شکال برای ر سیدن به سطح مطلوبی از قابلیت اطمینان [۸۹] ۶- مناسب جهت مدل‌سازی نرم‌افزارهای احتمالی «مستقل از گذشته» [۸۹] ۷- مناسب برای استفاده در فاز تست سیستم نرم‌افزاری [۸۹] ۸- وجود مدل زنجیره مارکوف در بسیاری از ابزارهای نرم‌افزاری مهم [۱۰۷]
	محدودیت‌ها	۱- سختی مدل‌سازی فرآیندهای قطعی به کمک این مدل‌ها [۸۹] ۲- دشواری مدل‌سازی پردازش‌هایی با انتهای نامعلوم (در نرم‌افزارهای با مقیاس بزرگ) [۸۹] ۳- دشواری مدل‌سازی نرم‌افزارهایی با فرآیندهای ترکیبی بی‌انتها [۸۹] ۴- تعمیرات سیستم در عمل به طور دقیق رفتار مارکوفی ندارند (یکی از موانع استفاده‌ی گسترده از مدل‌های مارکوف در صنعت و نرم‌افزار) [۱۰۶].
قطعی	مزایا	۱- مدل کردن فرآیند تست و اشکال‌زدایی به کمک فرآیندهای «شمارش خطا» (در بیشتر مواقع) [۹۵] ۲- اندازه‌گیری تعداد خطاهای باقیمانده در نرم‌افزار [۱۶] ۳- مناسب برای اندازه‌گیری پیچیدگی برنامه و بررسی میزان سختی عملیات آزمون و اشکال‌زدایی نرم‌افزار [۷]
	محدودیت‌ها	۱- داشتن فرض ثابت بودن نرخ خرابی بین هر دو تصحیح متوالی (حتی اگر در نرم‌افزار تغییری حاصل شود) [۱۶] ۲- داشتن فرض افزایش مشخص در قابلیت اطمینان در اثر تصحیح یک خطا [۱۶] ۳- داشتن فرض کاهش مشخص نرخ شکست (λ) در اثر تصحیح یک خطا [۱۶]
بیزین	مزایا	۱- مدل کردن فرآیند تست و اشکال‌زدایی به کمک فرآیندهای «شمارش خطا» (در بیشتر مواقع) [۹۵] ۲- اجازه دادن ترکیب اجراهای مختلف مشاهده شده در محیط عملکردی (حتی هنگامی که خرابی وجود ندارد) [۱۰۴]
	محدودیت‌ها	۱- کمبود ابزارهای مناسب در این زمینه [۲۷] ۲- دشواری یافتن اطلاعات مورد نیاز برای این مدل [۲۷]

۸- مراجع

[1] I. Sommerville, (2010), "Software Engineering", 9th ed., Harlow, England: Addison-Wesley

[2] R. S. Pressman, (2009), "Software Engineering-A Practitioner's Approach-Required", 7th ed., New York, USA: McGraw-Hill

[3] International Organization for Standarization,(2001), "ISO Standard 826: Software Engineering – Product Quality, parts 1, 2 and 3", Geneve, Switzerland

[4] E. Hull, K. Jackson, and J. Dick,(2005), "Requirements engineering", 2nd ed., London, England: Springer

[5] H. Pham, (2007). "System software reliability", New York, USA: Springer-Verlag

[6] F. Febrero, C. Calero, M. A. Moraga, (2014), "A systematic mapping study of software reliability modeling," Information and software technology, Elsevier, vol. 56, no. 8, pp. 839-849.

[7] M. Rahmani, A. Azadmanesh,(2011), "Exploitation of Quantitative Approaches to Software Reliability", Tech. Rep. cst-2011-002, Computer Science, University of Nebraska, Omaha.

[8] Lyu, M. R., M. Xie, (1996), "Handbook of Software Reliability Engineering", 1st ed., New York, USA: McGraw-Hill.

[9] B. Kitchenham,(2004), "Procedures for performing systematic reviews", Keele, UK, Keele University, vol. 33, pp. 1-26.

[10] S. Keele,(2007), "Guidelines for performing systematic literature reviews in software engineering", Technical report, EBSE Technical Report EBSE-2007-01, pp. 1-57.

[11] P. Brereton, B. Kitchenham, D. Budgen, M. Turner, M. Khalil,(2007), "Lessons from applying the systematic literature review process within the software engineering domain", Journal of systems and software, vol. 80, no. 4, pp. 571-583.

[12] B. Kitchenham, P. Brereton, D. Budgen, M. Turner, J. Bailey, S. Linkman,(2009), "literature reviews in software engineering–a systematic literature review", Information and software technology, vol. 51, no. 1, pp. 7-15.

[13] B. Kitchenham, R. Pretorius, D. Budgen, P. Brereton, M. Turner, M. Niazi, S. Linkman,(2010), "Systematic literature reviews in software engineering–a tertiary study", Information and Software Technology, vol. 52, no. 8, pp. 792-805.

[14] F. Q. Da Silva, A. L. Santos, S. Soares, A. C. C. França, C. V. Monteiro, F. F. Maciel,(2011), "Six years of systematic literature reviews in software engineering: An updated tertiary study", Information and Software Technology, vol. 53, no. 9, pp. 899-913.

[15] J. Pan, (2014), "Software Reliability", http://users.ece.cmu.edu/~koopman/des_s99/sw_reliability/#metrics.

[16] C. V Ramamoorthy and F. B. Bastani, (1982), "Software Reliability - Status and Perspectives", Softw. Eng. IEEE Trans., vol. SE-8, no. 4, pp. 354–371.

[17] L. Shanmugam and L. Florence, (2012),"An Overview of Software Reliability Models", in International Journal of Advanced Research in Computer Science and Software Engineering, vol. 2, no. 10, pp. 36–42.

[18] J. G. Shanthikumar,(1983), "Software reliability models: a review", journal of Microelectronics and Reliability, Elsevier, vol. 23, no. 5, pp. 903–943.

[19] C. A. Asad, M. I. Ullah, and M. J. Ur-Rehman, (2004),"An approach for software reliability model selection", in Proceedings ofthe 28th Annual International Computer Software and Applications Conference, pp. 534–539.

[20] S. S. Gokhale, P. N. Marinos, and K. S. Trivedi,(1996), "Important milestones in software reliability modeling", in 8th International Conference on Software Engineering and Knowledge Engineering (SEKE), Nevada, USA, pp. 345–352.

[21] A. L. Goel,(1985), "Software reliability models: Assumptions, limitations, and applicability", Softw. Eng. IEEE Trans., no. 12, pp. 1411–1423.

[22] X. Li, (2002), "Software reliability measurement: a survey", MSc. Thesis, Dept. Computer Science & Software Engineering, Concordia University.

[23] Z. KRAJČUŠKOVÁ,(2007), "Software reliability models," Proceeding of 17th International Conference on Radioelektronika, IEEE, pp.1-4.

[24] H. Pham, M. Pham,(1991), "Software reliability models for critical applications," Technical report, Idaho National Engineering Laboratory, EG&G-2663.

[25] P.-H. Seong, (2009), "Reliability and Risk Issues in Large Scale Safety-critical Digital Control Systems", Springer, 1st ed., Berlin, Germany, pp. 85–87.

[26] P. C. J. P. K. Kapur, H. Pham, A. Gupta, (2011), "Software Reliability Assessment with OR Applications", 1st ed., London, England: Springer.

[27] A. K. Pandey, N.K. Goyal, (2013), "Early Software Reliability Prediction a Fuzzy Logic Approach", Springer.

[28] N. Fenton, M. Neil, and D. Marquez,(2008), "Using Bayesian networks to predict software defects and reliability", Proc. Inst. Mech. Eng. Part O J. Risk Reliab., vol. 222, no. 4, pp. 701–712.
[29] Q. P. Hu, Y.-S. Dai, M. Xie, and S. H. Ng, (2006),"Early software reliability prediction with extended ANN model", in Computer Software and Applications Conferenc, COMPSAC'06. 30th Annual International, vol. 2, pp. 234–239.
[30] S. Mohanta, G. Vinod, A. K. Ghosh, and R. Mall,(2010), "An approach for early prediction of software reliability", ACM SIGSOFT Softw. Eng. Notes, vol. 35, no. 6, pp. 1–9.
[31] C. Smidts, M. Stutzke, and R. W. Stoddard, (1998),"Software reliability modeling: an approach to early reliability prediction", Reliab. IEEE Trans., vol. 47, no. 3, pp. 268–278.
[32] Rome Laboratory, (1987),"Methodology for Software Reliability Prediction and Assessment", Tech. Rep. RADC-TR-87-171.
[33] A. Immonen, E. Niemela, (2007),"Survey of reliability and availability prediction methods from the viewpoint of software architecture," Software & System Modeling, Springer, vol. 7, no. 1, pp. 49-65.
[34] S. S. Gokhale and K. S. Trivedi,(2006), "Analytical models for architecture-based software reliability prediction: A unification framework", Reliab. IEEE Trans., vol. 55, no. 4, pp. 578–590.
[35] R. C. Cheung, (1980),"A user-oriented software reliability model", Softw. Eng. IEEE Trans., no. 2, pp. 118–125.
[36] P. Kubat, (1989),"Assessing reliability of modular software", Oper. Res. Lett., vol. 8, no. 1, pp. 35–41.
[37] B. Littlewood, (1979), "software reliability model for modular program structure", IEEE Trans. Reliability, vol. 28, no. 3, pp. 241-246.
[38] R.H. Reussner, H.W. Schimidt, I.H. Poernomo, (2003), "Reliability prediction for component-based software architectures" , J.Sysetem Softw., vol. 66, no. 3, pp. 241-252.
[39] G.N. Rodrigues, D.S. Rosenblum, S. Uchitel,(2005), "Using scenarios to predict the reliability of concurrent component-based software systems " ,in Proceedings of the 8th international conference on Fundamental Approaches to Software Engineering, pp. 111-126.
[40] M.G. Thomason, J.A. Whittaker, (1999),"Rare failure-state in a Markov chain model for software reliability" ,in Proceedings of the 10th international symposiom on Software Reliability Engineering, pp. 12-19.
[41] S. S. Gokhale and K. S. Trivedi,(2002), "reliability prediction and sensitivity analysis based on software architecture", in Proceedings of the 3rd international symposiom on Software Reliability Engineering, pp. 64-75.
[42] M. L. Shooman,(1976), "Structural models for software reliability prediction",in Proceedings of the 2nd international conference on Software engineering, pp. 268–280.
[43] S. Krishnamurthy and A. P. Mathur, (1997),"On the estimation of reliability of a software system using reliabilities of its components", in Proceedings of 8th International Symposium on Software Reliability Engineering, pp. 146–155.
[44] S. M. Yacoub, B. Cukic, and H. H. Ammar,(1999), "Scenario-based reliability analysis of component-based software", in Proceedings of 10th International Symposium on Software Reliability Engineering, pp. 22–31.
[45] K. Goševa-Popstojanova and K. S. Trivedi, (2001), "Architecture-based approach to reliability assessment of software systems", Journal of Performance Evaluation, Elsevier, vol. 45, no. 2, pp. 179–204.
[46] M. Xie and C. Wohlin,(1995), "An additive reliability model for the analysis of modular software failure data", in Proceedings of 6th International Symposium on Software Reliability Engineering, pp. 188–194.
[47] W. W. Everett,(1999), "Software component reliability analysis", in Proceedings of 3rd IEEE Symposium on Application-Specific Systems and Software Engineering and Technology, pp. 204–211.
[48] M. Klein, et al., (1999),"Attribute-based atchitecture styles", in Proceedings First Working IFIP Conference on Software Architecture, pp. 225-243.
[49] L. Chung, B. Nixon, E. Yu, J. Mylopoulos,(2000), "Non-functional Requirements in Software Engineering". Kluwer Boston, Dordrecht 439.
[50] R. Kazman, G. Abowd, L. Bass, P. Clements, (1996),"Scenario-based analysis o software architecture " , IEEE Softw., vol. 13, no. 6, pp. 47-55.
[51] N. Lassing, D. Rijsenbrij, H. van Vilet, (1999),"On software architecture analysis of flexibility, complexity of changes: size isn't everything", in Proceedings of the 2nd Nordic Software Architecture Workshop.
[52] G. Molter,(1999), "Integrating SAAM in Domain-centric and Reuse-based development process", in Proceedings of the 2nd Nordic Software Architecture Workshop.
[53] R. Kazman, et al., (1999),"The architecture tradeoff analysis method", in Proceedings of the 4th IEEE International Conference on Engineering of Complex Computer Systems, pp. 68-78.
[54] S. S. Gokhale and K. S. Trivedi, (1999),"A time/structure based software reliability model", Annals of Software Engineering, vol. 8, no. 1–4, pp. 85–121.

[55] S. S. Gokhale, P. N. Marinos, and K. S. Trivedi,(1996), “Important milestones in software reliability modeling”, in 8th International Conference on Software Engineering and Knowledge Engineering (SEKE), Nevada, USA, pp. 345–352.
[56] H. A. Stiber, (2007), “A family of software reliability growth models”, in Proceeding of 31th Annual International Computer Software and Applications Conference, IEEE, vol. 2, pp. 217-224.
[57] H. Pham, (2003),“Software reliability and cost models: Perspectives, comparison, and practice”, Eur. J. Oper. Res., vol. 149, no. 3, pp. 475–489.
[58] H. Pham and X. Zhang,(1997), “An NHPP software reliability model and its comparison”, Int. J. Reliab. Qual. Saf. Eng., vol. 4, no. 03, pp. 269–282.
[59] Z. Jelinski and P. Moranda, (1972), “Software reliability research”, Statistical Computer Performance Evaluation, pp. 465–484.
[60] A. L. Goel and K. Okumoto, (2009),“Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures”, in IEEE Transactions on Reliability, pp. 206 – 211.
[61] S. Yamada, (2014), "Software reliability modeling Fundamental and Applications", Japan, Springer.
[62] Jinxia An, Jihong Zhu, (2010),“ Software reliability modeling with integrated test coverage,” in Proceeding of Fourth International Conference on Secure Software Integration and Reliability Improvement, IEEE, pp. 106-112.
[۶۳] آ. عیسی زاده و ح. جهانگیری، (۱۳۸۶)، "مدل‌سازی آماری قابلیت اطمینان نرم افزار"، ر اولین کنفرانس بین المللی تحقیق در عملیات ایران، جزیره کیش، ایران.
[64] T. A. Thayer, G. R. Craig, L. E. Frey, W. L. Hetrick, and M. Lipow, (1976), “Software reliability study”, Tech. Rep. ADA030798.
[65] S. Bologna and W. Ehrenberger, (1978),“Applicability of statistical models for reactor safety software verification”, ENEA Internal report: ENEA RT 47/78, Roma, Italy.
[66] J. R. Brown and M. Lipow, (1975),“Testing for software reliability”, in ACM SIGPLAN Notices, vol. 10, no. 6, pp. 518–527.
[67] M. H. Halstead, (1977),“Elements of software science”, Elsevier, New York, USA.
[68] E. Nelson, (1978), “Estimating software reliability from test data”, Microelectron. Reliab., vol. 17, no. 1, pp. 67–73.
[69] S. N. Weiss and E. J. Weyuker, (1988),“An extended domain-based model of software reliability”, Softw. Eng. IEEE Trans., vol. 14, no. 10, pp. 1512–1524.
[70] F. Grigorjev, N. Lascano, and J. L. Staude, (2003),“A Fault Seeding Experience”, in Proceedings of the 4th Argentine Symposium on Software Engineering (ASSE), Buenos Aires, Argentina.
[71] H. D. Mills, (1972),“On the statistical validation of computer programs”, IBM Fed. Syst. Div. Rep., pp. 72–6015.
[72] K.-Y. Cai, (1998),“On estimating the number of defects remaining in software”, J. Syst. Softw., vol. 40, no. 2, pp. 93–114.
[73] Y. Tohma, H. Yamano, M. Ohba, and R. Jacoby,(1991), “The estimation of parameters of the hypergeometric distribution and its application to the software reliability growth model”, IEEE Trans. Softw. Eng., vol. 17, no. 5, pp. 483–489.
[74] V. Volovoi,(2004), “Modeling of System Reliability Using Petri Nets with Aging Tokens”, J. Reliab. Eng. Syst. Saf., vol. 84, pp. 149–161.
[75] M. Xie, K.-L. Poh, and Y.-S. Dai, (2004), "Computing System Reliability: Models and Analysis", 1st ed., New York, USA: Springer.
[76] M. Ohba, (1984),“Software reliability analysis models”, IBM J. Res. Dev., vol. 28, no. 4, pp. 428–443.
[77] N. B. Fuqua, (2014), “The Applicability of Markov Analysis Methods to Reliability, Maintainability, and Safety”. [Online]. Available: http://theriac.org/DeskReference/viewDocument.php?id=201.
[78] G. J. Schick and R. W. Wolverton,(1973), “Assessment of software reliability”, in Proceedings of the Vortrage der jahrestagung dgor/papers of the annual meeting, Springer, New York, NY, USA, pp. 395–422.
[79] M. L. Shooman, (1972),"Probability Models for Software Reliability Prediction", Statistical Computer Performance Evaluation, W. Freiberger (ed.). New York: Academic Press.
[80] J. D. Musa, (1975),“A theory of software reliability and its application”, Softw. Eng. IEEE Trans., no. 3, pp. 312–327.
[81] G. J. Schick and R. W. Wolverton,(1978), “An analysis of competing software reliability models”, Softw. Eng. IEEE Trans., no. 2, pp. 104–120.
[82] P. B. Moranda,(1975), “Prediction of software reliability during debugging”, in Proceedings of Annual Reliability and Maintainability Symposium, no. JAN 28, pp. 327–332.
[83] C. V. Ramamoorthy and F. B. Bastani, (1980),“Modelling of the software reliability growth process”, in COMPSAC, pp. 161–169.

[84] B. Littlewood and J. L. Verrall,(1973), “A Bayesian reliability growth model for computer software”, IEEE Symp. Computer Software Reliability, New York, USA, pp. 70-77.
[85] L. K. Singh, A. K. Tripathi, and G. Vinod, (2011),“Software reliability early prediction in architectural design phase: Overview and Limitations”, J. Softw. Eng. Appl., vol. 4, p. 181.
[۸۶] غ. شاه‌محمدی و س. جلیلی، (۱۳۸۶)، "ارزیابی کمی سبک‌های معماری نرم‌افزار از دید قابلیت اعتماد با رویکرد مدل سازی دیاگرام بلوکی"، سیزدهمین کنفرانس ملی انجمن کامپیوتر ایران، جزیره کیش، ایران.
[87] “Markov Modeling for Reliability”، http://www.mathpages.com/home/kmath232/part1/part1.htm.
[۸۸] ح. عبدالملکی، م. رزازی و ا. فراهی، (۱۳۹۰)، "ارایه روشی برای ارزیابی قابلیت اطمینان معماری نرم افزار مبتنی بر مثال سبک ترکیبی لایه‌ای و لوله و فیلتر با نگاشت مدل شبکه پتری درزنجیره مارکف"، اولین همایش تخصصی سیستمهای هوشمند کامپیوتری و کاربردهای آنها، تهران، ایران.
[89] W.-L. Wang and M.-H. Chen, (2002),“Heterogeneous software reliability modeling”, in Proceedings of 13th International Symposium on Software Reliability Engineering, pp. 41 – 52.
[90] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, F. Torner, W. Meding, C. Hoglund,(2014), “Selecting software reliability growth models and improving their predictive accuracy using historical projects data”, System and Software, Elsevier, vol. 98, pp. 59–78.
[91] R. Lai and M. Garg,(2012), “A Detailed Study of NHPP Software Reliability Models (Invited Paper)”, J. Softw., vol. 7, no. 6, pp. 1296–1306.
[92] P. Lakey and A. Neufelder, (1997), “System and software reliability assurance notebook”, Rome Lab., New York, USA, Tech. Rep. RL-TR-97-XX.
[93] K. Goševa-Popstojanova and K. S. Trivedi,(2003), “Architecture-based approaches to software reliability prediction”, in international journal of computer mathematics with applications, vol. 46, no. 7, pp. 1023–1036.
[94] S. S. Gokhale and M.-T. Lyu,(2005), “A simulation approach to structure-based software reliability analysis”, Softw. Eng. IEEE Trans., vol. 31, no. 8, pp. 643–656.
[95] K.-C. Chiu, Y.-S. Huang, and T.-Z. Lee, (2008), “A study of software reliability growth from the perspective of learning effects”, in international journal of Reliability Engineering & System Safety, vol. 93, no. 10, pp. 1410–1421.
[96] B. Anniprincy and S. Sridhar, (2014),“An Improved Software Reliability Growth Model”, Int. J. Comput. Eng. Res., vol. 4, no. 2, pp. 22–28.
[97] K. M. Cheol, J. S. Cheol, and J. J. Ha, (2007),“Possibilities and Limitations of Applying Software Reliability Growth Models to Safety- Critical Software”, in journal of Nuclear Engineering and Technology, vol. 39, no. 2, pp. 129–132.
[98] V. Almering, M. van Genuchten, G. Cloudt, and P. J. M. Sonnemans, (2007), “Using software reliability growth models in practice”, Software, IEEE, vol. 24, no. 6, pp. 82–88.
[99] C. Stringfellow and A. A. Andrews,(2002), “An empirical method for selecting software reliability growth models”, Empir. Softw. Eng., vol. 7, no. 4, pp. 319–343.
[100] DOES Inc., (2014),“Statistical Design of Experiments (DOE)”, http://www.doesinc.com/knowledge.htm.
[101] S. Söhnlein and F. Saglietti,(2014), “Software Reliability Assessment by Statistical Analysis of Operational Experience”, http://ercim-news.ercim.eu/en75/special/software-reliability-assessment-by-statistical-analysis-of-operational-experience.
[102] M. A. A. Boon, E. Brandt, I. C. Ramos, A. Di Bucchianico, and R. Henzen,(2007), “A New Statistical Software Reliability Tool”, in Proceedings of VVSS2007-verification and validation of software systems, pp. 125–139.
[103]B. Cukic, E. Gunel, H. Singh, and G. U. O. Lan, (2003), “The theory of software reliability corroboration”, IEICE Trans. Inf. Syst., vol. 86, no. 10, pp. 2121–2129.
[104] I. Karanta, (2006),“Methods and problems of software reliability estimation”, VTT WP, vol. 63, p. 57.
[105] “Deterministic simulation”, 2014. http://en.wikipedia.org/wiki/Deterministic_simulation.
[106] K. Sharma, R. Garg, C. K. Nagpal, and R. K. Garg, (2010),“Selection of Optimal Software Reliability Growth Models Using a Distance Based Approach”, in IEEE Transactions on Reliability, pp. 266–276.
[107] R. V. Melnyk, (2014), “Petri Nets: An Alternative to Markov Chains”. [Online]. Available: http://www.theriac.org/DeskReference/viewDocument.php?id=80.